HDMI video cable eavesdropping

Jonas Tjomsland

Mini-project in Digital Signal Processing

May 19, 2020

1 Introduction

The electromagnetic emanations from wires, computers and other and electronic equipment was observed many years before Maxwell's equations helped mathematically describe the behaviour of electric and magnetic fields and the relationship between them. The interference these emanations can lead to is well known, it was a common cause for the disturbance that often occurred on older TVs and listeners of classic FM radio have most likely experienced noise resulting from this. However, fewer people are aware of the possibility to eavesdrop these emanations and extract information from the electronic devices and connected wires that emits them.

In this project we investigate IQ down-converted recordings, gathered with an antenna facing a Raspberry Pi computer connected with an HDMI cable to a LCD video screen. With the goal of reconstructing the image displayed on the screen, several approaches and measures introduced in related work has been implemented. The results show that we are able to recover most of the information displayed on the target screen, further confirming that electromagnetic eavesdropping is a substantially security risk even for the modern day computer equipment.

2 Related work

The first public work investigating the eavesdropping risk and opportunities that electromagnetic emanations opens up for was published by Wim van Eck in 1985 [1]. Work like this had already been conducted for military applications, but NSA's "TEMPEST" work [2] on compromising emanations was classified for many years. Wim van Eck's contribution showed that with cheap, of-the-shelf equipment, simple eavesdropping devices could be built to reconstruct information from target devices at a substantial distance.

It took several years before van Eck's work was extended. Markus Kuhn and Ross Anderson showed how electromagnetic radiation from devices can be controlled and leveraged for both attack and defense by hiding information in dither patterns [3]. Following this, Kuhn investigated the electromagnetic leakage from electronic devices even further in his technical report, *Compromising emanations: eavesdropping risks of computer displays* [4]. Through several real-world experiments Kuhn showed that van Eck's findings could be modified to work on modern day computer equipment as well. In addition, Kuhn presented an optical eavesdropping technique leveraging the variations of light emitted from CRT displays. There is limited work by other researchers on the topic, however in 2012 Elibol et al. published work where they develop a more mobile version of an eavesdropping device, able to reconstruct information from up to 50m [5]. They also present a method to automate the process of determining the synchronization frequency, this is crucial when no such information about the target device is available.

3 Methodology

In raster based video monitors, information is displayed in a sequential manner, assigning a light intensity to every pixel starting from the top left corner and line by line ending up in the lower right corner. The pixel intensity is determined by the signal strength at a specific point in time which is controlled by the internal pixel clock rate of the video controller. If we are able to match this clock rate with our sample rate and extract the signal, one pixel at a time, we should be able to reconstruct the information displayed on the target device. The challenge with this approach is to perfectly match the video controller's internal pixel rate. As Kuhn states [4], if the horizontal pixel rate is off by only 1 ppm the displayed information will roll to the right with a substantial speed.

We are given all the information needed about the target device, but the given pixel clock frequency is not accurate enough. If we use the given value of 25.175 MHz to resample the IQ signal from 64 MHz using linear interpolation, before constructing an image as a pixel matrix, we are not able reconstruct the image displayed on the target device. However, after some manual tuning we are quite quickly able to rotate the displayed image and see something meaningful. In Figure 1 three stages of manually tuning the pixel frequency is presented, using recording from a center frequency of 425MHz.

Figure 1: Three stages of manually tuning the pixel clock frequency for one frame.

This is for one frame only and to increase accuracy we look at the shift between frame 1 and 10, using this visual shift the accuracy of the pixel frequency can be further improved. The result after some fine tuning allows us to average the pixel intensities over different frames to enhance the reconstructed image, see Figure 2.

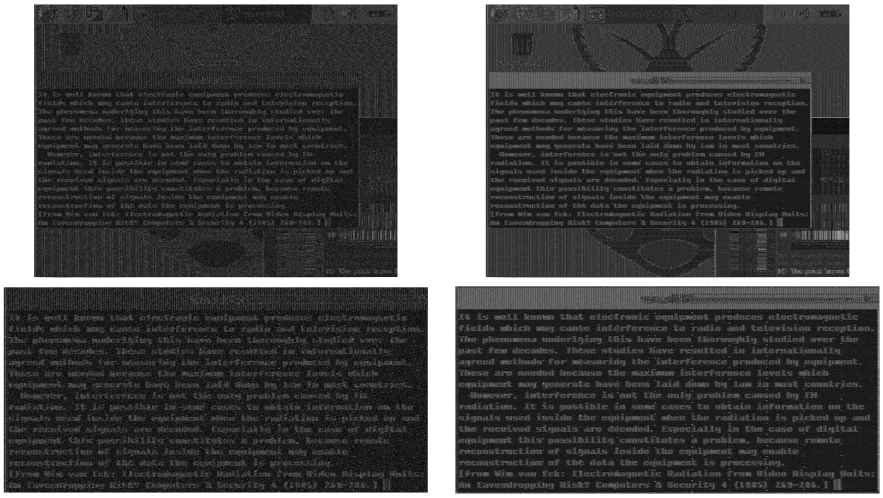

Figure 2: The reconstructed image after manual tuning of the pixel clock frequency (left) and the same image averaged over 20 frames (right).

This manual tuning process can be quite tedious and will at the same time not allow for the degree of accuracy needed in our estimate of the pixel rate. To both automate the tuning process and increase the accuracy we can leverage the fact that specific pixel intensities between consecutive frames should be the same, given that the same image always is displayed. Knowing this, we can look at the discrete auto-correlation of the IQ signal which is defined as,

$$R_{IQ,IQ}(j) = \sum_{i=-\infty}^{\infty} IQ_i \overline{IQ}_{i-j}, \tag{1}$$

where $\overline{IQ}$ is the complex conjugate of the IQ signal and j is the lag between samples. In our case we look at the amplitude modulated signal and therefore $\overline{IQ} = IQ$. This gives an indication of how similar the signal is with itself at a later point in time (j samples later). Using the estimate for the pixel clock frequency from the manual tuning above, the number of samples in each single frame is,

$$n_{samples} = x * y * \frac{f_s}{f_p}, \tag{2}$$

where x is the number of pixels on each line, y the number of lines, f_s the sampling frequency and f_p the pixel clock rate. This means that if our f_p is exact then we should see a peak in the auto-correlation plot at $n_{samples}$ later. Most likely, it is not accurate and we are likely to find the peak to differ from our estimate, we can use this difference to update our estimate of f_p. To reach the highest accuracy possible we can look at the cross-correlation between the first and the last frame in the IQ signal. However, since even the smallest inaccuracy in our estimate leads to the displayed image rolling to the right at a substantial speed we have to start investigating the cross correlation between two frames closer to each other. To automate this process I implement the following algorithm:

Algorithm 1: Automated determination of pixel clock rate

for *"Frame Jump" in [2, 5, 10, 20, 50, 59]* **do**
- Estimate the number of samples in a frame using f_p
- Estimate the samples on each line
- Take the 50 first lines of the first frame
- Take the 50 first lines of frame at "Frame Jump"
- Look at the cross correlation between the two
- Extract the index of the peak and look at the difference from the middle
- Update the estimate of f_p using the observed number of samples between the frames

end

Since we look at the cross-correlation between two frames we would expect the peak in the correlation plot to be at the middle index if the correct number of samples between the frames are used. As mentioned above, our manual tuned estimate is not accurate enough and we find the peak to differ from the middle index with a specific number of samples. That difference tells us how many samples there actually are between two frames and we use that to update our f_p as follows,

$$f_p = \frac{f_s y (F_{jump} + 1)}{(n_{samples}(F_{jump} + 1)d)x}, \tag{3}$$

where F_{jump} is the frame we calculate the cross-correlation with and d is the number of samples between the middle index and the cross-correlation peak. The method can be further investigated in the attached source code. In Figure 3 the cross-correlation plot between the first and the last frame is presented. This is from the last iteration of Algorithm 1 and the resulting pixel clock frequency obtained with the described method is 25200096.862872317 MHz.

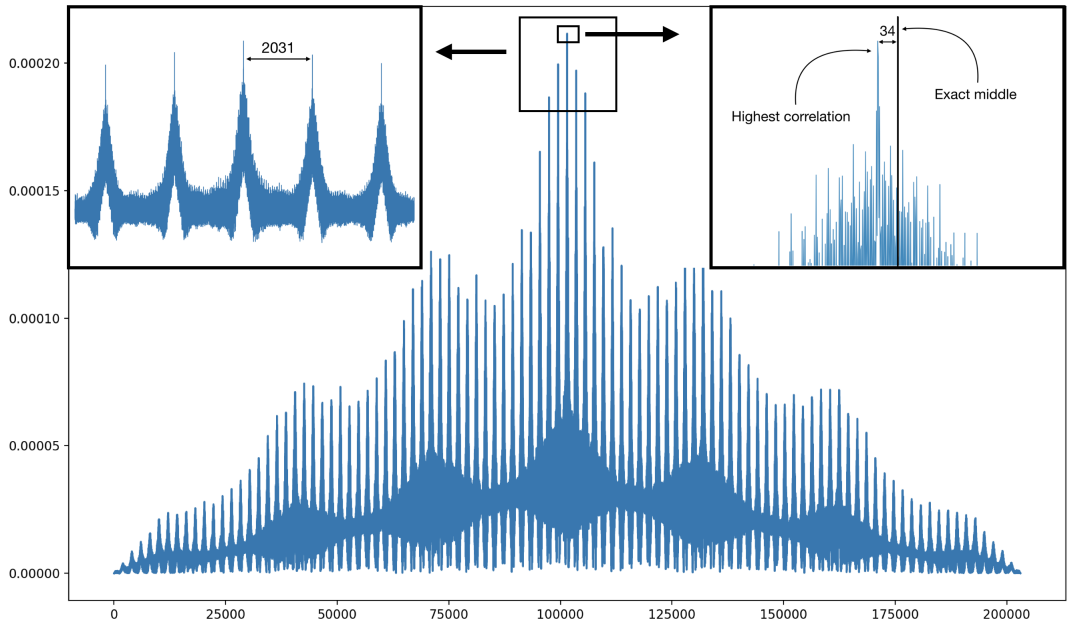

Figure 3: The cross-correlation between the first and the last frame after several steps of automatic pixel frequency tuning.

We see from Figure 3 that in the last iteration of Algorithm 1 there was only a difference of 34 samples between our estimate and the actual number of samples between the frames. Moreover, the pixel clock rate is not the only thing we can determine using the cross-correlation of the signal. We also see that the plot consists of numerous peaks which in this case represents pixel lines in the displayed image. This periodic pattern is a result of the repeating blanking intervals occurring every line and we can use them to further validate our line frequency. Using our values for f_p, f_s and x, the number of samples per line should be $n_{line} = x\frac{f_s}{f_p} = 2031$. In the upper left corner of Figure 3 another zoomed in version of the plot can be seen and the number of samples between the peaks are indicated.

For this project, we were given the approximate pixel rate, f_p, pixels per line, x and lines per frame, y, the automatic detection of these parameters are therefore not as crucial as it would be in a real-world applications of this. In a real-world attack such information about the target device would most likely not be available and necessary to determine. The correlation approach described above would then be very useful, approximate values of f_p and y could be inferred from the repeating patterns in the signal, x on the other hand would be more difficult to deduct due to the fact that it could be a continuous signals and the number of pixels could vary. However, if f_p and y is determined, traditional video modes, like those listed by VESA [6], can be used to determine the most likely mode running in the target device.

So far we have just amplitude demodulated the IQ signal by assigning each pixel the absolute value of the signal at every sample. This discards all phase and polarity information included, information which could have assisted in indicating edges in the displayed image. An alternative to AM demodulation is to look at the imaginary and real part of each IQ sample separately and display them either in different color channels (RGB) or as brightness and hue (HSV). In Figure 4 the result of such an approach can be seen.

Figure 4: Reconstructed image where imaginary and real parts of the IQ signal is placed in different color channels (left) or as hue and brightness (right).

As we can see in the figure above, the separation of the real and imaginary parts in different channels didn't automatically lead to a better reconstructed image. However, we can further investigate this approach by trying to "unrotate" the periodic phase change and make sure pixels maintain their phase even across multiple frames. We do so by looking at the spectrogram of the IQ data, trying to identify frequency peaks. By multiplying the signal with a complex phasor of frequency f_{peak} we can shift the peak to zero and thus obtain the unrotated signal. We will find peaks at integer multiples, k, of the pixel clock frequency f_p and when we use the 425MHz center frequency we observe one at $17 * f_p$ and one at $16 * f_p$. Initially I shifted the spectrum such that the peak at $k = 17$ ended up at zero and and averaged the imaginary and real part of the IQ signal separately over 50 frames before visualizing it as a HSV image. The resulting image was not very good and rather blurry. This lead me to try shifting the other peak visible in the 40MHz bandwidth spectrum before repeating the averaging. The resulting reconstructed image from this approach is presented in Figure 5.

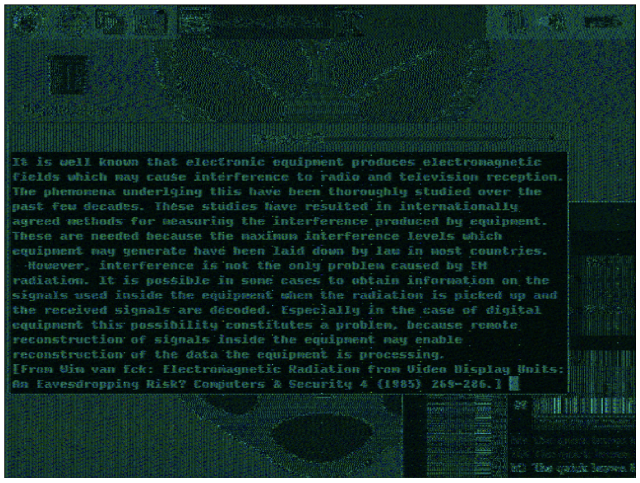

Figure 5: The reconstructed image after phase unrotation and displaying the polar coordinates of the complex numbers as brightness and hue, averaged over 50 frames.

If we compare the averaged figure from Figure 2 with Figure 5 we see that the phase information in fact can assist in indicating edges slightly better than when it is discarded in AM demodulation. The text displayed is a bit sharper and easier to read, however, the overall reconstructed image is not substantially better than with the initial approach.

All of the above experiments have been conducted using a center frequency of 425MHz. We where also given recordings centered at other frequencies and I investigated some of these to validate whether the initial center frequency used really gives the best result. Below is the result from two other center frequencies, 225MHz and 325MHz.

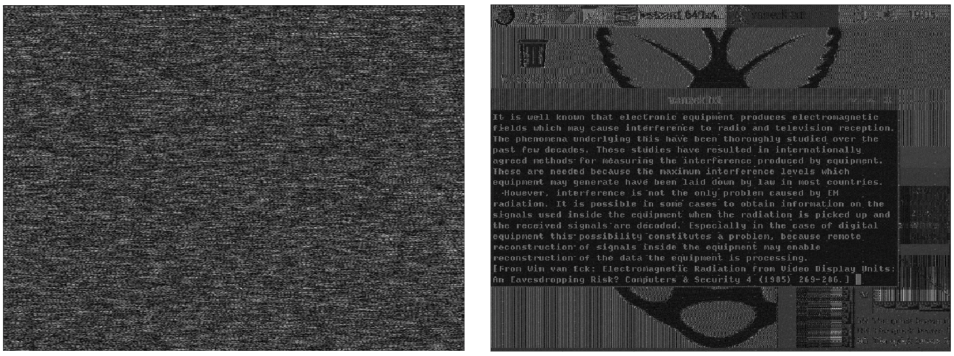

Figure 6: Image reconstructed using recordings with a center frequency of 225MHz (left) and 325MHz (right).

We see from Figure 6 that the image reconstructed from the 225MHz center frequency is too noisy to observe any information. This could be related to the "BBC National DAB ensemble" DAB channel being transmitted at a very close frequency [7]. The image reconstructed from the 325MHz center frequency on the other hand is very good. In fact, if we compare it to the initial reconstructed image where the 425MHz center frequency was used, we see that the quality is substantially better. See Figure 7 for a comparison between the two.

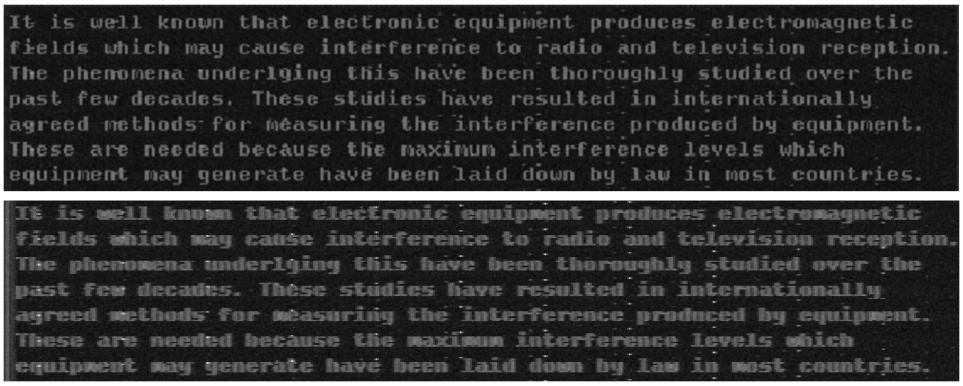

Figure 7: **Top:** Reconstructed with f_{center} 325MHz. **Bottom:** Reconstructed with f_{center} 425MHz

4 Additional comments

- **Increase bandwidth**
  With a bandwidth of 40MHz we are not able to reconstruct all of the information included in the bit pattern transmitted at approximately 250 Mbaud. A potential approach to extend this bandwidth could be to "merge" together several overlapping recordings. To do so, the recording either have to be taken at the exact same time, which would require multiple receivers, or we can leverage the periodic nature of a signal displaying the same image and try to perfectly align frames with the cross correlation method described above. The steps needed would be as follows; Take the Fourier transform of multiple recordings sampled with a center frequency separated by half the sampling bandwidth, make sure they are perfectly aligned using cross-correlation, shift their frequency spectrum by multiplying with different complex phasors so that they cover their respective parts of the frequency spectrum, and finally, "merge" them together using Linkwitz–Riley filters. Such an increasing of bandwidth would mean that eavesdropping of electromagnetic emanations could be successfully done even with very cheap SDR devices that delivers only a couple MHz of bandwidth.

- **Counter measures**
  Research on the topic mentions multiple counter measures against electromagnetic eavesdropping. There are the obvious measures, like covering the entire office in a Faraday cage or patrolling the perimeter in a radius large enough to make such eavesdropping is impossible, but there exists more clever ways. Kuhn and Anderson [3] proposes a simple approach where the use of fonts designed with certain spectral characteristics (pass them through a low-pass filter) can make text impossible to reconstruct for an attacker while still keeping a descent image quality on the target device. Further, for modern flat panel displays, Kuhn argues that the only reliable protection against compromising emanations is to randomize the less-significant bits in the color combinations that are transmitted [8]. This will lead to jamming that cannot easily be handled through periodic averaging.

5 Conclusion

The results presented in this report show how simple DSP techniques can be used for eavesdropping of electromagnetic emanations on modern computer equipment. The work investigates different ways of enhancing the quality of reconstructed images, it covers important elements as sampling accuracy, cross correlation and signal-to-noise ratio. Methods for increasing bandwidths and counter measures against attacks are briefly mentioned. Reasonable next steps, which I did not find time for in this work, would be to implement the bandwidth increasing method and to dive deeper into the *Transition Minimized Differential Signaling* (TMDS) interface and its 10-bit word encoding. A deeper understanding of this interface could potentially lead to some interesting research gaps in this topic.

6 Source code

Below, the crucial parts of the the source code written in this project are presented.

Defining variables

```
# Define target device parameters:
# Display resolution i 640x480, 60 fremes per second
# Toral resolution, 800 x 525 pixels
fps = 60
x = 800
y = 525
# Pixel clock: nominally 25.175 MHz +/- 0.5%
# By trial and error: fp = 25.200099e6
# By automatic tuning:
fp = 25200096.862872317
# Find horizontal frequency by dividing with number of pixels per line
fh = fp/x
# Find vertical frequency by dividing fh with number of lines
fv = fh/y
# Define sampling parameters & centre frequency
# Sampling frequency
fs = 64e6
# Centre frequency
fc = 425e6
```

Downsampling

```
# Match pixelrate by downsamlping from fs to fp:
t = np.arange(0, fs, fs/fp)
linear = interp1d(np.arange(fs), iq)
resampled = linear(t)
#Now Amplitude demodulate the signal:
s = np.abs(resampled);
```

Print image

```
# Print image:
# Construct matrix of image:
im = np.zeros((y,x));
n_sample_frame = x*y
first_frame = 800*262 + 447
frame = 0
n = first_frame + n_sample_frame*frame
print(n)
for i in range(y):
    for j in range(x):
        im[i][j] = s[n]
        n +=1

imsize = [x/50, y/50]
plt.figure(figsize = imsize)
plt.imshow(im[0:480,0:640], cmap='gray')
plt.show()
```

Compare frames

```
# Construct matrix of image:
im = np.zeros((y,x));
n_sample_frame = x*y
first_frame = 800*262 + 447
#first_frame = 800*375 + 570 (For 325MHz)
frame = 0
n = first_frame + n_sample_frame*frame
print(n)
for i in range(y):
    for j in range(x):
        im[i][j] = s[n]
        n +=1

# Generate another image to tune fp:
im2 = np.zeros((y,x));
frame = 10
n = first_frame + n_sample_frame*frame
print(n)
for i in range(y):
    for j in range(x):
        im2[i][j] = s[n]
        n +=1

# Comibine images
im_comb = np.vstack((im[0:300,:],im2[300:y-1,:]))
imsize = [x/50, y/50]
plt.figure(figsize = imsize)
plt.imshow(im_comb[0:480,0:640], cmap='gray')
plt.show()
```

Pixel rate tuning

```
# Assumed pixel rate:
fp = 25200099

for frame_jump in [2, 5, 10, 20, 50, 59]:
    # Assumed number of samples in a frame:
    n_s = int(x*y*fs/fp)
    # Assumed number of samples in a line:
    n_s_line = int(x*fs/fp)
    # Take the 50 first lines of the first frame:
    frame1 = iq[:n_s_line*50]
    # Take a section from the later frame frame:
    frame2 = iq[frame_jump*n_s:frame_jump*n_s+n_s_line*50]

    # Look at the cross correlation:
    corr = np.abs(np.correlate(frame1, frame2, 'full'))

    #Number of samples there actually are between the frames:
    index = np.argmax(corr)
    middle = corr.size//2
    diff = middle - index

    fp = (fs*y)/((n_s*(frame_jump+1)+diff)/(frame_jump+1)/x)

print(fp)
print(diff)
```

Average over frames

```
# Average over frames:
# Generate matrix for 50 frames:;
n_sample_frame = x*y
first_frame = 800*262 + 447
frame = 0
n = first_frame + n_sample_frame*frame
im_frames = {}
for frame in range(50):
    im = np.zeros((y,x));
    n = first_frame + n_sample_frame*frame
    for i in range(y):
        for j in range(x):
            im[i][j] = s[n]
            n +=1
    im_frames[str(frame)] = im

# Averaged image:
im = np.zeros((y,x))
for key in im_frames:
    im += im_frames[key]
im = im/50

imsize = [x/50, y/50]
plt.figure(figsize = imsize)
plt.imshow(im[0:480,0:640], cmap='gray')
plt.show()
```

RGB image

```
# RGB:
# Construct matrix of image:
im = np.zeros((y,x,3));
n_sample_frame = x*y
first_frame = 800*262 + 447
frame = 0
n = first_frame + n_sample_frame*frame
for i in range(y):
    for j in range(x):
        im[i][j][0] = np.real(resampled[n])
        im[i][j][1] = np.imag(resampled[n])
        n +=1

max_R = np.amax(im[:,:,0])
min_R = np.amin(im[:,:,0])
max_G = np.amax(im[:,:,1])
min_G = np.amin(im[:,:,1])

im[:,:,0] = (im[:,:,0]-min_R)/(max_R-min_R)
im[:,:,1] = (im[:,:,1]-min_G)/(max_G-min_G)
imsize = [x/50, y/50]
plt.figure(figsize = imsize)
plt.imshow(im[0:480,0:640], 'plasma')
plt.show()
```

HSV image

```
#HSV
# Construct matrix of image:
im = np.zeros((y,x,3));
n_sample_frame = x*y
first_frame = 800*262 + 447
frame = 10
n = first_frame + n_sample_frame*frame
for i in range(y):
    for j in range(x):
        im[i,j,0] = np.angle(resampled_shifted[n])
        im[i,j,1] = 1
        im[i,j,2] = np.abs(resampled_shifted[n])
        n +=1

max_angle = np.amax(im[:,:,0])
min_angle = np.amin(im[:,:,0])
max_abs = np.amax(im[:,:,2])
min_abs = np.amin(im[:,:,2])

im[:,:,0] = (im[:,:,0]-min_angle)/(max_angle-min_angle)
im[:,:,2] = (im[:,:,2]-min_abs)/(max_abs-min_abs)
im = hsv_to_rgb(im)
plt.figure(figsize = imsize)
plt.imshow(im[0:480,0:640], 'plasma')
plt.show()
```

Shifting spectrum

```
# Determine how much to shift the spectrum
fu = 3.4016466689e6
print(17*fp - fc - fu)
```

```
-7.061846554279327e-05
```

```
# Multiply with complex phasor to shift spectrum:
n = np.arange(0, iq.size, 1)
f = fu-fp
iq_shifted = iq*np.exp(-2j*pi*n*f/fs)

# Print spectogram:
plt.figure()
plt.specgram(iq_shifted[0:int(fs/5)], Fs = fs)
plt.show()
```

Averaged HSV after shift

```
# Avergae HSV image:
# Generate matrix for 50 frames:
n_sample_frame = x*y
first_frame = 800*262 + 447
frame = 0
im_frames = {}
for frame in range(50):
    im = np.zeros((y,x,3));
    n = first_frame + n_sample_frame*frame
    for i in range(y):
        for j in range(x):
            im[i,j,0] = np.angle(resampled_shifted[n])
            im[i,j,1] = 1
            im[i,j,2] = np.abs(resampled_shifted[n])
            n +=1
    im_frames[str(frame)] = im

# Averaged image:
im = np.zeros((y,x,3))
for key in im_frames:
    im += im_frames[key]
im = im/50

max_angle = np.amax(im[:,:,0])
min_angle = np.amin(im[:,:,0])
max_abs = np.amax(im[:,:,2])
min_abs = np.amin(im[:,:,2])

im[:,:,0] = (im[:,:,0]-min_angle)/(max_angle-min_angle)
im[:,:,2] = (im[:,:,2]-min_abs)/(max_abs-min_abs)
im = hsv_to_rgb(im)
imsize = [x/50, y/50]
plt.figure(figsize = imsize)
plt.imshow(im[0:480,0:640], 'gray')
plt.show()
```

References

[1] W. Van Eck, "Electromagnetic radiation from video display units: an eavesdropping risk?" *Computers & Security*, vol. 4, no. 4, pp. 269–286, 1985.

[2] J. Friedman, "Tempest: A signal problem," *NSA Cryptologic Spectrum*, vol. 35, p. 76, 1972.

[3] M. G. Kuhn and R. J. Anderson, "Soft tempest: Hidden data transmission using electromagnetic emanations," in *International Workshop on Information Hiding*. Springer, 1998, pp. 124–142.

[4] M. G. Kuhn, "Compromising emanations: eavesdropping risks of computer displays," Ph.D. dissertation, University of Cambridge, 2002.

[5] F. Elibol, U. Sarac, and I. Erer, "Realistic eavesdropping attacks on computer displays with low-cost and mobile receiver system," in *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*. IEEE, 2012, pp. 1767–1771.

[6] VESA, "Monitor timing specifcations," *Video Electronics Standards Association (VESA)*.

[7] M. Kuhn. (2019) Radio-frequency spectrum usage in west cambridge. [Online]. Available: https://www.cl.cam.ac.uk/~mgk25/local-rf-spectrum.html

[8] M. G. Kuhn, "Electromagnetic eavesdropping risks of flat-panel displays," in *International Workshop on Privacy Enhancing Technologies*. Springer, 2004, pp. 88–107.