


DSP Homework12

Zheng, Yu 

November 21, 2022

Contents

1	Talks about Videos	2
1.1	Summary	2
1.1.1	SDR	2
1.1.2	IPv6	2
1.2	Fuether thought	2
1.2.1	SDRS and Regular Radios	2
1.2.2	IPv6	3
2	My Voice	3
2.1	Analysis and Results	3
2.2	Errors and deficiencies	4
3	Information contained in the phase spectrum and the amplitude spectrum	5
3.1	The amplitude image	5
3.2	The phase image	6
3.3	Conclusion	6
A	Code listings—Find voice’s frequency	6
B	Code listings—Restoration of image	7

Abstract

For this week's homework, I reviewed the two videos we watched in class and wrote a summary of them and some thoughts of my own. Next, I tried to analyze my voice signals to determine the lowest and highest frequencies I could emit. Finally, I analyzed the information carried by the amplitude spectrum and phase spectrum in class.

1 Talks about Videos

1.1 Summary

1.1.1 SDR

The first video is about software radio, which at the beginning, introduces the functions of SDR: listening to radio stations, listening to aircraft radios, and so on. Later, it talks about the hardware components of SDR: SDR receiver, antenna, computer running SDR software, and introduces various types of SDR receiver, such as HackRF One, SDRplay RSP1A, and so on. Different kinds of antennas are also introduced, such as Long Wire, Shortwave Sloper, etc. And several SDR software, it also recommends esdRSharp for beginners, but care needs to be taken to install the correct drivers.

1.1.2 IPv6

The second video is about IP addresses, and at first the human uses IPv4, thinking it contains enough addresses. However, as the number of devices requiring IP addresses increased, the version of IP addresses had to be changed. The value of IP4 changed from 32bits to 128bits of IPv6. The IPv6 address consists of eight parts, each of which contains four hexadecimal characters ($8 \times 4 \times 4 = 128$ bits). The first four parts are the network part (the first 48bits are the Global Prefix, the last 24bits are the Subnet ID), and the last four parts are the host part (Interface ID). A simplified approach is also mentioned: use :: to represent successive zeros, but this can only be used once.

1.2 Further thought

1.2.1 SDRS and Regular Radios

Through the study of other subjects, we already know the basic block diagram of a common radio (see Figure 1). It consists of a mixer, local oscillator, IF amplifier, amplitude detector, according to demand can also add a low frequency amplifier. From beginning to end, the signal in the receiver is analog.

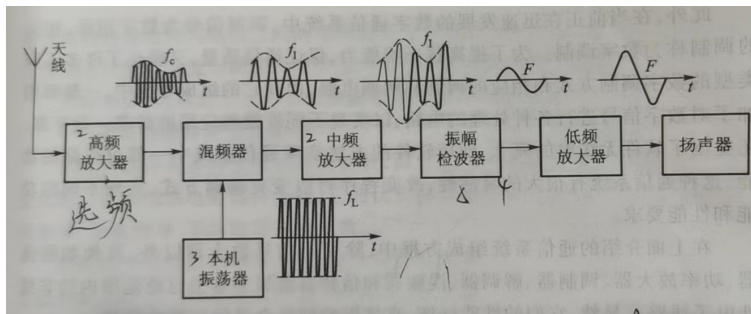


Figure 1: Traditional radio block diagram[1]

If the above analog signal for digital conversion and digital processing, that is, DSP technology, then we get DSP radio.[2] The function circuit needed by DSP radio is integrated into a chip, which is called DSP radio chip. In the chip, it can complete A/D conversion, radio station selection, FM/AM IF modulation signal demodulation, signal decoding and other functions.

However, the aforementioned DSP radio is built on the DSP hardware platform, which needs to be controlled by software, which is referred to in the video as software-defined infinity. Its DSP technology base is reflected in its SDR receiver and antenna. It can be directly through the software to define functions: front-end reception, IF processing and signal baseband processing.[2] The core idea is to use broadband analog/digital converters as close to the antenna as possible to complete the digitalization of the signal as early as possible, so that the function of the radio receiver as much as possible in software defined and controlled.

From the perspective of structure, SDR is actually more complex than traditional radio, but after a high degree of integration, the overall size of SDR is smaller. In addition, all functions of SDR such as searching radio station, switching frequency band, volume control, mute control and reading internal data of the chip are realized by software control. According to the needs of the product, the function can be increased or decreased, as long as the software is modified. This is convenient, and can reduce the number of changes in the hardware circuit, to ensure the service life.

1.2.2 IPv6

Out of curiosity, I looked at my own computer's IP address, as shown in Figure 2. However, I found that in addition to IPv4 and IPv6 addresses, temporary IPv6 addresses and local link IPv6 addresses appeared on the computer. This aroused my interest, so I went to find some information about them.

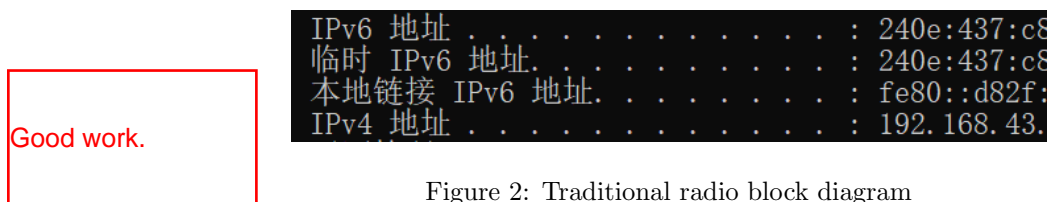


Figure 2: Traditional radio block diagram

1. The temporary IPv6 address is randomly generated by the system so as not to expose the address of the local device. This is used for the security of the computer, because the rules of address generation are uniform, so it is not safe to expose the address of the machine directly. If a PC has both Ipv6 and temporary Ipv6, the PC preferentially uses the temporary Ipv6 address for external communication.[3]
2. For local link IPv6, when IPv6 is enabled on a node, each interface of the node automatically generates an address.[4] The 64-bit prefix of the address is specified by the standard, and the next 64-bit address is constructed in EUI-64 format. Its default prefix is fe80::, which matches the address in the diagram. However, this address can only be used on the local link and cannot be routed between subnets. The reason for using local link addresses is that an interface can be configured with many IPv6 addresses, so many next hops are possible when learning routes. Therefore, the Link Local address uniquely identifies a node. The next hop of the Local Link is the Link Local address of the peer end. During network re-addressing, the Link Local addresses of nodes and routers do not change and can be easily modified without worrying about network unreachable.

2 My Voice

2.1 Analysis and Results

Why not ask around, including me?

For my own voice, ~~I do not quite understand how to produce a higher or lower sound frequency, so I only use the sound frequency of normal speech for FFT change analysis.~~ Since distortion free sampling can only be done if the sampling frequency is greater than twice the maximum frequency, I decided to try to use a larger sampling frequency to ensure a better frequency plot.

As shown below, this is the sound image when the sampling frequency is 10000Hz.

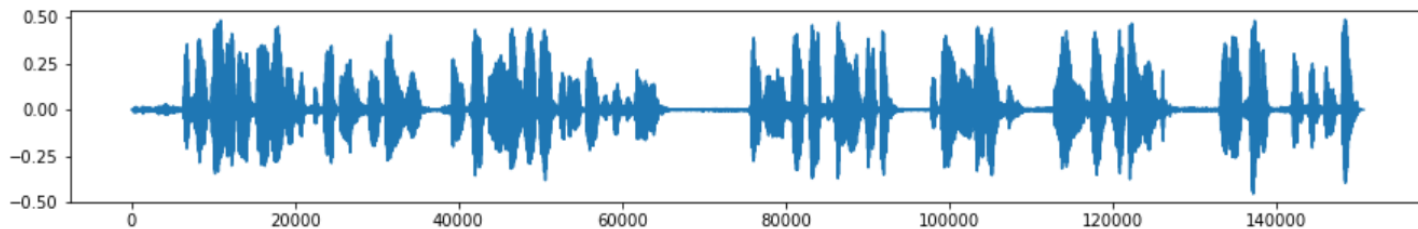


Figure 3: Time domain diagram of the initial sound

After I transform FFT, I get its spectrum. However, due to the existence of noise in nature, it is inevitable that these sound signals will be picked up when recording. So I'm going to do a simple filtering on it in the frequency domain. First, the corresponding amplitude spectrum is calculated. The value of the amplitude spectrum at a certain frequency represents the maximum value of the wave of the corresponding frequency in the time domain. Therefore, I believe that when the amplitude of the wave is less than a range, it can be identified as the noise existing in nature. So I took the following code and zeroed out $\frac{1}{10}$ of the amplitudes less than the maximum for all frequencies. data3 here is the filtered amplitude spectrum. The resulting spectrum is shown in Figure 5.

```
1 wav_path='C:/Users/17105/Desktop/voice.wav'
2 sample_rate=10000
3 voice = librosa.core.load(wav_path, sr=sample_rate)[0]
4 data1 = fft(voice)
5 data2 = abs(data1)
6 avg=max(data2)/10
```

```

7 data1[np.where(abs(data1)<=avg)]=0+0j
8 data3 = abs(data1)

```

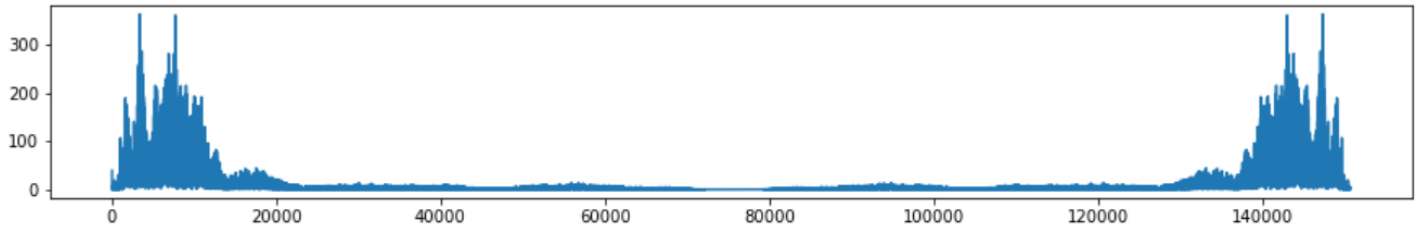


Figure 4: Initial amplitude spectrum

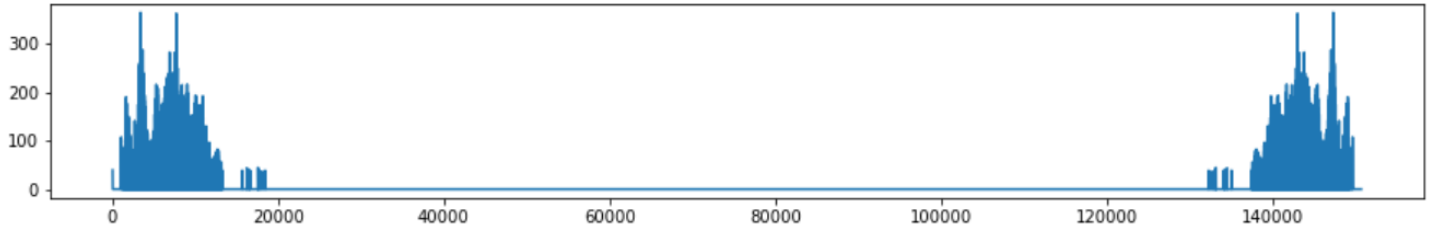


Figure 5: The amplitude spectrum after processing

Then it's a matter of finding the highest frequency and the lowest frequency. After processing, the highest frequency can directly find the corresponding farthest point whose amplitude is not 0. It is important to note that because of the symmetry of the frequency domain graph, the search only needs to be done half the way to the obtained point. The lowest frequency can directly find the first point of magnitude not zero, but first eliminate the DC component at $n=0$. And then they can get the final result from $f = \frac{k}{N}f_s$ when we derive the DFT.

who are they?

```

1 high = 0
2 l = int(len(data3)/2)
3 for i in range(l):
4     if(data3[i] != 0.0):
5         high = i
6 low = 0
7 for i in range(l):
8     if i == 0:
9         continue
10    if(data3[i] != 0.0):
11        low = i
12        break
13
14 p1 = high/len(voice)*sample_rate
15 p2 = low/len(voice)*sample_rate
16 print("Highest frequency:%fHz"%p1)
17 print("Lowest frequency:%fHz"%p2)

```

So, the range of sound that my sound system can produce is about [68.8Hz, 1221.9Hz], as Figure 6.

Highest frequency:1221.868365Hz
 Lowest frequency:68.803079Hz

Figure 6: Result

2.2 Errors and deficiencies

1. First, my simple filtering is based on my general understanding and may not be reliable. In addition, the multiple selection of filtering amplitude will also directly affect the final result.

2. Second, as you can see in Figure 7, there is a part of the frequency that is detached from the "big army". I have reason to think that they are also ambient noise, but I have not thought of a good way to remove them, so the highest frequency results above may be inaccurate. But it's also possible that the frequency component of my own sound, in the middle margin, is actually smaller and that's what's causing them to filter out.

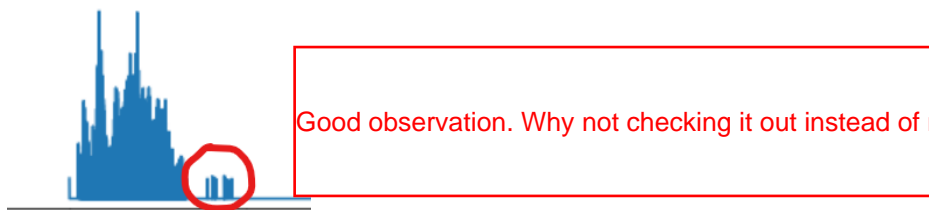


Figure 7: The questionable part

3 Information contained in the phase spectrum and the amplitude spectrum

If we take an image and we apply FFT to it, the information contained in one of its pixels is $a+bj$. Then the amplitude image, the information of each point is $|a + bj| = \sqrt{a^2 + b^2}$. For the phase image, the information at each point is equal to $\arctan(\frac{b}{a})$.

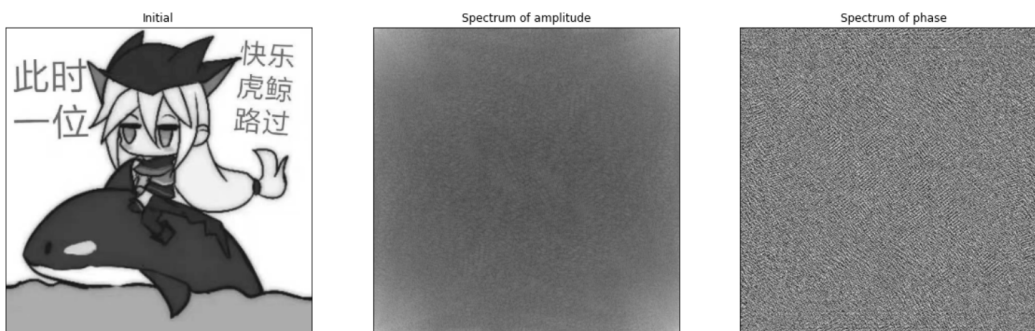


Figure 8: The initial image and its amplitude-frequency diagram and phase-frequency diagram

3.1 The amplitude image

For the amplitude spectrum, it is equivalent to a FFT transformation graph with amplitude $\sqrt{a^2 + b^2}$ but phase equal 0. Using the amplitude image alone is equivalent to superimposing waves of different magnitude but with zero initial phase. Because if the central phase of all the waves is 0, and $e^0 = 1$, it's equivalent to adding all the amplitudes together in center. As we can see from the language description alone, this is not going to recover the image. As in the Figure 9, there is no order when the waveform is superimposed.

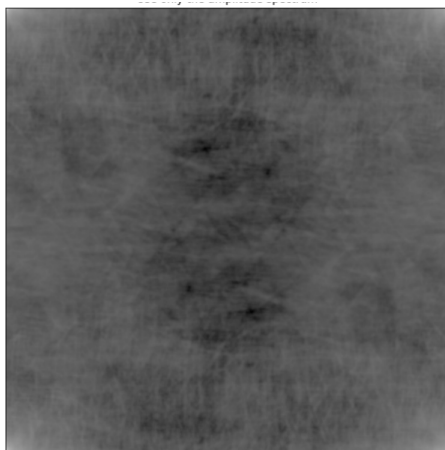
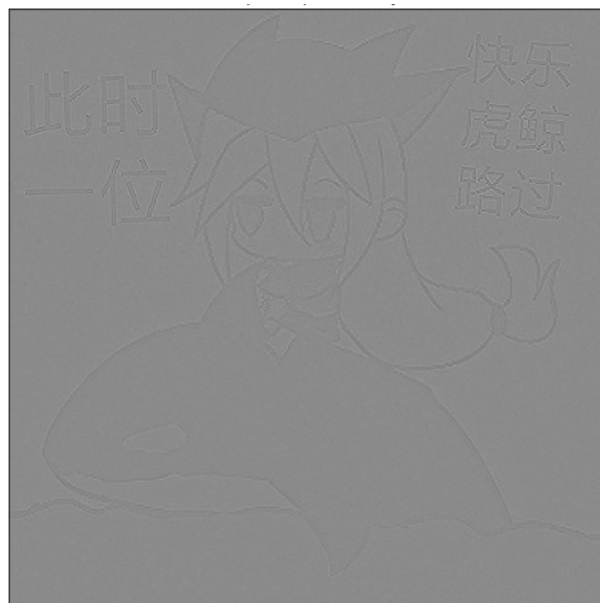


Figure 9: Use only the amplitude spectrum

3.2 The phase image

As for the phase diagram, it is equivalent to an FFT diagram with phase $\arctan(\frac{b}{a})$ but amplitude "1". Using the phase diagram alone to transform is equivalent to making the waves move during the superposition, but without the size information of each wave, both high and low frequency components are the same. But when we do this, we can get a rough outline of the original image. The following picture is the image obtained by IFFT using the amplitude spectrum alone. If we look closely, we can see the entire original image, but the trace is very faint. This gives the impression that the phase spectrum tells us which pixels on the image are colored (not white), but does not have a specific grayscale value.



Good experiment design. But any the

Figure 10: Use phase spectrum only

3.3 Conclusion

Therefore, we can use another way to express the above analysis: the phase spectrum contains the position information of each point in the original image, that is, it implies the contour of the image; The amplitude spectrum contains the color information of each point of the original image, namely the gray value.

References

- [1] :Feng Jun, Xie Hekui.The nonlinear part of the electronic circuit
- [2] :<https://sourl.cn/Wep8rT>
- [3] :<https://www.zhihu.com/question/316884179>
- [4] :<https://sourl.cn/9qSDFR>
- [5] :<https://www.zhihu.com/question/23718291>

Appendix A Code listings—Find voice's frequency

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import librosa
4 from scipy.fftpack import fft,ifft
5 //Importing Files
6 wav_path='C:/Users/17105/Desktop/voice.wav'
7 sample_rate=10000
8 voice = librosa.core.load(wav_path, sr=sample_rate)[0]
9 data1 = fft(voice)
10 data2 = abs(data1)
```

```

11 //Simple filtering is performed
12 avg=max(data2)/10
13 data1[np.where(abs(data1)<=avg)]=0+0j
14 data3 = abs(data1)
15 data4 = ifft(data1)
16 //Find the highest and lowest frequencies
17 high = 0
18 l = int(len(data3)/2)
19 for i in range(l):
20     if(data3[i] != 0.0):
21         high = i
22 low = 0
23 for i in range(l):
24     if i == 0:
25         continue
26     if(data3[i] != 0.0):
27         low = i
28         break
29
30 p1 = high/len(voice)*sample_rate
31 p2 = low/len(voice)*sample_rate
32 print("Highest frequency:%fHz"%p1)
33 print("Lowest frequency:%fHz"%p2)
34 //drawing
35 fig = plt.figure(figsize = (15,10))
36 ax = fig.add_subplot(4,1,1)
37 ax.plot(voice)
38 ax = fig.add_subplot(4,1,2)
39 ax.plot(data2)
40 ax = fig.add_subplot(4,1,3)
41 ax.plot(data3)
42 ax = fig.add_subplot(4,1,4)
43 ax.plot(data4)

```

Appendix B Code listings—Restoration of image

```

1 import numpy as np
2 import matplotlib
3 import matplotlib.pyplot as plt
4 import cv2
5 //*****
6 file = 'C:/Users/17105/Desktop/5.jpg'
7 image = cv2.imread(file, cv2.IMREAD_GRAYSCALE)
8
9 plt.figure(figsize=(20,20))
10
11 ax = plt.subplot(1,3,1, xticks=[], yticks=[])
12 ax.imshow(image, cmap='gray')
13 ax.set_title('Initial')
14
15 pic = np.fft.fft2(image)
16
17 m = np.abs(pic)
18 ax = plt.subplot(1,3,2, xticks=[], yticks=[])
19 ax.imshow(m, cmap='gray', norm = matplotlib.colors.LogNorm())
20 ax.set_title('Spectrum of amplitude')
21
22 a = np.angle(pic)
23 ax = plt.subplot(1,3,3, xticks=[], yticks=[])
24 ax.imshow(a, cmap='gray')
25 ax.set_title('Spectrum of phase')
26
27 plt.show()
28 //*****

```

```
29 //Because I wrote the code on a jupyternotebook, the following paragraph is actually a separate one
30 plt.figure(figsize=(20,20))
31 image1 = np.real(np.fft.ifft2(m*np.exp(1j*0)))
32 ax = plt.subplot(2,1,1, xticks=[], yticks=[])
33 ax.imshow(image1, cmap='gray', norm = matplotlib.colors.LogNorm())
34 ax.set_title('Use only the amplitude spectrum')
35
36 image2 = np.real(np.fft.ifft2(1*np.exp(1j*a)))
37 ax = plt.subplot(2,1,2, xticks=[], yticks=[])
38 ax.imshow(image2, cmap='gray')
39 ax.set_title('Use phase spectrum only')
40
41 plt.show()
```
