COMP9336 Report

Device-to-device Communication over Audio

z3447294
Li Li

In the project, our task is to work out a simple one-directional communication between transmitter and receiver based on audible and inaudible frequency. There are five tasks totally as follows. The experimental devices are based on laptop and Android phone.

- Task1

1 Implementation

In order to detect single tone detection, the requirement is that we only need to consider Goertzel algorithm with parameters sample rate, target frequency and block size. This algorithm can return a power value called magnitude which can help detect the right frequency in the environment data.

Table 1. key points

P1	In terms of Task1, transmitter is easy to be implemented by the helper document given on the Moodle. Also, there are many code fragment on the website to be referenced. Here I would not discussion it.
P2	For task1, receiver should utilize Goertzel to process data to calculate the magnitude. Generally, if the data of target frequency is in the data we obtained from environment, the magnitude of target frequency should be much higher than others. Therefore, it could be considered that we can calculate each unit HZ from 1 to 20000; once the unit is detected with the highest magnitude, we can return it as the result.
P3	Sample rate should be kept above 44100Hz, because our upper is 20000Hz. The following tasks should have the same settings.

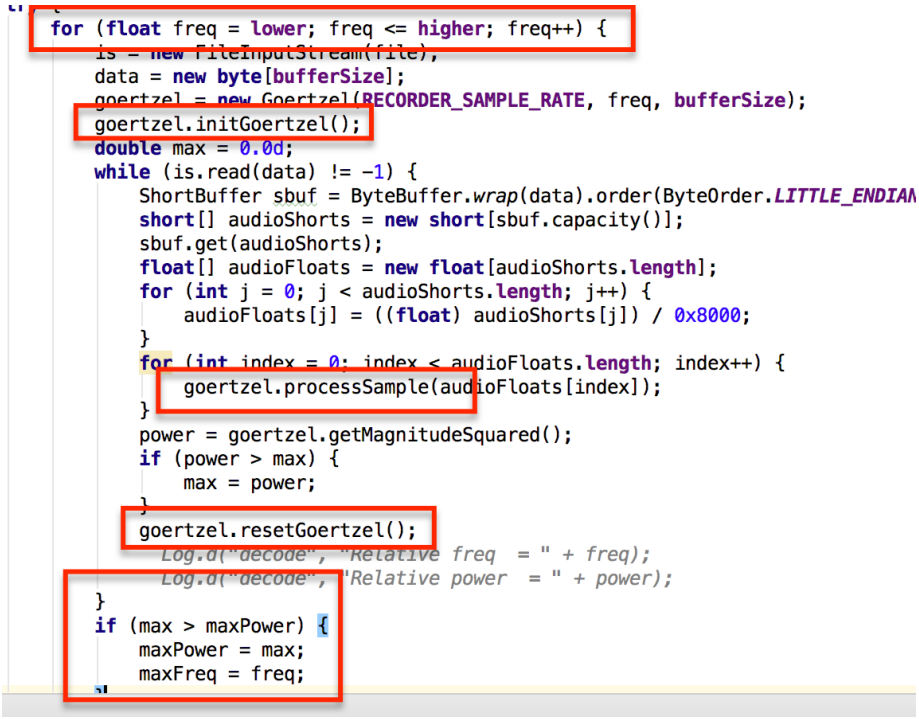

Figure 1. core code in Task1

2 Problems

The above-mentioned P2 could consume a lot of time to scan the whole band to detect the right frequency, but it should be quite accurate except the testing environment is very noisy. However, the reason I choose this method is that is it quite straightforward and accurate on my device. I also try to find threshold of magnitude based on the range of frequency by experiments; its advantage is that it can avoid scanning the whole band like description in P2, and its disadvantage could be that it still cannot ensure high accuracy because it depends on many factors like environments or devices. Therefore, I give up to set up threshold but I know it could help improve detection rate.

- Task2

1 Implementation

Task2 is to require us to hardcode different frequencies to different digits. Based on Task1, it should be rather easy. The tricky part should be detecting less audible frequency. Theoretically, people cannot hear voice over 16KHZ. Therefore, I choose the range between 16KHZ and 20KHZ as less audible frequency.

Table 2. key points

P1	On transmitter end, we should be map frequencies to digits. The choice is as following Figure2. As for generating signal, there is an implementation in helper document on Moodle like s(t) = A*sin(2*pi*f*t).
P2	On receiver end, based on Task1, we have two ways to detect. The first way is to scan the whole band from 500 to 13000Hz and from 16000 to 18400Hz; the second way is to directly scan the self-ruled frequencies one by one as seen in Figure2, which could be much faster but not accurate for high less audible frequency.

```
if ("audible".equals(type)){
    map.put("1", "500");
    map.put("2", "600");
    map.put("3", "700");
    map.put("4", "800");
    map.put("5", "900");
    map.put("6", "1000");
    map.put("7", "1100");
    map.put("8", "1200");
    map.put("9", "1300");
} else {
    map.put("1", "16000");
    map.put("2", "16300");
    map.put("3", "16600");
    map.put("4", "16900");
    map.put("5", "17200");
    map.put("6", "17500");
    map.put("7", "17800");
    map.put("8", "18100");
    map.put("9", "18400");
}
```

Figure 2. hardcode frequency to digit

```
private String task2(){
    Map<String, String> map = task2Map();
    List list = new ArrayList<String>(map.keySet());
    float maxFreq;
    String num;
    if ("audible".equals(type)) {
        maxFreq = detectByGivenFreqList(new File(getTempFilename()), list);
        num = map.get(String.valueOf(maxFreq));
    } else {
        maxFreq = detect(new File(getTempFilename()));
        int confidence = 149;
        if (maxFreq > 16000 - confidence && maxFreq < 16000 + confidence) {
            num =  map.get("16000.0");
        } else if (maxFreq > 16300 - confidence && maxFreq < 16300 + confidence) {
            num = map.get("16300.0");
        } else if (maxFreq > 16600 - confidence && maxFreq < 16600 + confidence) {
            num = map.get("16600.0");
        } else if (maxFreq > 16900 - confidence && maxFreq < 16900 + confidence) {
            num = map.get("16900.0");
        } else if (maxFreq > 17200 - confidence && maxFreq < 17200 + confidence) {
            num = map.get("17200.0");
        } else if (maxFreq > 17500 - confidence && maxFreq < 17500 + confidence) {
            num = map.get("17500.0");
        } else if (maxFreq > 17800 - confidence && maxFreq < 17800 + confidence) {
            num = map.get("17800.0");
        } else if (maxFreq > 18100 - confidence && maxFreq < 18100 + confidence) {
            num = map.get("18100.0");
        } else if (maxFreq > 18400 - confidence && maxFreq < 18400 + confidence) {
            num = map.get("18400.0");
        } else {
            num = "try again";
        }
```

Figure 3. task2 key code

The reason to choose these frequencies is from experiments. At the beginning, I try each frequency with interval 50 Hz, but it is not accurate sometimes. That is why I choose 100 Hz interval. Here as we know, higher frequency is, faster and larger consumption of power is; therefore, the magnitude value should be quite small or closer to each other. That is why I choose bigger interval.

2 Problems

There is still a big problem in Task2. For high frequency or less audible, only if the environment is quiet enough, it could be detected. However, if I try threshold, it is better but there are some error situations. For example, if lower frequency of data from noisy environment has a higher magnitude than that of target less audible frequency, then the code would execute code fragment of low frequency because magnitude of low frequency is beyond the threshold of target frequency.

- Task3

1 Implementation

Task3 is to implement dual tone detection. Actually, it is an extension on Task2. Therefore, we would need to use two frequencies to map a digit.

Table 3. key points

P1	On transmitter end, we should map frequencies to digits based on Table1 standard DTMF in specification. For inaudible frequencies mapped to digits, we have to set up our own frequency based on the detecting accuracy of receiver. The experimental result is as Figure 4. As to generate multiple tone signal, we just need to have a signal with two frequencies by s(t) = s1(t) + s2(t).
P2	On receiver end, based on Task1 and Task2, the best way is to directly scan the self-ruled frequencies one by one as seen in Figure5, which could be much faster but not accurate for high frequency. As long as we get one frequency in lower and higher frequency with highest magnitude respectively, we can target the digit.
P3	As mentioned in P2, this approach could be not accurate enough; however, as two frequencies target one digit, which could be much stable than single mapped one digit, it is much accurate in terms of experimental results. The only thing I have to be concerned is to set up good gap of less audible frequency so that it can analyse the right digit because two close less audible frequencies could have very similar magnitudes and interfere each other.

```
    Map<String, String> map = new HashMap<>();
    if ("audible".equals(type)){
        Log.d("decode", "audible");
        map.put("697.0:1209.0", "1");
        map.put("697.0:1336.0", "2");
        map.put("697.0:1477.0", "3");
        map.put("770.0:1209.0", "4");
        map.put("770.0:1336.0", "5");
        map.put("770.0:1477.0", "6");
        map.put("852.0:1209.0", "7");
        map.put("852.0:1336.0","8");
        map.put("852.0:1477.0","9");
        RECORDER_SAMPLE_RATE = 8000;
    } else {
        // 16000 : 18000
        Log.d("decode", "inaudible");
        map.put("16100.0:18110.0", "1");
        map.put("16100.0:18700.0", "2");
        map.put("16100.0:19325.0", "3");
        map.put("16700.0:18110.0", "4");
        map.put("16700.0:18700.0", "5");
        map.put("16700.0:19325.0", "6");
        map.put("17600.0:18110.0", "7");
        map.put("17600.0:18700.0", "8");
        map.put("17600.0:19325.0", "9");
        RECORDER_SAMPLE_RATE = 41000;
        lower = 16000;
        higher = 19425;
    }
```

Figure 4. frequencies to digits for Task3

```
    Map<String, String> map = task3Map();
    float maxFreqHigh;
    float maxFreqLow;
    String freqPair;
    String num;
    if ("audible".equals(type)) {
        String[] lows = {"697", "770", "852"};
        String[] highs = {"1209", "1336", "1477"};
        List listLows = Arrays.asList(lows);
        List listHighs = Arrays.asList(highs);
        maxFreqLow = detectByGivenFreqList(new File(getTempFilename()), listLows);
        maxFreqHigh = detectByGivenFreqList(new File(getTempFilename()), listHighs);
        freqPair = String.valueOf(maxFreqLow) + ":" + String.valueOf(maxFreqHigh);
        num = map.get(freqPair);
    } else {
        String[] lows = {"16100", "16700", "17600"};
        String[] highs = {"18110", "18700", "19325"};
        List listLows = Arrays.asList(lows);
        List listHighs = Arrays.asList(highs);
        maxFreqLow = detectByGivenFreqList(new File(getTempFilename()), listLows);
        maxFreqHigh = detectByGivenFreqList(new File(getTempFilename()), listHighs);
        freqPair = String.valueOf(maxFreqLow) + ":" + String.valueOf(maxFreqHigh);
        num = map.get(freqPair);
    }
    Log.d("decode", "freq  = " + freqPair);
    Log.d("decode", "num  = " + num);
    return num;
}
```

Figure 5. key code for Task3

2 Problems

The problem is still related to accuracy. Task 3 is only changed to two band to map a digit. Conversely, it makes the less audible situation more accurate. However, in order to get an accurate result, we cannot only depend on this characteristic. Namely, choosing an appropriate band combination for a digit is essential by repetitive experiments.

- Task4

1 Implementation

In this task, I use a series of bits to represent the letters or digits of word/number by ASCII standard. For example, given letter a, ASCII value is 97, I would send two digits 9 and 7 respectively using two frequencies to assign one digit. Actually, I try to implement a preamble that has fixed length and checksum but fails. Therefore, the following could be a simple one.

Table 4. key points

P1	Modulation: As mentioned in above-mentioned paragraph, transmitter would directly send each digit of ASCII decimal value based on standard DTMF, plus 0.
P2	Synchronization: For each packet, transmitter should send a flag to tell receiver that it is time to receive a packet called starting flag. Also there is a flag to assign the end of the packet. During the whole process, transmitter would send each digit or interval flag per second to receiver that is standby. For example, given letter a (ASCII decimal value 97), it would be like this ^9#7$. Caret is a starting flag, hash is an interval flag and dollar sign means ending. See Figure 6. It is a consecutive process to transmit, receive and process.
P3	Threshold: By experiment on my device, the magnitude squared could be around 100 below 5000 Hz. I would use this to filter out some low power points to guarantee meaningful points.
P4	Interval flag #: This flag is vital to ensure to get the exact digit. In the above example, when receiver gets #, it would start to calculate frequency of 0-9 ten digits to see which digit is most frequent at once. And then, receiver can obtain 9. Similarly, receiver get 7 until it meets $ and stops.

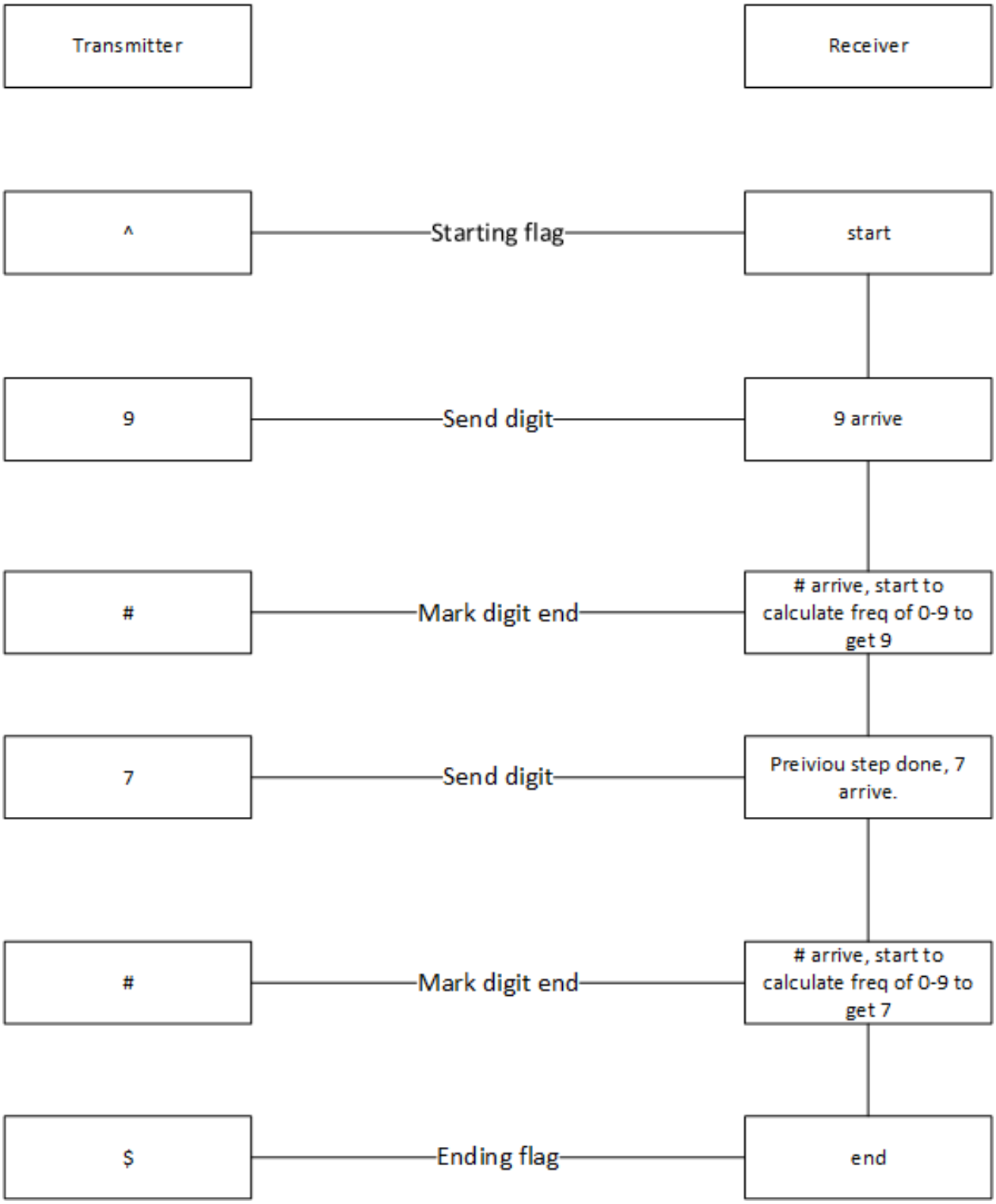

Figure 6. communication flow example

As seen in Figure 6., this is a consecutive process of communication. After obtaining the whole content, it would start to translate to be the real content and show on the device.

2 Problems

So far, this method could cost a lot time to send a few letters. I try to reduce the transmitting time but fails. In addition to this, what I construct right now is not like a real preamble but an effective way to send some short messages without time limitations.

Overall, this project is quite interesting and challenging. As I am new to this field, I made a lot of mistakes on analysis before my receiver become smart. From this experiment, actually, device-to- device communication is quite challenging. There are a bunch of things in the discovery. At the beginning, it cost me a lot of time to understand how Goertzel works and get single tone detection. The next thing is less audible which is hard to be detective due to changes of order of magnitude and its curve characteristic. However, it is still not quite accurate especially in the noisy environment. In the Task4, designing a feasible way to make my device be capable of communicating in one direction is very tricky. Fortunately, I make it more or less. In a word, this project is positive and make me open-minded.