**COMP9336 Assignment Report**

**Title: Device-to-device Communication over Audio**

1. **Task 1 Single Tone Detection**

The purpose of goertzel algorithm is to find the energy of a particular frequency signal from a given sample for evaluation of effectiveness.

This algorithm has several key parameters:

1. The sampling rate R refers to how many samples per second are needed for the data to be analyzed.
2. The target frequency f refers to the value of this frequency that needs to be detected and evaluated.

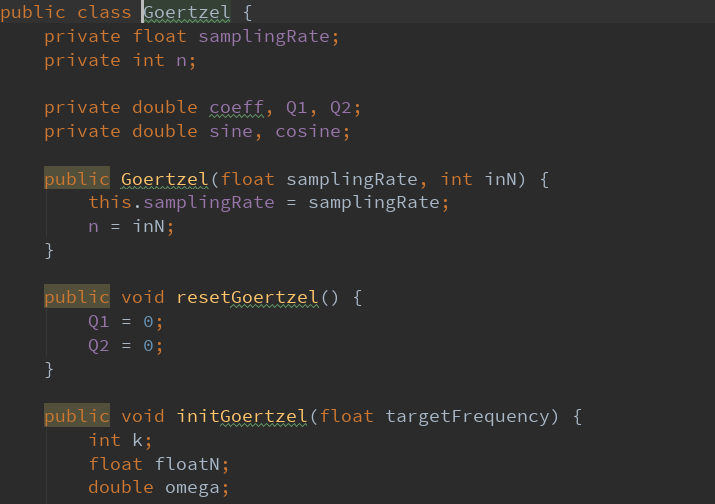
(c) The number of samples in the detection segment is N, that is, every N samples, this algorithm gives an evaluation of the frequency f.

(d) The detection section contains the complete number of cycles of the target frequency K

In this project, The Goertzel algorithm I used reference to <https://codeday.me/bug/20190613/1235954.html> and

<https://github.com/JorenSix/TarsosDSP/blob/master/src/examples/be/tarsos/dsp/example/GoertzelDTMF.java>

, and I modified the initial method, delete some unnecessary initial operations as shows below.



The maximum frequency range that works for my hardware is 17700Hz, from around 300Hz to 18000Hz. It need much time if traversing all frequencies. So here I use n-point search to reduce the duration time of search. In the beginning, I use the goertzel algorithm to calculate the energy value every 300 and find out which one is the largest. Next, I changed the step to 30, and then searched for the interval near the maximum value which I got in last search. Finally, set step to 1, and searching for the interval near the maximum value that is calculated last time.

1. **Task 2 Extension of Single Tone Detection**

In this task, I choose 18 frequencies totally for two versions (audible version and less audible version). For version, I use goertzel algorithm to compare energy values of frequencies. The one owns the largest energy value is what we received. The maximum frequency range that works for my hardware is 17700Hz, from around 300Hz to 18000Hz.

The two version is as below,

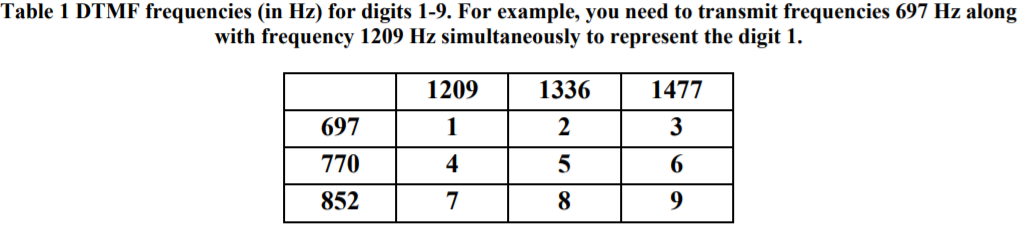
Audible version:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Frequency | 2000 | 2200 | 2400 | 2600 | 2800 | 3000 | 3200 | 3400 | 3600 |
| No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Frequency | 16000 | 16200 | 16400 | 16600 | 16800 | 17000 | 17200 | 17400 | 17600 |
| No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Less audible version:

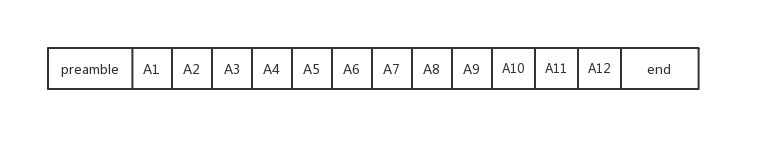
1. **Task 3 Dual Tone Detection**
2. Addition of multiple frequencies in the same signal makes the communication more robust to noise. In my design, I try to use goertzel algorithm to calculate the energy value of these 6 frequencies when the maximum energy value is larger than threshold I set. Then judge which number is sent according to the two of the largest two. It completes the dual tone detection.



1. Due to hardware limitation, I can’t test the inaudible band. However, the principle of dual tone detection in the inaudible frequency band is the same as that in audible band. I still try to design a table as below to represent the DTMP in inaudible band.

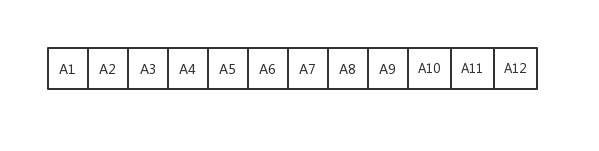
|  |  |  |  |
| --- | --- | --- | --- |
|  | 21209 | 21336 | 21477 |
| 20697 | 1 | 2 | 3 |
| 20770 | 4 | 5 | 6 |
| 20852 | 7 | 8 | 9 |

1. **Task 4 Packetized Data Communication with Audio Tones**



In preamble, I design to use two consecutive different frequencies, 1200Hz and 1000Hz, to start the data bit receiving. This is to avoid misunderstanding that there is packet transmission when single tone existing. In the coding format, A1, A2, A4, A8 are check bits which calculated from data bits (Related calculation methods will be introduced in task5). The rest is data bits. In addition, I have an end bit to finish transmitting. In practice, I used 1200Hz to represent that all transmissions have ended. In order to increase the data transfer rate, I used a base-64. This is because my data bits and check digits are exactly 12bits, so one character only need send twice. This effectively increases the rate of sending according to ( Nyquist theory).

1. **Task 5 Error correction**



In this task, I use hamming code, as figure show above. Among that, A1, A2, A4, A8 is check bits, the rest is data bits. The check rule is :

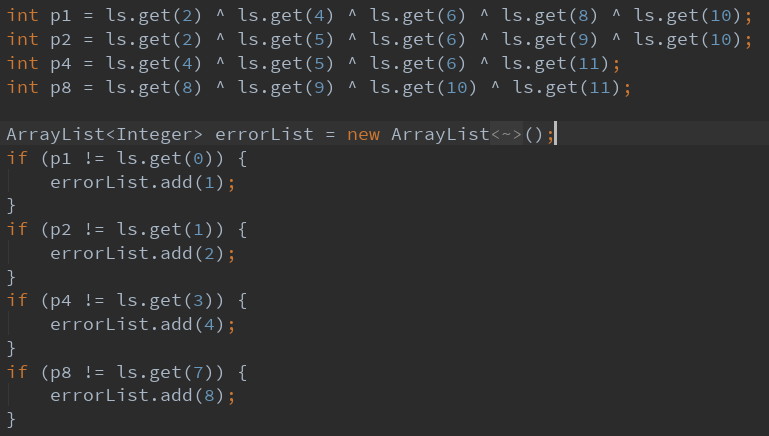
A1 = A3 ⊕ A5 ⊕ A7 ⊕ A9 ⊕ A11 A2 = A3 ⊕ A6 ⊕ A7 ⊕ A10 ⊕ A11

A4 = A5 ⊕ A6 ⊕ A7 ⊕ A12 A8 = A9 ⊕ A10 ⊕ A11 ⊕ A12

On the sender side, we can calculate the correct check bits by check rule and data bits. And then, send these 12 bits to the receiver. On the receive side, calculate the check bits again, and compare with the 4 check bits received.

1. When a 1 bit error occurs, we can find the error and correct it. From the rule above, we can find that each information bit appears in the calculation formula of at least two check digits. So, it is impossible that only one of check bits which calculated on received side is different with the received unless a 1 bit error occur on one of the check bit. Next, we add all indexes of check bits which are different with the received, and we know the index of error bit. Finally, flip the error bit (1->0, 0->1). For example, A3 occurs error. From the formula above, we can see that the A2, A1 will change. Add the indexes (2 + 1 = 3), and we know the A3 is error.
2. Hamming code only can detect some parts of 2 bits error. For example, A3 and A12 have an error during the transfer. So, we can find A1, A2, A4, A8 are different with the received. In this time, 1 + 2+ 4+ 8 larger than 12, we can know the 2 bits error occurs. But we cannot correct this error.

The code is as below:



**Reference**

[1] codeday. 2018. Java Beep. [ONLINE] Available at:

<https://codeday.me/bug/20181112/376662.html>. [Accessed 25 July 2019].

[2] embedded. 2002. The Goertzel Algorithm. [ONLINE] Available at:

<https://www.embedded.com/design/configurable-systems/4024443/The-Goertzel-Algorithm>. [Accessed 30 July 2019].

[3] CSDN. 2018. Hamming Code. [ONLINE] Available at:

<https://blog.csdn.net/Yonggie/article/details/83186280> [Accessed 30 July 2019].

[4] GeeksforGeeks. 2018. Computer Network | Hamming Code. [ONLINE] Available at:

<https://www.geeksforgeeks.org/computer-network-hamming-code/>[Accessed 30 July 2019].

[5] 0110.be. 2016. TarsosDSP API 2.4. [ONLINE] Available at:

<https://0110.be/releases/TarsosDSP/TarsosDSP-latest/TarsosDSP-latest-Documentation/> [Accessed 30 July 2019].

[6] Github. 2014. TarsosDSP. [ONLINE] Available at:

<https://github.com/JorenSix/TarsosDSP> [Accessed 30 July 2019].

[7] CSDN. 2019. AudioRecord. [ONLINE] Available at:

<https://blog.csdn.net/u010126792/article/details/86309592>[Accessed 30 July 2019].

[8] Android.developer AudioRecord [ONLINE] Available at:

<https://developer.android.com/reference/android/media/AudioRecord>[Accessed 25 July 2019].