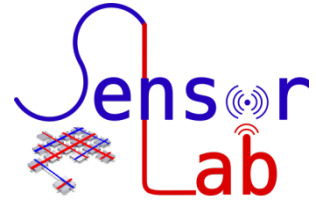




GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN



Georg-August-Universität Göttingen
Faculty of Mathematics and Computer Science
Institute of Computer Science
Telematics Group
Sensorlab

Lab 6 –Sound Recognition

Chencheng Liang – University of Goettingen – 11603960

Introduction

This lab use MTS310CB to detect the voice information, and use regular programming mote with base station program to listen to the message. The listening java commands should be modify to use a higher default baud rate, otherwise it cannot receive any information. And the regular base station listening program cannot get the voice data properly, so we need to use the capture application to receive the audio data.

In assignment 6.1 practical, I modified capture.java to process the audio data, and recognize the noise , then print it on the screen. This is the audio processing in PC. And I modified code in MicReadStreamC.nc, so the audio processing is done at the mote. Unlike in the PC(print information on the screen to should the result), in the mote, according to the noise state, I can change the LEDs and buzzers to be the feedback.

In assignment 6.1 theory, I need to consider the pros and cons for processing audio signal on the PC and on the sensor mote. Also I need to consider if the Fourier transforms is useful for analyzing audio data, and if it is feasible in real time audio data capturing.

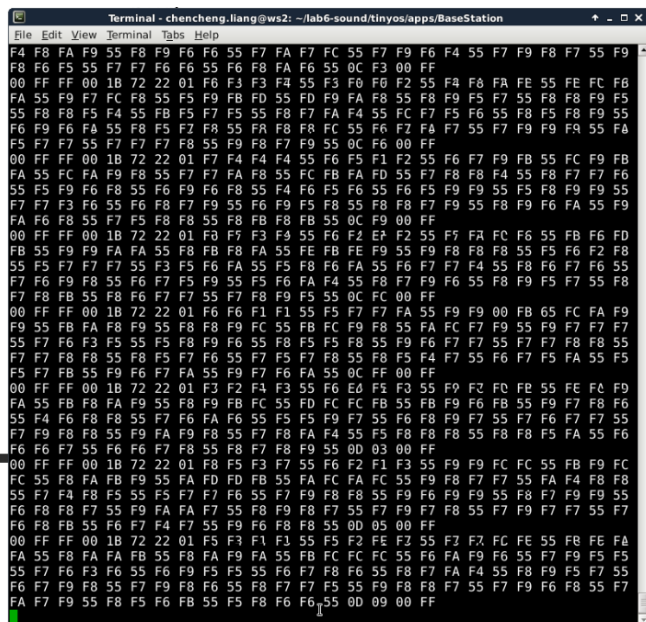
Other audio processing approaches are briefly introduced in that theory part.

Assignment 6.1

Practical:

For the setup, first go to MicReadStreamTest directory, then compile the program with command “make iris”, then install it into a mote. Then go to Basestation directory, install base station program to another mote. Then connect the mote with program MicReadStreamTest to MTS310CB sensor to detect and send audio data.

If I use basic base station listening command “java net.tinyos.tools.Listen -comm serial@/dev/ttyUSB1:iris”, it will not work at all, because audio data have higher baud rate. So I need to use correct baud rate by command “java net.tinyos.tools.Listen -comm serial@/dev/ttyUSB1:230400”. But the result is like this:

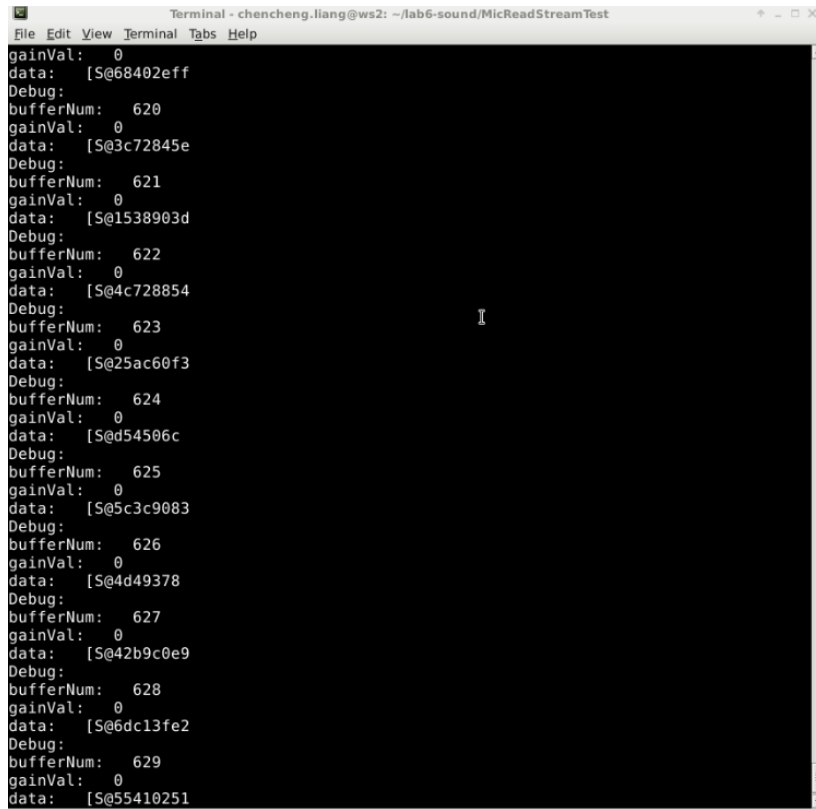


```
Terminal - chengcheng.liang@ws2: ~/lab6-sound/tinyos/apps/BaseStation
F4 F8 FA F9 55 F8 F9 F6 F6 55 F7 FA F7 FC 55 F7 F9 F6 F4 55 F7 F9 F8 F7 55 F9
F8 F6 F5 55 F7 F7 F6 F6 55 F6 F8 FA F6 55 0C F3 00 FF
00 FF 00 1B 72 22 01 F6 F4 F3 F4 55 F3 F0 F0 F2 55 F4 F8 FA F8 55 FE FC F8
FA 55 F9 F7 FC F8 55 F9 F8 FD 55 FD F9 FA F8 55 F8 F9 F5 F7 55 F8 F8 F9 F5
55 F8 F8 F5 F4 55 F8 F5 F7 F5 55 F8 F7 FA F4 55 FC F7 F5 F6 55 F8 F5 F8 F9 55
F6 F9 F6 FA 55 F8 F5 F7 F8 55 F8 F8 F8 FC 55 F6 F2 FA F7 55 F7 F9 F9 F9 55 FA
F5 F7 F7 55 F7 F7 F8 55 F9 F8 F7 F9 55 0C F6 00 FF
00 FF 00 1B 72 22 01 F7 F4 F4 F4 55 F6 F5 F1 F2 55 F6 F7 F9 FB 55 FC F9 FB
FA 55 FC FA F9 F8 55 F7 F7 FA F8 55 FC FB FA FD 55 F7 F8 F8 F4 55 F8 F7 F7 F6
55 F5 F9 F6 F8 55 F6 F9 F6 F8 55 F4 F6 F5 F6 55 F6 F5 F9 F9 55 F5 F8 F9 F9 55
F7 F7 F3 F6 55 F6 F8 F7 F9 55 F6 F9 F5 F8 55 F8 F8 F7 F9 55 F8 F9 F6 FA 55 F9
FA F6 F8 55 F7 F5 F8 F8 55 F8 FB F8 FB 55 0C F9 00 FF
00 FF 00 1B 72 22 01 F0 F7 F3 F4 55 F6 F2 E7 F2 55 F7 FA FC F6 55 FB F6 FD
FB 55 F9 F9 FA FA 55 F8 FB F8 FA 55 FE FB FE F9 55 F9 F8 F8 55 F5 F6 F2 F8
55 F5 F7 F7 F7 55 F3 F5 F6 FA 55 F5 F8 F6 FA 55 F6 F7 F7 F4 55 F8 F6 F7 F6 55
F7 F6 F9 F8 55 F6 F7 F5 F9 55 F5 F6 FA F4 55 F8 F7 F9 F6 55 F8 F9 F5 F7 55 F8
F7 F8 FB 55 F8 F6 F7 F7 55 F7 F8 F9 F5 55 0C FC 00 FF
00 FF 00 1B 72 22 01 F6 F6 F1 F1 55 F5 F7 F7 FA 55 F9 F9 00 FB 65 FC FA F9
F9 55 FB FA F8 F9 55 F8 F8 F9 FC 55 FB FC F9 F8 55 FA FC F7 F9 55 F9 F7 F7 F7
55 F7 F6 F3 F5 55 F5 F8 F9 F6 55 F8 F5 F5 F8 55 F9 F6 F7 F7 55 F7 F7 F8 F8 55
F7 F7 F8 F8 55 F8 F5 F7 F6 55 F7 F5 F7 F8 55 F8 F5 F4 F7 55 F6 F7 F5 FA 55 F5
F5 F7 FB 55 F9 F6 F7 FA 55 F9 F7 F6 FA 55 0C FF 00 FF
00 FF 00 1B 72 22 01 F3 F2 FA F3 55 F6 E0 FE F3 55 F9 F2 FC F6 55 FE FA F9
FA 55 FB F8 FA F9 55 F8 F9 FB FC 55 FD FC FC FB 55 FB F9 F6 FB 55 F9 F7 F8 F6
55 F4 F6 F8 F8 55 F7 F6 FA F6 55 F5 F5 F9 F7 55 F6 F8 F9 F7 55 F7 F6 F7 F7 55
F7 F9 F8 F8 55 F9 FA F9 F8 55 F7 F8 FA F4 55 F5 F8 F8 F8 55 F8 F8 F5 FA 55 F6
F6 F6 F7 55 F6 F6 F7 F8 55 F8 F7 F8 F9 55 0D 03 00 FF
00 FF 00 1B 72 22 01 F8 F5 F3 F7 55 F6 F2 F1 F3 55 F9 F9 FC FC 55 FB F9 FC
FC 55 F8 FA FB F9 55 FA FD FD FB 55 FA FC FA FC 55 F9 F8 F7 F7 55 FA F4 F8 F8
55 F7 F4 F8 F5 55 F5 F7 F6 55 F7 F9 F8 F8 55 F9 F6 F9 F9 55 F8 F7 F9 F9 55
F6 F8 F8 F7 55 F9 FA F7 55 F8 F9 F8 F7 55 F7 F9 F7 F8 55 F7 F9 F7 F7 55 F7
F6 F8 FB 55 F6 F7 F4 F7 55 F9 F6 F8 F8 55 0D 05 00 FF
00 FF 00 1B 72 22 01 F5 F3 F1 F1 55 F5 F2 FE F2 55 F3 F2 FC FE 55 F8 FE FA
FA 55 F8 FA FA FB 55 F8 FA F9 FA 55 FB FC FC FC 55 F6 FA F9 F6 55 F7 F9 F5 F5
55 F7 F6 F3 F6 55 F6 F9 F5 55 F6 F7 F8 F6 55 F8 F7 FA F4 55 F8 F9 F5 F7 55
F6 F7 F9 F8 55 F7 F9 F8 F6 55 F8 F7 F7 F5 55 F9 F8 F8 F7 55 F7 F9 F6 F8 55 F7
FA F7 F9 55 F8 F5 F6 FB 55 F5 F8 F6 F6 55 0D 09 00 FF
```

So I need to use Capture application in MicReadStreamTest to listen to the audio data in base station

with the command "java Capture -comm serial@/dev/ttyUSB1:230400". However in Capture.java, there is no proper print code, so I cannot see any information on the screen. So I add some out.println() inside of Capture.java to print something on the screen.

To finish this task, I need to know in the message which part represents the audio data, so I print all variables received in the base station on the screen:



```
Terminal - chencheng.liang@ws2: ~/lab6-sound/MicReadStreamTest
File Edit View Terminal Tabs Help
gainVal: 0
data: [S@68402eff
Debug:
bufferNum: 620
gainVal: 0
data: [S@3c72845e
Debug:
bufferNum: 621
gainVal: 0
data: [S@1538903d
Debug:
bufferNum: 622
gainVal: 0
data: [S@4c728854
Debug:
bufferNum: 623
gainVal: 0
data: [S@25ac60f3
Debug:
bufferNum: 624
gainVal: 0
data: [S@d54506c
Debug:
bufferNum: 625
gainVal: 0
data: [S@5c3c9083
Debug:
bufferNum: 626
gainVal: 0
data: [S@4d49378
Debug:
bufferNum: 627
gainVal: 0
data: [S@42b9c0e9
Debug:
bufferNum: 628
gainVal: 0
data: [S@6dc13fe2
Debug:
bufferNum: 629
gainVal: 0
data: [S@55410251
```

By this I know that variable bufferNum represents the data count, variable gainVal represent the gain, and variable data[] represents the audio information. Then I found variable data[] can be processed in a function called dataCapture() in Capture.java. And for calling the function dataCapture(), I need to call function initCapture(). So in function messageReceived(), which processes the received data from mote, I call function initCapture() to initialize the processing of received data. And in function dataCapture(), I found a variable ares[] received the audio data, so I print all area[] on the screen:

```
Terminal - chencheng.liang@ws2: ~/lab6-sound/MicReadStreamTest
File Edit View Terminal Tabs Help
Ares: -1600
Ares: 2048
Ares: -384
Ares: -1792
Ares: -1984
Ares: 576
Ares: 640
Ares: -2368
Ares: 640
Ares: -1408
Ares: -1088
Ares: 640
Ares: 1152
Ares: 448
Ares: 256
Ares: -2496
Ares: 2688
Ares: -1984
Ares: -2240
Ares: 2240
Ares: 0
Ares: 384
Ares: -2304
Ares: -3136
Ares: 192
Ares: 64
Ares: -576
Ares: 256
Ares: -2560
Ares: 1280
Ares: -384
Ares: -2560
Ares: 512
Ares: -960
Ares: -3072
Ares: -2112
Ares: -256
Ares: -512
Ares: 512
Ares: -1024
Ares: -64
Ares: -448
```

The data come very fast. However I still can see that when I make some noise, this data changed dramatically. So it verifies that this variable `area[]` denotes the audio data. By processing this variable, the program can recognize the sudden noise.

I declare two global variables(`int count`, `int avg`) to process the audio data, the processing method is very simple. Add every 1000 `ares[]` together, get a average, and shrink down the range, and store it to a variable `avg`. If `avg` more than 10, it is a noise, otherwise, it is not. And receive audio data every 1000 times, print once on the screen, then I have the following result:

```
Terminal - chencheng.liang@ws2: ~/lab6-sound/MicReadStreamTest
File Edit View Terminal Tabs Help
The buffer number is: 10282
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:5
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:15
Noise detected !
The current gain is: 255
The buffer number is: 10319
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:2
1000 Average Ares:7
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:6
1000 Average Ares:0
1000 Average Ares:10
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:11
Noise detected !
The current gain is: 255
The buffer number is: 40391
1000 Average Ares:0
1000 Average Ares:2
1000 Average Ares:14
Noise detected !
The current gain is: 0
The buffer number is: 40410
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:12
Noise detected !
The current gain is: 0
The buffer number is: 40428
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:35
Noise detected !
The current gain is: 0
The buffer number is: 40452
1000 Average Ares:0
1000 Average Ares:0
1000 Average Ares:33
Noise detected !
```

In the MicReadStreamC.nc, I did almost the same process on variable buf[], so according to the audio, the mote can response by LEDs or buzzers.

Theory:

Other approach for audio signal, we can run some sound reorganization algorithm on the PC terminal or on the motes, these algorithm can process the raw sound data in different ways.

I think audio processing should be put on the PC terminal, because audio data stream is RAM-consuming, and usually bigger than 8KB RAM which is the motes have. And processing these audio data for the motes, it needs extra computer power. But for the motes, they use battery as power supply, which is limited. But if we put the process on PC, then the motes still should send a lot of data to the PC, which also consumes a lot of power. And for the PC, if there are thousands of motes sending messages back, it should have large RAM and powerful CPU.

Fourier transforms is useful for analyzing sensed audio data. A algorithm called Fast Fourier

Transform(FFT) by decomposing an N point time domain signal into N time domain signals each composed of a single point. The second step is to calculate the N frequency spectra corresponding to these N time domain signals. Lastly, the N spectra are synthesized into a single frequency spectrum [1]. The complexity of normal Fourier is $O(N^2)$, and for FFT is $O(N \log N)$. I think this speed is considered slow in computer world, because $O(N)$ is normal speed. So I think Fourier transform is not fast at all.

Calculating Fourier coefficients is complex, I think the motes cannot do that with such small RAM and slow CPU speed in real time.

I do not know any other audio processing approach but I read a article [2], which use time-frequency approaches to do it. In that article it mentions that a majority of audio processing techniques address the following 3 application areas: compression, classification, and security. So in the mote we can run the compression algorithm to compress the audio data, then transmit them, which can save a lot of energy, and in PC terminal we decompress the package, and use different classification algorithm to process that audio data.

Lessons Learned & Conclusion

In this lesson, I learn I can use MTS310CB to collect audio signal from the sensor, and listen to the message in the base station. And the noise application I implemented on it has many potential uses in practical, for example, at least we can put the sensor there to find out if there is sound, then record it, so we can figure out what was happening at that place in a sense.

I learned in the base station I need to use the correct baud rate then I can receive that message from motes. And by using Capture application I can analyze the received audio signal.

I learned how to deal with audio data in both theory and practical. In the beginning I do not know which data is audio data, what is the structure of audio data. In practical part of this lab, most of time I used to try to find out where I can receive the audio data, and how can I process the data(they are arrays with high sending frequency). And I just used some easy way to smooth the data, then find the high pitch, and recognize it as the noise. But actually, there are many other algorithms which can process it and event classify it.

I also learned some pros and cons about at where we should implement the audio signal program. The audio signal process can be implemented on both PC and the motes, If it is implemented by PC, then the motes need to send back a lot of raw data, which cost more power. For solving this, we can let the mote run a compression algorithm, then send the compressed data. If the audio signal process is implemented on the motes, it must be a simple program, otherwise the hardware cannot handle that, and it will get more power consuming. And because the limited resources the motes have, it may cannot send back the real time data.

Before this lab I do not know how the radio sensor works, and now I understand, and I know how to catch the audio signal, and do some simple process on it.

References

[1]<http://www.dspguide.com/ch12/2.htm>

[2] <http://asp.eurasipjournals.springeropen.com/articles/10.1155/2010/451695>

