A Java Multithread Audio Acquisition for Audio Surveillance

C. Dadula and E. Dadios
De La Salle University, 2401 Taft Avenue, Malate, Manila 1004 Philippines.
cristina_dadula@dlsu.edu.ph

***Abstract*—This paper presents a real time audio acquisition using a personal computer and multiple USB microphones. When audio stream is available, audio processing techniques can be added to make a real time audio surveillance. The output of this work can be used in audio processing techniques that require audio signals from multiple microphones like source localization and source separation. The designed algorithm implements Java threads and Java sound API (Applications Programming Interface). Two to five microphone threads were run for about 1 second. Each setup was run for 5 trials. The 2-, 3-, and 4-microphone setup provided a promising result for a given hardware specification. The maximum difference for 2-microphone setup was 939 samples or 117 milliseconds duration. For 3-microphone setup, the maximum difference was 1109 samples or 138 milliseconds duration but the two-microphone setup got equal number of samples for about 60% of the trials. For 4-microphone set-up, the maximum number of difference was 1024 samples or about 128 milliseconds but 3 microphones3 microphones got equal number of samples for about 40% of the trials. The 5-microphone setup demonstrated that system resources cannot support that much number of microphones because the one microphone got a very low number of samples compared to other 4 microphones.**

***Index Terms*—Audio Surveillance; Audio Signal Processing; Audio Recording; Java Multithread; Real Time Audio Acquisition; USB Microphones.**

I. INTRODUCTION

Audio surveillance is the process of capturing audio signals of a particular area, and processing the captured audio signals for detecting abnormal events like gunshots, explosions, and screams [1]. The use of audio in surveillance would augment video because usually video is not enough to provide necessary information in case an event occurred. For example, in transport surveillance video is not enough to monitor vehicles activity as well as detect possible abnormal event situations [2]. Abnormal situations may occur outside the view of cameras making it impossible to detect by human operators. In such cases, audio analysis complements video analysis in surveillance systems to detect effectively abnormal events. Nowadays, IP cameras are already equipped with microphones ready to facilitate audio surveillance system. One of the advantages of audio analysis is that it can be employed day and night and it doesn't have to deal with the variations of illumination. But the main issue is, audio analysis or audio processing is very challenging in open environments such as roads.

The audio analysis for audio surveillance may involve audio signal processing techniques such as source localization, sound source separation, and recognition of one or more acoustic emissions. In real time, these processing techniques require audio signals from multiple microphones.

Sound source localization is important in tracking audio targets. The most common approach to sound source localization is based on time delay estimation of audio signals between pairs of microphones. The data along with the known positions of microphones are used to generate hyperbolic curves which are optimized to obtain estimate of the location of the source [3]. Other algorithmic approaches are based on steered response power of beam former and high resolution spectral estimation.

Sound source separation is the process of separating multiple sound sources from the mixture of sources or audio signals from the audio sensor or microphone. The approach is divided into blind source separation (BSS), semi-blind source separation (SBSS), and informed source separation (ISS) [4].

Blind source separation (BSS) is the process of separating sound sources without the knowledge of how the sources are mixed [5]. There are two mixing models of blind source separation: instantaneous and convolutive. Instantaneous mixing model assumes that sound mixture is a linear combination of independent sound sources, $x = As$, where x represents a vector of mixture of sound sources, A is a mixing matrix, and s is vector of independent sound sources. Mathematically if one can get $y = B\,x$, where $B=A^{-1}$ then y is equal to vector s. One approach to BSS is to find the demixing matrix y such that each vector in y that represents an independent source is independent as possible based on some optimization criteria like independent component analysis [6]. In convolutive mixing, the mixture is assumed as the combination of the convolution of sources and finite impulse response filter that represents the transfer function from the sources to the corresponding microphones [7] [8].

In SBSS, there is a partial knowledge about the source signals such as the components of the reference signals [9]. On the other hand, the ISS assumes that some prior information is known enough to determine reasonable target values for the constraint. Instead of optimizing the constraint, the constraint is used toin each component to converge in order to get the corresponding target value [10].

Given the idea that it would be very effective if source localization, source separation and other audio processing techniques used in audio surveillance are done in real time as much as possible. When abnormal event is detected just after a few minutes the event has started the emergency personnel could respond immediately. In cases where the monitored area has only few activities, audio event detection could be used to alert the human operator that an activity is going on or the system may save the time of occurrence of events. The availability of the time of occurrence of events may hasten the review of the stored video from surveillance cameras when necessary.

This paper attempts to design an algorithm for real time audio acquisition using a personal computer and off-the-shelf USB microphones. When audio stream from several microphones is available, audio processing functionalities can be added to make audio processing or surveillance real time as much as possible. The algorithm and its implementation could be used for audio signal analysis that requires audio signals from multiple microphones such as source localization and source separation. The proposed algorithm for audio acquisition utilized USB microphones and implemented Java threads. A thread is a sequential program that has its own thread of control and can execute concurrently with other threads. Recording from each microphone was implemented using threads. The threads accepted the target data line where data was read, and the file name where the data was saved. The proposed algorithm ran threads within 1 second. The number of samples obtained from each microphone were compared. This was done in order to get an idea of how many microphones the hardware setup could handle.

The paper is organized as follows: section 2 gives a brief about Java threads, section 3 presents the design implementation, section 4 discusses the results, and conclusion is in section 5.

II. JAVA THREADS

Seventy-five percent of the real world mature java projects explicitly create threads or employ some concurrent algorithms [11]. A thread is a sequential program that has its own thread of control and can execute concurrently with other threads [12] [13]. Threads compete for time in the same processor or they may execute in parallel on separate processors. A multi-threaded program contains multiple threads or processes. Usually, there are more threads than processors. Threads take turns executing on the processors.

A Java thread can be created by extending Thread class and overrides run method with the necessary statements or implementing the Runnable interface where run method is defined [14]. Figure 1 shows an example of implementing threads.

In this algorithm, a Java thread is implemented by extending the Thread class with the target data line and file objects as the constructor parameters. Audio is read from a target data line object so one should be created for each microphone. The file object holds the filename audio read from a microphone is saved.

III. DESIGN IMPLEMENTATION

A. Hardware materials and setup

The hardware setup consisted of a computer (Macbook Pro 13-in, 2.8 GHz Intel Core i5 processor, 8GB 16MHz DDR3) and 2 to 5 units of omnidirectional USB microphones (UMIK-1). Figure 2 shows a sample actual setup using 4 microphones. The 4 microphones were connected to a USB hub which was connected to one of the USB ports of the computer. During the experiment, other application programs were running like Matlab, Microsoft Excel, and Java Netbeans IDE (Integrated Development Environment). These application programs also consumed the resources of the computer.

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new HelloThread()).start();
    }

}
```

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Figure 1: Example of Java thread implementations [15]

Figure 2: Hardware setup

B. Audio acquisition algorithm

The proposed algorithm basically consisted of two main steps: The first step determined the installed audio inputs or microphones. Not all installed audio devices could deliver audio data to an application. The output of this step was used to get the index of the desired microphones which has the label “Umik_1’’ microphones. The flowchart is shown in Figure 3.

The second step was the audio signal acquisition from Umik_1 microphones. The flowchart of this step is shown in Figure 4. Mixer objects were created for each desired microphone. This was implemented by using the device index found in the first step. The audio format was set to 8000 sample rate, PCM signed encoding, 16 bits per sample, 1 channel, 2 bytes per frame, and little-endian. The target data lines were created for each microphone. The target data line was where the application gets audio data. To read data from the target data line, the target data line objects needed to invoke open then start methods. When the target data line was ready to send data to the program, a thread for each microphone was created. The microphone threads created audio input stream that read data from the target data line and wrote the stream of data to a specified file. Threads ran for about 1000 milliseconds. After reading data from the target

data line, the target data line needed to invoke stop and close methods.

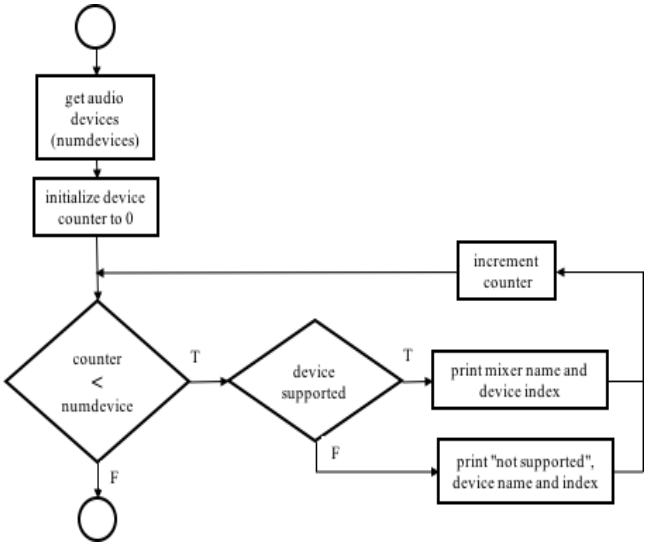

Figure 3: Flowchart of getting installed audio devices

The algorithm implemented the Java sound API (Applications Programming Interface). The audio-input system has three basic elements: (1) the input port, microphone port, or line-in port, (2) the mixer where data is placed, and (3) the target data line where an application can read audio data [16]. The Java sound API was used for controlling audio input and output. It provided functions for installing, accessing, and manipulating system resources such as audio mixers, file readers and writers, and sound format converters.

The recording used the javax.sound.sampled package. The following were some of the classes used from the package: *AudioSystem*, *Line*, *Mixer*, *TargetDataLine*, and *DataLine*. The *AudioSystem* object is the entry point to the sampled-audio system resources. *Line* interface represents a mono or multi-channel audio feed. *DataLine* adds media-related functionalities such as start and stop to its *superinterface*, *Line*. *Mixer* objects are software interface to the physical input device such as microphone. And lastly, *TargetDataLine* is a type of *DataLine* from which audio can be read.

IV. Testing

The algorithm was tested by utilizing 2-, 3-, 4-, and 5-microphone setups. The files from each microphone were examined by tabulating the number of samples obtained from each recording. Five trials were tested for each microphone setup. The number of samples obtained also corresponded to the duration of the recorded audio signal.

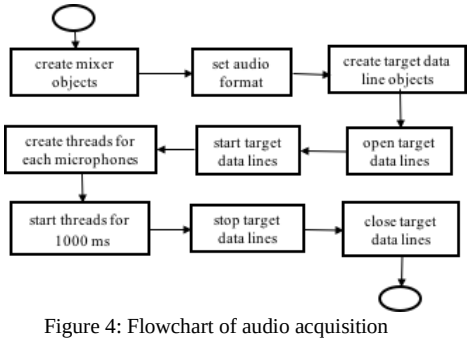

Figure 4: Flowchart of audio acquisition

V. Results and Discussion

The output of determining the audio input devices installed in the computer is shown in Figure 5. By inspection, the desired input devices were "Umik_1 Gain_ 18dB" microphones which had indices of 3, 4, 5, and 6.

```
run:
mixer name: Default Audio Device * index: 0
mixer name: Built-in Microphone * index: 1
not supported;Built-in Output * index: 2
mixer name: Umik_1  Gain_ 18dB   * index: 3
mixer name: Umik_1  Gain_ 18dB   * index: 4
mixer name: Umik_1  Gain_ 18dB   * index: 5
mixer name: Umik_1  Gain_ 18dB   * index: 6
not supported;Port Built-in Microphone * index: 7
not supported;Port Built-in Output * index: 8
not supported;Port Umik_1  Gain_ 18dB   * index: 9
not supported;Port Umik_1  Gain_ 18dB   * index: 10
not supported;Port Umik_1  Gain_ 18dB   * index: 11
not supported;Port Umik_1  Gain_ 18dB   * index: 12
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 5: Sample output of getting audio devices

The first setup used only two microphones. The number of samples obtained from the recorded audio signals in five trials are shown in Table 1 and Figure 6. The average number of samples in five trials of mic 1 and mic 2 were 4366 and 5253.6, respectively. The maximum difference was in trial 1 5288-4349 which was 939 samples. It was about 117 milliseconds duration. The average difference was 887.6 samples equivalent to 887.6/8000 seconds or 111 milliseconds.

Table 1
Number of samples from 2-microphone setup

Trial	Mic 1	Mic 2
1	4349	5288
2	4434	5288
3	4349	5288
4	4349	5202
5	4349	5202
Ave	4366	5253.6

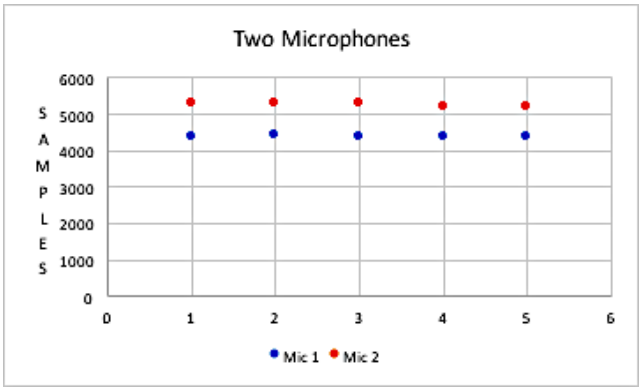

Figure 6: Number of samples from two-microphone setup

The number of samples for 3-microphone setup is shown in Table 2 and Figure 7. The average number of samples for mic 1, mic 2, and mic 3 is 4434.2, 5407, and 5390, respectively. There was a big difference between the number of samples in mic 1 and mic 2, and mic 1 and mic 3. The

maximum difference was in trial 1, 5458-4349 which consisted of 1109 samples or 138 milliseconds duration. However, there was only a small difference between mic 1 and mic 2, the maximum difference was in trial 1, 170 samples (5458-5288) equivalent to 22 milliseconds duration. In addition, two microphones got the same number of samples in 60 % of the trials (3 out of 5).

Table 2
Number of samples from 3-microphone setup

Trial	Mic 1	Mic 2	Mic 3
1	4349	5458	5288
2	4434	5373	5373
3	4434	5373	5373
4	4520	5373	5458
5	4434	5458	5458
Ave	4434.2	5407	5390

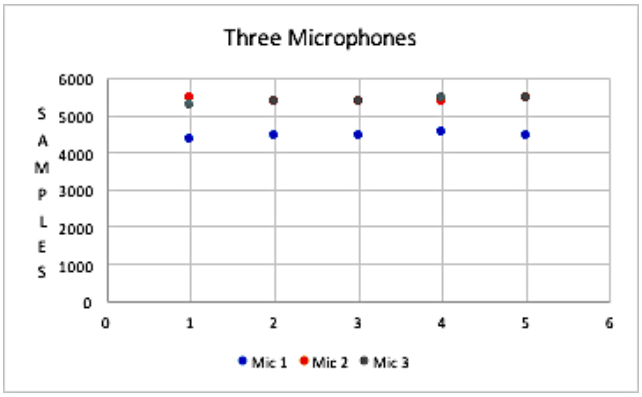

Figure 7: Number of samples from 3-microphone setup

The result of 4-microphone setup shown in Table 3 and Figure 8. It was similar to the 3-microphone setup where the difference of the number of samples from two microphones was very small compared to the difference of each with respect to the other microphone. In the 4-microphone set-up, the difference of the number of samples between 3 microphones was very small compared to the difference of each with respect to the other microphone. The maximum difference between the 3 microphones is 85 samples or 11 milliseconds duration. The maximum difference of the 3 with respect to the other microphone was 1024 samples or 128 milliseconds duration. The sample difference is plotted in Figure 9. Another thing, microphone 3 and 4 had equal number of samples in 2 out of 5 trials, and 3 microphone got the same number of samples in 1 trial (1 out of 5).

Table 3
Number of samples from 4-microphone setup

Trial	Mic 1	Mic 2	Mic 3	Mic 4
1	4602	5458	5373	5373
2	4520	5373	5458	5458
3	4520	5458	5373	5373
4	4349	5373	5373	5373
5	4349	5288	5288	5288
Ave	4468	5390	5373	5373

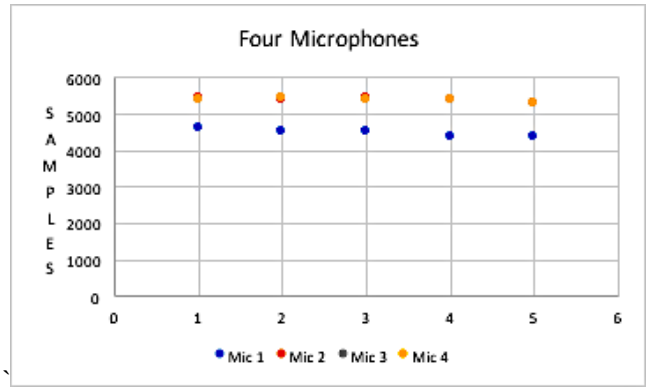

Figure 8: Number of samples from 4-microphone setup

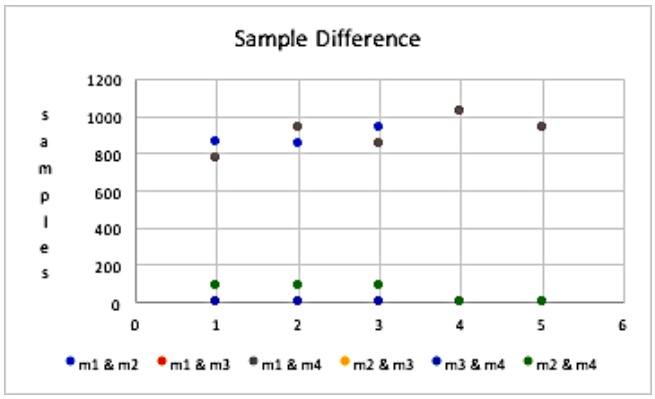

Figure 9: Sample difference between microphones

A further increase in the number of microphones resulted in one microphone with very few samples compared to other microphones as illustrated in Table 4. Microphone 4 got only 86 in trial 3, which is about 2% of the samples of other microphones. This observation implies that additional microphones cannot just be added due the limitation of the computer's resources.

Table 4
Number of samples from 5-microphone setup

Trial	Mic 1	Mic 2	Mic 3	Mic 4	Mic 5
1	4349	5373	5373	2386	3072
2	5288	5373	5288	3239	1963
3	5117	5288	5288	86	5202
4	4349	5288	5288	258	5033
5	5288	5358	5458	4096	1192
Ave	4878	5336	5339	2013	3292

V. CONCLUSION

The proposed algorithm successfully recorded audio signals using java threads The number of samples recorded from each microphone was not always the same**.** The 2-, 3-, and 4-microphone setups provided a promising result for a given hardware specification. The maximum difference for 2-microphone setup was 939 samples or 117 milliseconds duration. For 3-microphone setup, the maximum difference was 1109 samples or 138 milliseconds duration but two microphones got equal number of samples for about 60% of the trial. For 4-microphone set-up, the maximum number of difference was 1024 samples or about 128 milliseconds but 3 microphones got equal number of samples for about 40% of the trial. The 5-microphone setup demonstrated that the system resources cannot support that number of microphones

because one microphone got a very low number of samples compared to the other 4 microphones.

ACKNOWLEDGMENT

The authors would like to acknowledge the financial support given by the Commission on Higher Education(CHED), De La Salle University, Department of Science and Technology - Philippine Council for Industry, Energy, and Emerging Technology Research and Development (DOST-PCIEERD), and Mindanao State University-General Santos City.

REFERENCES

[1] S. Ntalampiras, "Audio Surveillance," WIT Transactions on State of the Art in Science and Engineering , vol. 54, pp. 191-205, 2012.

[2] P. Foggia, N. Petkov, A. Saggese, N. Strisciuglio and M. Vento, "Audio Surveillance of Roads: A System for Detecting Anomalous Sounds," IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 1, pp. 279-288, Jan 2016.

[3] H. Alghassi, S. Tafazoli and P. Lawrence, "The Audio Surveillance Eye," in International Conference of Video and Signal Based Surveillance , 2006.

[4] A. Ferreira and D. Alarcão, "Real-time blind source separation system with applications to distant speech recognition," Applied Acoustics, no. 113, pp. 170-184, 2016.

[5] M. Pedersen, J. Larsen, U. Kjems and L. Parra, "A Survey of Convolutive Blind Source Separation Methods," Multichannel Speech Processing Handbook, pp. 1065-1084, 2007.

[6] C. P. Dadula and D. P. Dadios, "A Genetic Algorithm for Blind Source Separation Using Independent Component Analysis," in HNICEM, Philippines, 2014.

[7] H. Buchner, R. Aichner and W. Kellermann, "A Generalization of Blind Source Separation Algorithms for Convolutive Mixtures Based on Second-Order Statistics," Transactions on Speech and Audio Processing , vol. 13, no. 1, JANUARY 2005.

[8] R. Aichner, H. Buchner and W. Kellermann, "Convolutive Blind Source Separation for Noisy Mixtures," in Speech and Audio Processing in Adverse Environments, E. Hänsler and G. Schmidt, Eds., Springer Berlin Heidelber, pp. 469-524.

[9] F. Nesta, T. Wada and B. Juang, "Batch-Online Semi-Blind Source Separation Applied to Multi-Channel Acoustic Echo Cancellation," Transactions on Audio, Speech and Language Processing , vol. 19, no. 3, pp. 583-599, March 2011.

[10] C. Rohlfing and J. Becker, "Generalized Constraints for NMF with Application to Informed Source Separation," in 24th European Signal Processing Conference , 2016.

[11] G. Pinto, W. Torres, B. Fernandes, F. Castor and R. Barros, "A large-scale study on the usage of Java's concurrent programming constructs," The Journal of Systems and Software , vol. 106, pp. 59-81, 2015.

[12] G. Andrews, Foundations of Multithreaded, Parallel, and Distributed Programming, Addison-Wesley, 2000.

[13] L. Chen, "A Multi-thread Data Flow Solution Applying to Java Extension," in Physics Procedia, 2012.

[14] B. Sanden, Design of Multithreaded Software, Wiley, 2011.

[15] "Thread Objects," 15 March 2017. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/concurrency/threads.html.

[16] "Overview of the Sampled Package," [Online]. Available: https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/. [Accessed March 2017].

[17] "Overview of the Sampled Package," [Online]. Available: https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/. [Accessed March 2017].

[18] C. Dadula and E. Dadios, "Event Detection Using Adaptive Neuro Fuzzy Inference System for a Public Transport Vehicle," in 11th International Conference of the Eastern Asia Society for Transportation Studies, 2016.

[19] Y. Hu and P. Loizou, "Evaluation of Objective Quality Measures for Speech Enhancement," Transactions on Audio, Speech, and Language Processing, vol. 16, no. 1, January 2008.

[20] T. Kinnunen and H. Li, "An Overview of Text-Independent Speaker Recognition: from Features to Supervectors," Speech Communication, vol. 52, pp. 12-40, 2010.

[21] F. Bimbot, J. Bonastre, C. Fredouille, G. Gravier, I. Chagnolleau, S. Meignier, T. Merlin and J. Garcia, "A Tutorial on Text-Independent Speaker Verification," EURASIP Journal on Applied Signal Processing , vol. 4, pp. 430-451, 2004.