Sliding-FFT 를 이용한 특정사운드 감지 알고리즘

지동주 0 조동현 전경구 성미영
인천대학교 멀티미디어시스템공학과
chidj10@gmail.com, uiwangcho@hanmail.net, kjun@incheon.ac.kr, mysung@incheon.ac.kr

Sliding FFT based Algorithm for Detecting Specific Sound

Dongju Chi0 Donghyun Cho Kyungkoo Jun Meeyoung Sung
Dept. of Multimedia-System Engineering, University of Incheon

요 약

적절한 촉각자극과 결합된 효과음은 실감성을 향상시킨다고 알려져 있다. 예를 들어, 영화나 게임의 총소리 효과음에 진동자극이 결합되면 훨씬 몰입감이 증진된다. 영화의 경우, 미디어 파일 내에 진동효과 관련한 정보를 부가하는 연구가 진행되기도 하였으나 대중화되지는 못했다. 게임의 경우, 진동을 유발시키도록 프로그래밍하는 방식을 사용하기도 한다. 하지만 진동을 고려하여 개발하는 게임은 전체 10% 미만일 정도로 일반화되어 있지 않다. 따라서 효과음향을 실시간으로 감지하여 진동을 발생시키는 시스템이 바람직하다. 본 논문에서는 이러한 시스템에서 특정 효과음을 실시간으로 감지하는 알고리즘을 제안한다. 이 알고리즘에서는 감지하고자 하는 효과음의 주파수 분포를 미리 분석해서 저장해 놓는다. 입력되는 효과음에 대해 실시간으로 주파수를 분석하여 저장된 값과의 차이를 비교하여 특정 효과음을 감지하게 된다. 실시간 주파수 분포에는 sliding fast Fourier transform (SFFT)를 사용한다. 이는 특정 효과음의 시작순간을 명확히 알 수 없기 때문이다. 제안 알고리즘을 First Person Shooting (FPS) 게임에 적용하여 성능분석을 하였다. 소음이 없을 경우, 감지율은 80~90%였으나, 소음 정도가 커질수록 감지율이 선형적으로 반비례하였다. 또한 감지에 걸리는 시간은 효과음 발생순간부터 0.45초 이내였다.

1. 서 론

최근 연구결과에 따르면 청각적인 효과와 촉각적인 효과가 동시에 발생할 경우, 인간은 촉각과 청각을 각각 사용해서 상황을 인지하는 것보다 더욱 효과적으로 상황을 인지할 수 있다. 이에 따라, 진동 등 여러 가지 촉각적 효과를 소리와 결합한 휴먼 인터페이스의 연구가 활발히 진행되고 있다[1].

이러한 진동이 가능한 휴먼 인터페이스로는 진동마우스[2], 진동헤드셋[3] 등이 있다. 이들 기기는 소리의 특정 주파수 영역에 따라 진동하는 원리를 가지고 있다. 이들은 여러 가지 소리를 구별할 수 없어 한 가지 패턴의 진동만을 발생시킨다. 이러한 단순성은 오히려 실감성에 역효과를 발생시킬 수 있다.

게임 구현 시 진동을 유발할 수 있도록 할 수 있다. 하지만 이를 고려하는 게임은 10% 미만으로 대중화되어 있지 않다.

따라서 바람직한 시스템은 특정 사운드에 따라 진동을 발생시킬 수 있어야 한다. 즉, 효과음을 실시간으로 분석하여 특정 사운드 발생을 감지할 수 있어야 한다. 예를 들어, FPS 게임에서 사격반동이 큰 저격총을 발사할 때는 강한 진동을 발생시킨다. 반면, 그렇지 않은 소총은 작은 진동을 울려주는 것이 필요하다.

이러한 시스템에서는 효과음을 실시간 분석해서, 특정 효과음을 정확하고 신속하게 감지할 수 있는 방법이 필요하다.

2. 특정 사운드 감지 알고리즘

본 논문에서는 컴퓨터에서 출력되는 음향을 실시간으로 분석하여 특정 효과음 발생을 감지하는 방법을 제안한다.

방법의 핵심은 주파수 분포의 차이를 비교하는 것이다. 이를 위해 Fast Fourier Transform(FFT)를 이용한다. 특정 효과음을 미리 FFT를 이용해 주파수 분석하고 그 결과를 저장해 놓는다. 그리고 입력되는 효과음에 대해 실시간으로 Sliding FFT를 실시하여 저장된 결과와 아래 식을 이용해서 비교한다.

본 논문은 2008년 산업자원부 및 인천 정보 산업 진흥원의 실감형 3D 네트워크 가상환경 기술 개발의 연구비 지원에 의하여 연구되었음.

$$diff = \sum_{x=0}^{n-1} (|sample_x - \text{realtime}_x|) \quad (1)$$

수식 (1)에서 sample_x는 특정 효과음의 특징을 나타내는 주파수 f_x 에서의 magnitude이다. realtime_x 는 실시간 효과음의 주파수 f_x 에서의 magnitude이다. n은 효과음의 특징을 나타내는 주파수들의 개수이다.

수식 (1)에서 실시간 주파수 분석 결과는 저장된 값과의 절대값 차이를 계산한다. 이 결과가 작을수록 특정 효과음일 가능성이 높아진다.

특정 효과음을 미리 분석하여 특징 주파수와 해당 magnitude값을 구하는 방법은 다음과 같다.

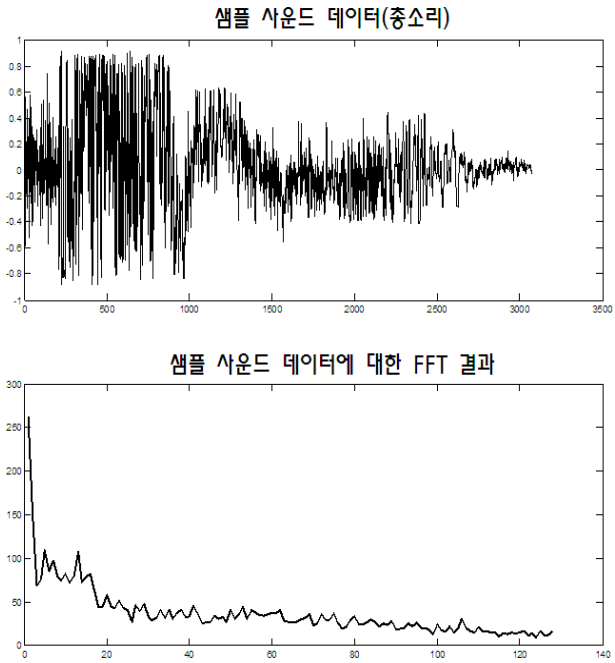


그림 1 샘플 사운드 데이터에 대한 FFT 결과

특정 효과음만을 포함한 사운드 파일을 이용하여 FFT를 수행하면 그림 1의 아래쪽 그래프와 같은 결과를 얻을 수 있다. 여기서는 특정 효과음으로 총소리를 사용했다. 그림 1의 위쪽 그래프는 총소리의 시간 축에서의 waveform을 보여준다. 아래쪽 그래프에서 x 축은 주파수를, y축은 magnitude를 나타낸다. 몇몇 주파수에서 magnitude값이 다른 곳보다 상대적으로 큰 것을 볼 수 있다. 이들 주파수 f_x는 특정 효과음의 특징을 결정한다. 그림 2의 아래쪽 그래프는 새소리의 FFT 결과가 총소리의 결과가 확연히 다름을 보여준다. 따라서 주파수 f_x 을 이용해서 소리를 구분할 수 있다는 것을 알 수 있다.

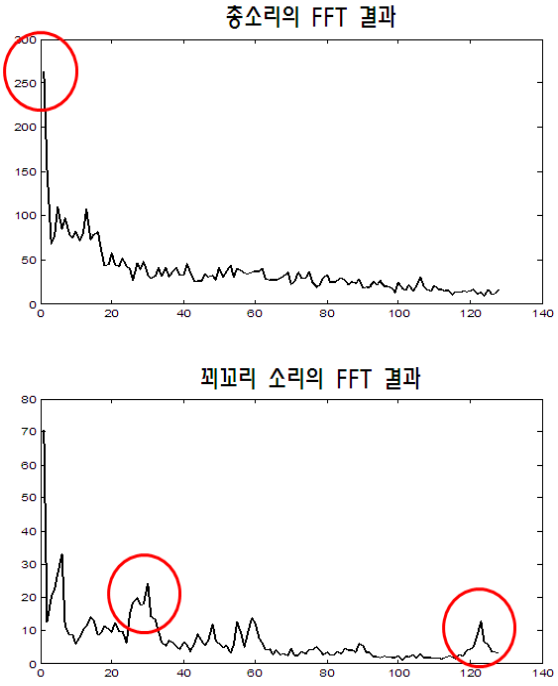


그림 2 FFT 결과의 차이

연속적으로 입력되는 효과음 중에서 특징 효과음을 구별해 내기 위해서는 실시간으로 FFT를 적용해야 한다. 이를 위해 SFFT를 이용한다. 이것은 FFT를 한 번 수행하기 위해 저장한 사운드 샘플의 집합인 윈도우를 조금씩 옮겨가면서 비교하는 것이다. 그림 3과 같이 기존 FFT는 윈도우를 이동할 때 겹치는 부분이 없지만, SFFT는 겹치는 부분이 있다.

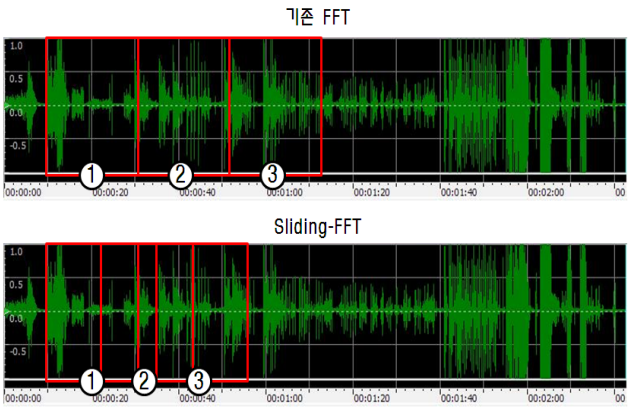


그림 3 윈도우 이동 방식의 차이

실시간 분석에서는 특정 효과음이 언제 발생할지 알 수 없으므로 SFFT를 사용해야 한다. 예를 들어, SFFT는 그림 4와 같이 적용된다. 하나의 윈도우가 256개의 사운드 샘플을 포함한다. 그리고 윈도우 4개의 FFT 결과를 합한 것을 주파수 특성을 찾는데 사용한다.

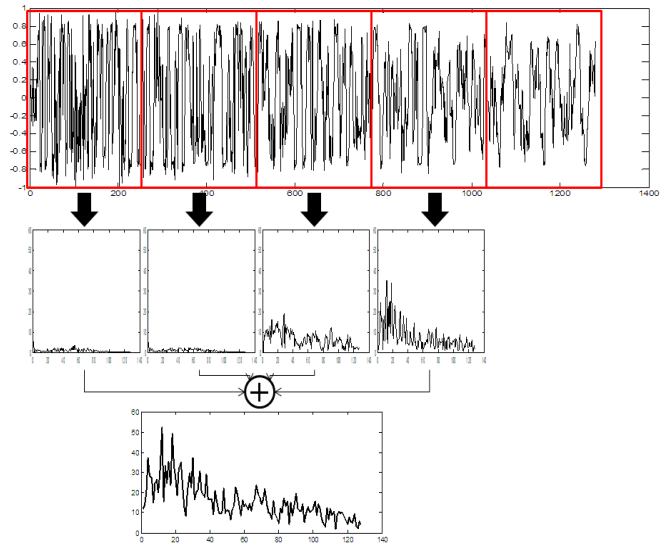


그림 4 윈도우 4개의 FFT 결과를 더함

이 때 다음 1024개의 사운드 샘플을 분석할 때, 제일 처음 윈도우의 샘플 256개를 현재 출력되는 사운드 샘플 256개로 교체한다. 이 경우, FFT는 최근에 출력된 256개에 대해서만 수행하면 된다. 나머지 3개에 대한 FFT 결과는 이미 알고 있기 때문에 FFT를 다시 수행할 필요가 없다. 이로 인해 연산량이 줄어들어 훨씬 빠른 처리가 가능하다.

알고리즘을 상세히 설명하기 위해 몇 가지 요소를 정의할 필요가 있다.

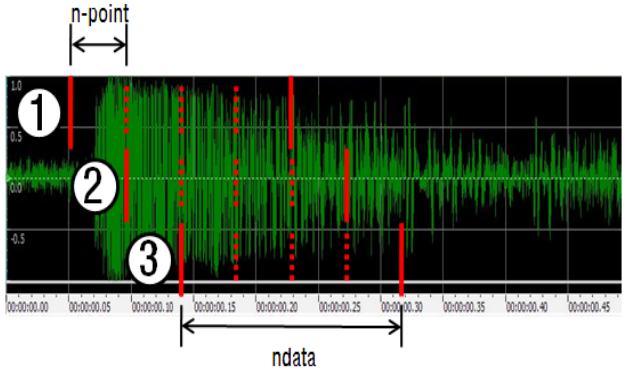


그림 5 n-point와 ndata

- n-point

FFT에서 중요한 두 가지 의미를 가지고 잇다. 첫 번째는 FFT를 한 번 수행하기 위해 모아야 하는 사운드 샘플의 개수이다. 두 번째는 FFT를 수행한 결과로 나오는 magnitude의 개수이다. 그림 5는 n-point를 설명하고 있다.

- ndata

ndata는 효과음 감지 판단을 위해 필요로 하는 사운드 샘플의 개수이다. 그림 5는 ndata 설명하고 있다.

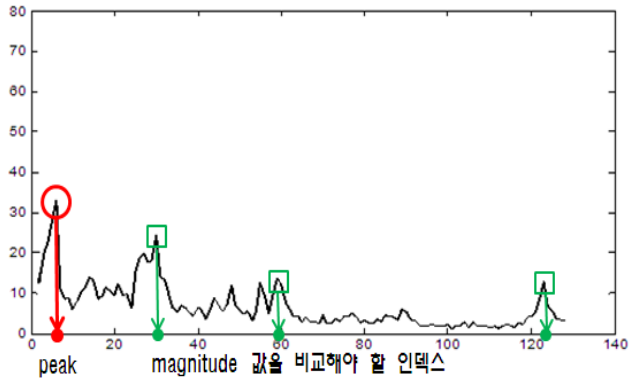


그림 6 peak와 magnitude 값 비교 인덱스

- peak

FFT 결과에서 magnitude가 가장 크게 나오는 부분의 주파수 인덱스이다. 그림 6은 peak을 보여주고 있다.

- 비교 magnitude 인덱스와 값

샘플 사운드 데이터를 분석했을 때 나온 magnitude 중 몇몇 특징적인 값들이 인덱스와 그 값이다. 모든 magnitude를 비교하려면 시간이 오래 걸려서 몇몇 특징적인 값만을 비교에 이용한다. 예를 들어, FFT 결과 3번째, 5번째, 10번째에서 특징적인 값이 나타났다. 실시간 사운드 데이터를 분석하면서 3번째, 5번째, 10번째 값들과의 차이가 적은지에 따라서 감지하려는 소리를 판단한다.

- threshold

샘플 사운드 데이터의 magnitude와 실시간 사운드 데이터의 magnitude 간의 차이의 절대값을 계산한다. 이 값을 모두 더한 값이 threshold보다 작으면 두 주파수 성분 간의 차이가 적다는 것이므로 감지하려는 소리로 판단한다.

3. 알고리즘 구현 시 문제점

본 장에서는 제안 알고리즘을 적용하는 데 있어 고려해야 할 점과 그에 대한 처리방법을 설명한다.

3.1 볼륨에 대한 고려

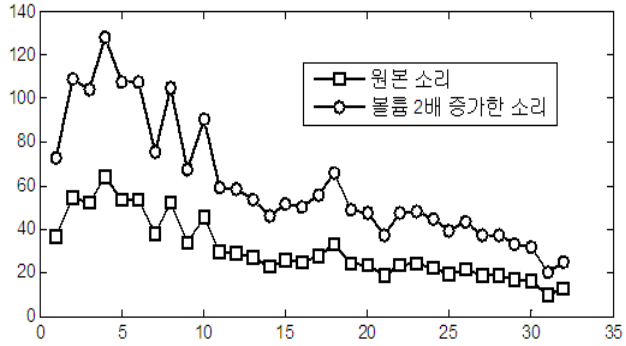


그림 7 볼륨의 차이에 따른 FFT 결과 차이

동일한 소리가 볼륨이 다르게 발생할 때, 그림 7과 같이 다른 FFT 결과가 나온다. 이를 고려하기 위해 볼륨의 크기에 따라 magnitude 값은 변하지만 각 값 사이의 비율은 변하지 않는다는 점을 이용한다. magnitude 간의 상대값을 계산한다. 이러한 상대값을 구하는 normalization이라고 한다.

제안하는 알고리즘에서는 FFT를 통해 나온 magnitude 값들을 가장 큰 값으로 나눠준다. 결과적으로 가장 큰 값은 1이고, 나머지는 1이하가 된다.

3.2 계산량과 시간에 대한 고려

사운드에 대해 빠른 반응성이 요구되는 휴먼 인터페이스에서는 처리속도가 매우 중요하다. 하지만 모든 magnitude를 normalization 및 비교까지 하려면 처리 시간이 늦어진다. 따라서 각 소리마다 특징적으로 나타나는 주파수의 magnitude 값만을 비교한다.

4. 알고리즘 순서도

본 장에서는 실제 알고리즘의 순서도를 설명한다. 이해를 돕기 위해 실제 효과음을 분석하는 과정을 보여준다.

4.1 샘플 사운드 데이터의 주파수 분석

감지하고자 하는 사운드의 주파수 성분을 미리 분석하기 위해 샘플 데이터를 FFT 연산해야 한다. 순서는 그림 8과 같다.

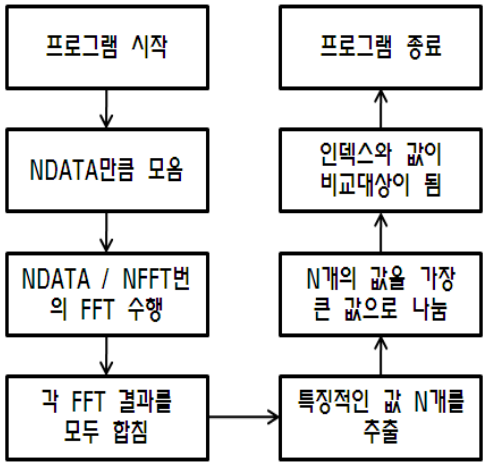


그림 8 주파수 분석 시나리오

샘플 데이터에서 ndata 만큼의 사운드 데이터를 모아 윈도우를 ndata / n-point 개로 나누어 각각 FFT를 수행한다. FFT 결과를 합치고, 가장 특징적인 값 N개를 추출한다. 그리고 N개의 값을 가장 큰 magnitude로 나눈 후에 비교 값과 비교인덱스로 설정한다.

표 1 실험환경

총소리	카빈(소총), 스프링필드(저격총)
Sampling rate	8000 Hz
NFFT	256
NDATA	3072(윈도우 12개)
N	6개(가장 큰 magnitude 제외)

표 1과 같은 실험환경에서 샘플 데이터의 주파수 성분을 분석했다.

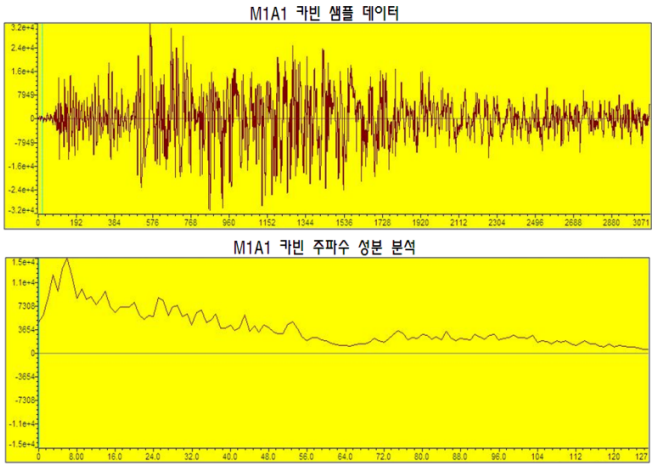


그림 9 M1A1 카빈 분석결과

M1A1 카빈 총소리를 분석한 결과는 그림 9와 같았다.

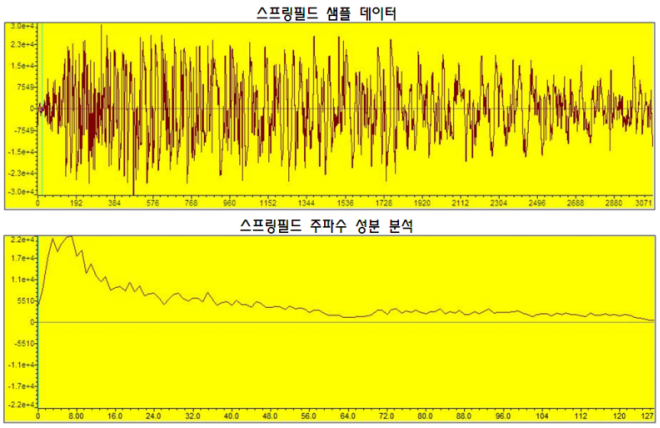


그림 10 스프링필드 분석 결과

스프링필드 총소리를 분석한 결과는 그림 10과 같았다.

표 2 비교 결과

인덱스	카빈	스프링필드
14	0.659072	0.528224
20	0.537152	0.359379
25	0.588533	0.281332
26	0.556529	0.210863
29	0.510331	0.341546
34	0.456349	0.238225

두 총소리에서 특징적으로 차이가 나는 부분을 뽑아내서 표 2와 같은 결과를 얻었다.

4.2 실시간 사운드 데이터 감지

순서도는 그림 11과 같다.

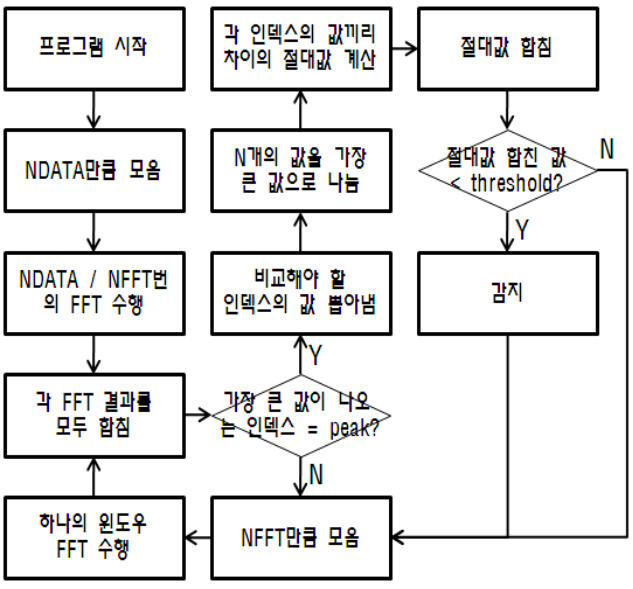


그림 11 실시간 사운드 감지 시나리오

실험환경은 샘플 사운드 데이터를 분석할 때 사용한 표 1과 동일하다. FFT의 결과를 합치는 것은 샘플 데이터를 분석할 때와 같다. 가장 큰 magnitude의 인덱스가 peak와 같으면 비교해야 할 인덱스에 있는 값에 대해 normalization을 수행한다. 각 인덱스 값 사이의 차이를 구해 절대값을 취해 합친 값이 threshold보다 작으면 감지하려는 소리로 판단한다. 초기에 한 번만 모든 윈도우에 대해 FFT 연산한다. 다음부터는 하나의 윈도우만 FFT를 수행해서 이전에 계산된 FFT 값들과 합하면 된다.

5. 성능 평가

본 장에서는 제안한 알고리즘의 성능을 평가한다. 이를 위해 제안한 알고리즘을 DSP 보드상에서 구동한다. 컴퓨터 상에서 FPS 게임을 실행하면서 측정하였다.

5.1 알고리즘 수행 시간 측정

감지하고자 하는 소리의 개수를 1개로 했을 때와 2개로 했을 때의 처리 시간을 비교하기 위해 실험했다. DSP 보드에 있는 타이머 기능을 이용해 루틴이 실행되는 동안 시간을 측정했다.

감지하려는 소리의 개수에 따라 256개로 이뤄진 한 개의 윈도우를 FFT하고 magnitude를 비교하는 데 걸리는 시간을 측정했다. 총소리를 감지하기 위해서 최대 3072개의 사운드 데이터 즉, 12개의 윈도우를 FFT 연산을 통해 비교해야 한다. 그에 따라서 걸리는 시간 계산은 실험1에서 나온 결과에 12를 곱했다.

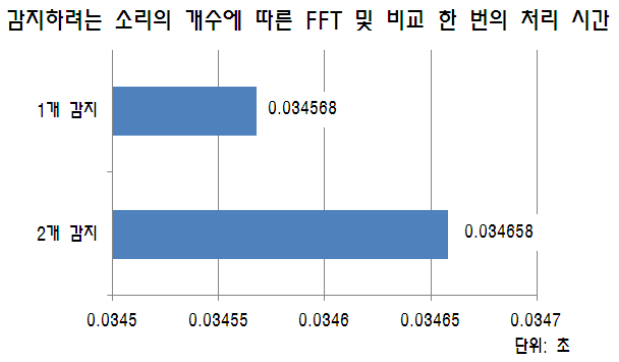


그림 12 소리의 개수에 따른 FFT 및 비교 한 번의 처리 시간

하나의 윈도우를 FFT 연산 및 비교하는 데 걸리는 시간은 그림 12와 같았다. 1개만 감지할 때는 0.034568초가 걸리는데 비해, 2개를 감지할 땐 0.034658초가 걸렸다.

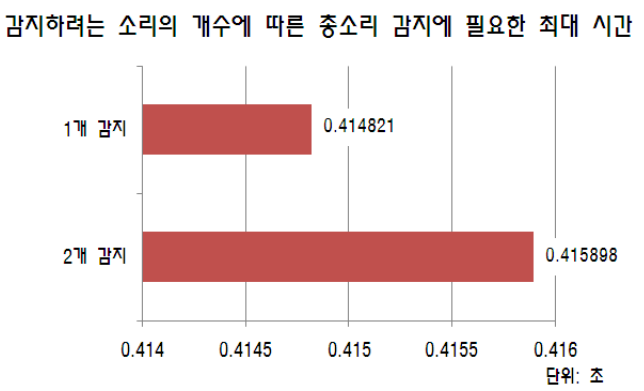


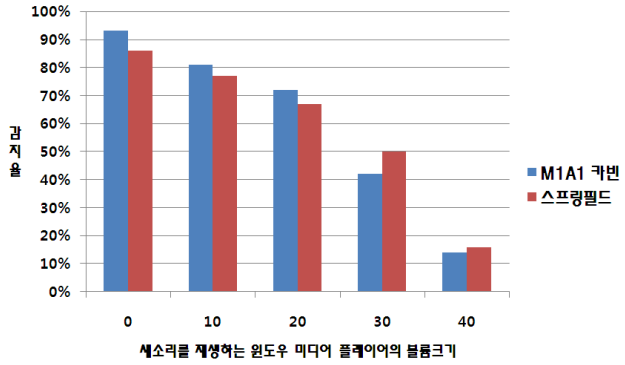
그림 13 소리의 개수에 따른 총소리 감지에 필요한 최대 시간

소리를 감지하기 위해 최대로 걸리는 시간은 12번째 윈도우까지 비교하고 감지했을 경우이다. 이 시간은 1개를 감지할 경우 0.414821초이고, 2개를 감지할 경우 0.415698초로 측정되었다.

8000Hz로 sampling된 사운드 데이터에서 3072개의 데이터는 0.384초에 해당한다. 즉, 사운드 데이터를 모으는 것을 제외한 FFT 및 비교 연산에 걸리는 시간은 1개를 감지할 때 0.030821초, 2개를 감지할 때 0.031898초의 시간이 걸렸다.

5.2 노이즈에 따른 감지율의 차이

게임을 수행하는 도중에 새소리를 지속적으로 재생시키면서 테스트를 수행했다. 게임을 실행하면서 나오는 소리는 일정하게 유지시킨다. 새소리를 재생시킬 윈도우 미디어 플레이어는 0~100까지 볼륨을 설정할 수 있다. 이 볼륨을 0, 10, 20, 30, 40으로 증가시키면서 테스트를 했다.



해당 실험 결과 그림 12와 같은 결과를 얻을 수 있었다. 새소리가 없을 때는 8~90%대의 감지율을 얻었다. 하지만 새소리의 재생 볼륨이 커질수록 감지율이 점점 낮아졌다.

6. 결론

본 논문에서는 실시간으로 들어오는 사운드 데이터 중 특정 사운드를 감지하는 알고리즘을 구현했다. 또한 성능평가를 통해 제안 방법이 실시간 면에서는 활용이 가능함을 알 수 있었다. 하지만 노이즈에 대해서는 취약했다. 이를 위해 향후에는 필터 등의 기능이 추가되어야 한다.

7. 참고문헌

[1] C. R. Fuller and A. H. von Flotow, “Active Control of Sound and Vibration,” in IEEE Control Systems, 1995.
[2] http://www.sound-scape.com/
[3] http://www.xfxforce.com/