

FLINDERS UNIVERSITY

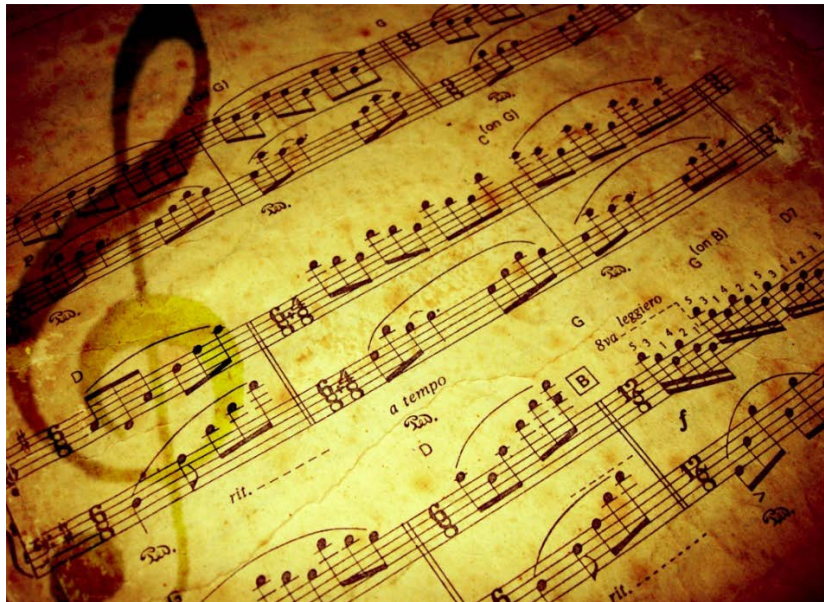
Music from Bio-Signals

Software design for music synthesis

Student Name: Chen Chen

Submission Date: 28/10/2016

Academic Supervisor: Kenneth Pope



(Reference: *Treble on Music Score* by yalorx2)

Submitted to the Computer Science, Engineering, and Mathematics in the
Faculty of Science and Engineering in partial fulfilment of the requirements for
the Master degree of biomedical engineering at Flinders University –
Adelaide Australia

Abstract

Music is a kind of art and entertainment that reflects the real life of human beings. Normally, people could listen to the music recorded from disc or downloaded from the internet. This kind of music is created by musician while its content could not be changed after recording. The project of 'Music from bio-signals' is seeking a new way to synthesis music, which could be composed by the movement of human body in real-time. During this process, physiological signals in body system would be converted to digital signals, and then synthesized to music by software in the computer.

Previous students who involved in the project mainly focus on design of hardware design, therefore the software design of the project is still a blank field. In this project, which is the software design for music synthesis in 'Music from Bio-Signals', MATLAB is the main programming software for music composition. The design starts from the simple sound synthesized from static EMG signals. Since the real-time manipulation of music would be achieved in the final goal, a useful toolbox in MATLAB is introduced to stream out sound from real-time data. Afterwards harmonious music with multiple real-time controls is produced in the computer. In the end, some connecting tests between software and hardware would be carried on.

This project has achieved real-time control of music in MATLAB with four different sounds and two different musical instruments. It has made a great progress for the software design in the project-'Music from Bio-signals'. Further research in the software design could focus on the real-time controllability from the hardware terminals and the reduction of transmission latency between hardware and software.

Declaration

I certify that this work does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Signed 陈晨 Date 28/10/2016

Contents

Abstract	1
Declaration	2
1.0 Introduction	6
2.0 Literature Review	9
2.1 Bio-signals	9
2.1.1 Characteristics	9
2.1.2 Electromyogram (EMG).....	10
2.1.3 Electrocardiogram (ECG).....	10
2.1.4 Electroencephalogram (EEG).....	11
2.2 Music Generation.....	11
2.2.1 Sine Wave	11
2.2.2 Harmonics	12
2.2.3 Octaves	12
2.2.4 ADSR envelope.....	13
2.3 Sound Synthesis Tools.....	13
2.3.1 Max/Msp	13
2.3.2 Pure Data	14
2.3.3 MATLAB	14
2.3.4 Synthesis Toolkit.....	14
2.3.5 Super Collider	14
2.3.6 Open Source Control	15
2.3.7 MIDI.....	15
2.4 Related Researches	15
2.4.1 Music from Bio-signals.....	15
2.4.2 Interactive Instrument Technology in the Musical Performance	16
2.4.3 Digital Musical Instruments Driven by Bio-signals from Musicians.....	16
3.0 Project Design	17
3.1 Sound from static data	17
3.1.1 Test EMG signals	17
3.1.2 Create notes	17

3.2 Music synthesis from sound.....	19
3.2.1 Add harmonics	19
3.2.2 Change ADSR envelope.....	20
3.2.3 Length of notes and pause	21
3.3 Streaming real-time sound	21
3.3.1 Principle of DSP system toolbox.....	21
3.3.2 Introduce ASIO driver.....	23
3.3.3 Test the function of DSP toolbox	23
3.3.4 Minimize the signal discontinuities of sound.....	25
3.4 Music composition from real-time sound	25
3.4.1 Apply the previous model of music composition.....	26
3.4.2 Introduce pentatonic scale	27
3.4.3 Implement multiple controls	27
3.5 Wireless transmission Test	28
3.5.1 Read data from hardware	28
3.5.2 Stream out sound from hardware	29
4.0 Results	30
4.1 Music from static data.....	30
4.1.1 Simple sound with notes	30
4.1.2 Simulating musical instruments	31
4.2 Music from real-time data.....	31
4.2.1 Streaming real-time sound	31
4.2.2 Streaming sound with notes	31
4.2.3 Simulating sound of flute	32
4.2.4 Streaming sound of flute in C major pentatonic.....	32
4.2.5 Combination first and second audios	32
4.2.6 Introduce third sound of piano with a new rhythm	33
4.2.7 Combination third and fourth audios.....	34
4.2.8 Play the four audios with two rhythms.....	34
4.3 Music test through wireless transmission	35
4.3.1 Test the program from one input pin in Bluetooth	35
4.3.2 Test the program from two input pins in Bluetooth	35

4.3.3 Test the program using sine-wave input data through the bread board.....	36
5.0 Discussion	37
5.1 Music from static data.....	37
5.2 Music from real-time data.....	38
5.3 Music test through hardware in real-time	40
6.0 Conclusion.....	42
7.0 Recommendations	43
7.1 Software installation	43
7.2 Maneuverability and conciseness of programming	43
8.0 Acknowledgment.....	44
9.0 References	45
Appendix	48
1. Codes for creating notes from static data (undigested draft)	48
2. Codes for composition from static sound.....	51
3. Codes for the function of ‘ <i>make_note</i> ’ in static composition	52
4. Codes for streaming real-time sound from sine-wave	52
5. Codes for streaming real-time music with multiple controls	53
6. Codes for the function of ‘ <i>frame_note</i> ’ in multiple controls (first flute)	56
7. Codes for the function of ‘ <i>frame_note_4</i> ’ in multiple controls (second flute)	56
8. Codes for the function of ‘ <i>frame_note_second_audio</i> ’ in multiple controls (piano)...	57

1.0 Introduction

Music is an abstract art which emerges long before the start of language that is used to express the thoughts and feelings in human history. With development of human labor, the original music produced by striking stone or wood is to celebrate the harvest and show the pleasure forming a prototype of music. Generally speaking, music is an art performance composed by sound with melody, rhythm, voice, harmony and a large variety of musical instruments. In recent years, researchers are studying the therapeutic effect of music inspired by lullaby in order to expand a new field of physical therapy and children rehabilitation.

The project of Music from Bio-signals is seeking a way using bio-signals in human body to control the streaming music in computer in real-time. This form of music is different from the traditional type which is produced by musical instruments. All the specific parameters, such as the volume, pitch, length of notes and interval, as well as rhythm and melody are created by different kinds of physiological parameters in human body. The very slight variation of bio-signals in different regions of body would be detected and then converted into these parameters in a piece of music. It means that people could compose music when they move their limbs, walk on the road or do any other activities in any circumstances with a device of 'Music from Bio-signals'. It is would be a significant design of this device which could not only be applied on the research of children rehabilitation and physical therapy, but also could be used on the art performance which could combine the dance and music into a new form of art enriching the entertainment life of the public.

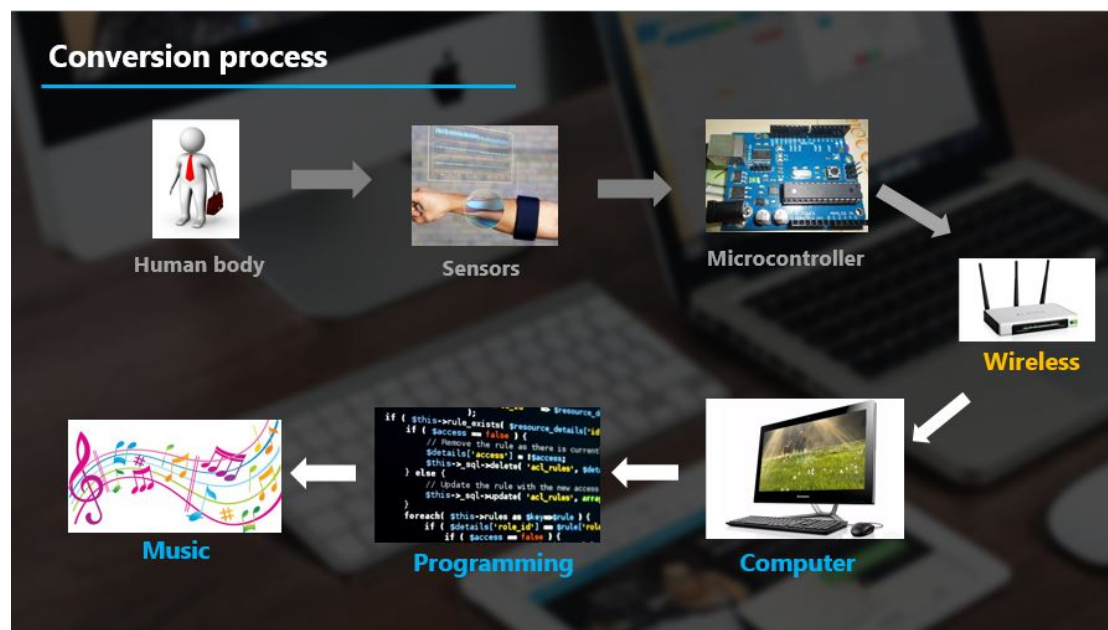


Figure 1. The Conversion process of 'Music from Bio-signals'

As bio-signals has been widely used in the field of medical diagnose and analysis in recent decades, the acquisition of these signals tends to simplicity. Regarding to this project, a semi-finished hardware has been researched and developed for acquiring the data from human body and converting the data into digital signals by previous students. However, the software component of music synthesis for bio-signals is still a blank field in this project. Therefore, making a reliable musical program for this project is a critical process to achieve the function of desired device. Figure 1 shows the overall process of ‘Music from Bio-signals’ while the highlight blue components illustrate the scope of this project.

Before the start of this project, the segmented goals of project design in this year are listed as follows:

- Investigate the characteristics of bio-signals in a digital expression such as the frequency, amplitude, the waveform etc.
- Understand the basic knowledge of music theory including the rhythm, pitch, volume and notes as well as musical instruments.
- Compare different software for music synthesis, select reliable software for programming
- Write computer program to control every variable in a piece of music using the static digital data of bio-signals
- Build a model for streaming continuous sound in the programming software
- Apply the previous static program on the dynamic model for music synthesis
- Revise the program to stream a harmonious music
- Add more input channels into this program to achieve multiple controls in hardware terminals

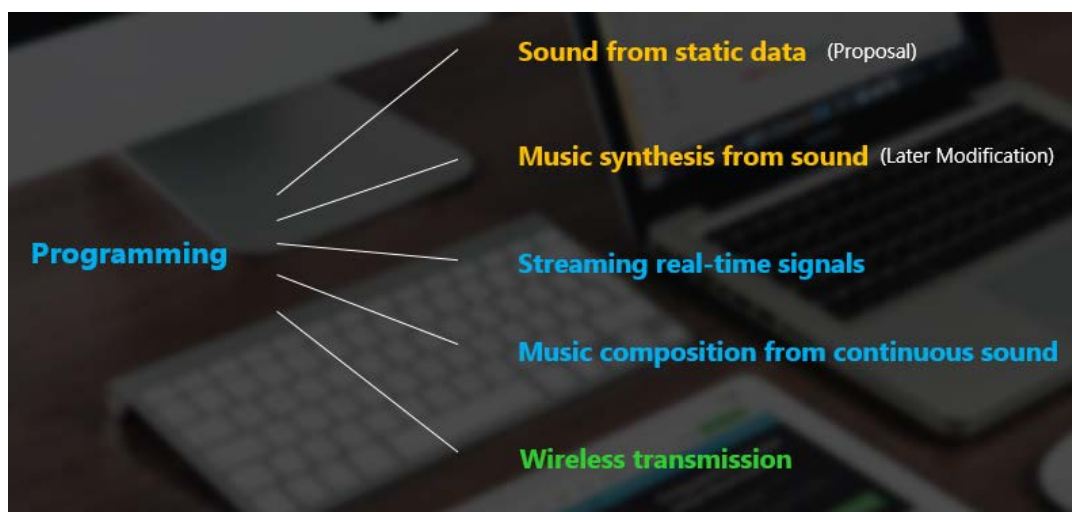


Figure 2. The expected programming tasks in the project

The additional tasks are planned if there are enough time reminded:

- Test the input digital data which is introduced by the detecting and transmitting

hardware developed in previous researches

- Improve the compatibility of the software in this project with the designed hardware
- Improve the stability and reduce the latency through the signal transmission process
- Complete the real-time control of music from bio-signals

2.0 Literature Review

Researchers have found that music is an effective therapeutic tool in the field of physical therapy (Novotney, 2013). In the trial at the University of Alberta, the pain and distress of forty-two children patients at three to eleven years old are reduced by the relaxing music (Hartling et al., 2013). As the widely application of bio-signals in the domain of disease analysis and diagnosis, acquisition of accurate bio-signals is easier and feasible than that in many years ago. These signals could be used to generate music for physical therapy as well as a new format of art performance using musical synthesis software. Based on the current situation, there is no university or individual providing the complete design of 'Music from Bio-signals'. Therefore, some related concepts and researches should be investigated firstly before the explanation of this project.

2.1 Bio-signals

There are numerous physiological processes in the system of human body. Each process contains a lot of useful information which reflect the health condition of physiological system. For example, the voltage recorded on the surface of scalp, the blood pressure measured by electronic sphygmomanometer, the body temperature calculated by thermometer from patient (Brown & Gupta, 2008). Such information associated with the nature of human body are biomedical signals. Biomedical signals are divided into three types, biomechanical signals, bioelectrical signals and biochemical signals. Specifically, the bioelectrical signals detected directly by electrodes from human body are widely used to monitor and analyze the health condition in clinical medicine field. The commonly used biomedical signals include electromyogram (EMG) that indicates the electrical activity of muscular cells, electrocardiogram (ECG) that represents the heart health, and electroencephalogram (EEG) that shows the electrical activity of brain (Muthuswamy, 2004).

2.1.1 Characteristics

Signals could be classified into continuous and discrete types. For example, the algebraic expression $y = x(t)$, $t \in \mathbb{R}$ contains variables, such as time or space while the other expression $y = x(n)$, $n=0, 1, 2, 3, \dots$ includes finite number of points. During the process of digital signal analysis, researches often deal with the discrete signals. Signals could also be divided into deterministic and random types (Kabal, 2004). Deterministic signals can be explained by mathematical functions while random signals usually have uncertain components which can only be analyzed by statistic techniques. For example, both the ECG signal and blood pressure have deterministic sections while EMG signal could be analyzed by standard deviation or root mean square which is one of the methods of time-domain measurement of bio-signals. For example, the root mean square shows mechanical feature of muscular nerves, when the EMG signal is measured. Meanwhile, the smoothness of the signals could be explained by average rectified value through measurement. Besides, the frequency

domain characteristics of digital signals could be illustrated by Fourier transform techniques (Norali, Som & Kangar-arau, 2009).

2.1.2 Electromyogram (EMG)

In 1849, a German physiologist Dubois Reymond found that the electrical activity of muscle contraction could be recorded. In 1890 the term electromyogram is then introduced by Marey when he recorded the first contraction of skeleton muscle. (Cram & Kasman, 1998). In twentieth century, the acquisition method of myoelectric signal is gradually improved and applied for clinical diagnosis such as myopathic and neuropathic disorders. In principle, the electrical signals between muscles and central or peripheral system control the movement of extremities. These signals are the summation of action potential produced by motor units in the region detected by electrodes. Therefore, any abnormal changes in the spinal cord, neuromuscular conjunction or motor-neurons could reflect the occurrence of diseases in the electromyogram. Generally the measurement potential ranges from 20mV to 50mV while the bandwidth of the amplified myoelectric signal is greater than the range from 0 to 4k Hz (Brown & Gupta, 2008).

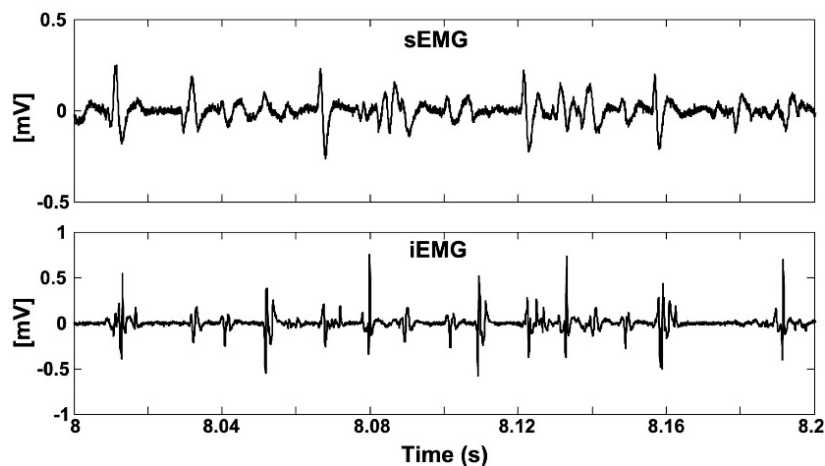


Figure 3. Two channels of EMG signal from surface electrode and needle sensor (De Luca et al., 2006)

2.1.3 Electrocardiogram (ECG)

Electrocardiogram is measured by skin electrodes as a voltage from electrical activity of myocardium in human body. It contains the features of action potential from different parts of heart. In clinical, there are 12 standard leads monitoring the condition of heart for further diagnosis according to the waveform of electrocardiogram. In the ECG diagram, the wave is split up as P wave, QRS wave, T wave and U wave (Olvera, 2006). The frequency and amplitude of each section explain the specific state of heart health. Usually the measured potential from

myocardium is about 90mV, but it would be reduced to 1-2mV when it is reached to the skin.

2.1.4 Electroencephalogram (EEG)

Electroencephalogram is often measured by 21 electrodes to monitor the post-synaptic potentials in cerebral cortex clinically. The frequency range of EEG waveform could be classified as alpha, beta, theta and delta. These frequency ranges could be used to indicate the normal condition or pathological state of neurological patient. During the anesthesia process in intensive care unit, the effect of drug dose on brain activity performs clearly on EEG waveform with changes of frequency and amplitude. The amplitude range is normally 1 to 100mV while the frequency range could be up to 40Hz (Teplan, 2002).

2.2 Music Generation

When a guitarist pluck the strings by hand, the airs near the guitar vibrate and broadcast to the eardrums in human ears. Then the human ears capture this vibration as a sound which consists of complex oscillations. The continuous sound which has appropriate pitch, volume, harmonious rhythm and interval is the basic components of music.

2.2.1 Sine Wave

Sine wave which has fixed frequency and volume is the simplest example of sound in a piece of music. Basically, sound is controlled by frequency, amplitude, and duration time in a sine wave. If a unit sine wave has the frequency with 440Hz in 10 second, it would sound like a clear buzzing for ten second. The frequency range of sound which can be perceived by human ear is 20 Hz to 20k Hz. The agaric expression of a sine wave is $P = A \times \sin(2\pi f \times t)$ while P represents the air pressure (DiCanio, 2015). In a sine wave, the amplitude is related to the loudness of sound while the frequency dominates the pitch of sound. Hertz and Decibel are used to be the unit of frequency and amplitude in the waveform of a sound respectively.

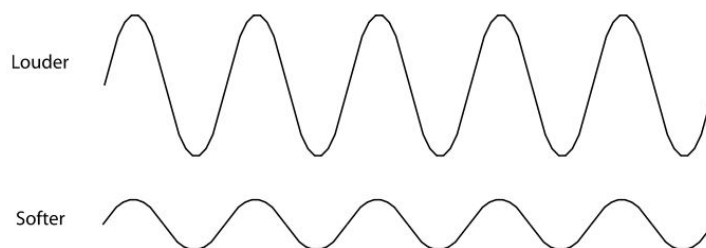


Figure 4. The effect of amplitude in Sine Wave on the loudness of sound (Schmidt-Jones, 2007)

2.2.2 Harmonics

A musical sound is usually the combination of many sine waves with different frequencies. Among those frequencies the lowest one is fundamental frequency while the multiples of this specified frequency are the frequencies of harmonics (Bain, 2003). Figure 5 shows the relationship between harmonics and the fundamental frequency in musical notes of string.


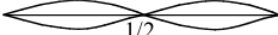




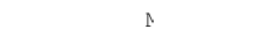

Note	Frequency	Harmonic	Diagram of vibrating string
low low low A	$f = 55 \text{ Hz}$	fundamental	
low low A	$f = 110 \text{ Hz}$	second	
low E	$f = 165 \text{ Hz}$	third	
low A	$f = 220 \text{ Hz}$	fourth	
middle C [#]	$f = 275 \text{ Hz}$	fifth	
middle E	$f = 330 \text{ Hz}$	sixth	
approx. middle G	$f = 385 \text{ Hz}$	seventh	
middle A	$f = 440 \text{ Hz}$	eighth	

Figure 5. Fundamental and harmonics of string notes (Petersen, 2001)

The sounds of different instruments could be distinguished by human ears because they have different harmonics and envelope of volume. For example, the variation of pressure against time in the waveform of flute, oboe and violin is shown in figure 6. The volume and fundamental frequency of these instruments may be same, but their harmonics are different. These features could be used to synthesize the instrumental music from bio-signals as different timbres.

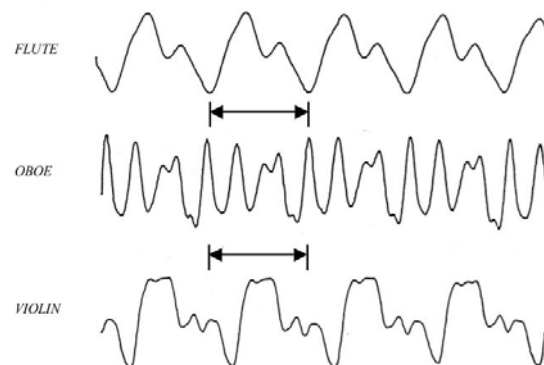


Figure 6. Sound waveform of different instruments (Petersen, 2001)

2.2.3 Octaves

The space between one frequency of a pitch and its half or double is an octave in the musical notation. In the typical musical scales, one octave consists of twelve notes, so the interval between the initial C and the final C is one octave. The relationship

between the notes in octave and frequency is shown in Figure 6. For example, the 4th octave A is fit as 440 Hz while the 3th octave A is set as half of 440 Hz, which is 220 Hz.

	OCTAVE NUMBER								
	0	1	2	3	4	5	6	7	8
C	16.3516	32.7032	65.4064	130.813	261.626	523.251	1046.50	2093.00	4186.01
C#	17.3239	34.6478	69.2957	138.591	277.183	554.365	1108.73	2217.46	4434.92
D	18.3540	36.7081	73.4162	146.832	293.665	587.330	1174.66	2349.32	4698.64
D#	19.4454	38.8909	77.7817	155.563	311.127	622.254	1244.51	2489.02	4978.03
E	20.6017	41.2034	82.4069	164.814	329.628	659.255	1318.51	2637.02	5274.04
F	21.8268	43.6536	87.3071	174.614	349.228	698.456	1396.91	2793.83	5587.65
F#	23.1247	46.2493	92.4986	184.997	369.994	739.989	1479.98	2959.96	5919.91
G	24.4997	48.9994	97.9989	195.998	391.995	783.991	1567.98	3135.96	6271.93
G#	25.9565	51.9131	103.826	207.652	415.305	830.609	1661.22	3322.44	6644.88
A	27.5000	55.0000	110.000	220.000	440.000	880.000	1760.00	3520.00	7040.00
A#	29.1352	58.2705	116.541	233.082	466.164	932.328	1864.66	3729.31	7458.62
B	30.8671	61.7354	123.471	246.942	493.883	987.767	1975.53	3951.07	7902.13

Figure 7. Octave and Frequency in Piano keyboard
(De Leon, 2000)

2.2.4 ADSR envelope

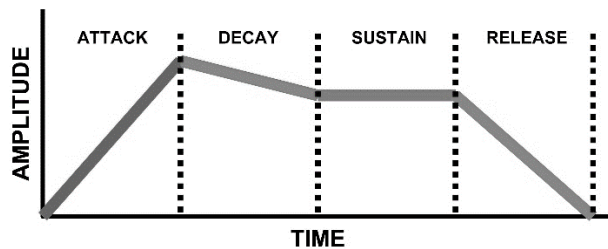


Figure 8. The amplitude change of ADSR envelope
(Mathew, Abraham & Scaria, 2015)

During the performance of musical instruments, musical tones usually take certain time reaching to the full volume then decay to zero gradually. This characteristic of tone makes the music soft and graceful for audience to hear. The rise period and decline phase are called attack and decay stage while the stable period of tone which hold by the executant is the sustain stage. At last, the tone would decay to zero in the release stage (Mathew, Abraham & Scaria, 2015). The whole process of volume change is named as Attack-Decay-Sustain-Release (ADSR) envelop.

2.3 Sound Synthesis Tools

2.3.1 Max/Msp

Max/Msp is a programming environment of a software developed firstly in 1980 at IRCAM for interactive system of computer music. It is currently used for signal generator, user interface and digital operator for real-time synthesis system of interactive music. In this system, the input signals could be interpreted well by creating connections between different objects in the edit model (Wang, 2008). The widgets such as graphs, knobs and message boxes, could be manipulated in interactive interface in the run model. Therefore, it is a very popular programming

system for music synthesis.

2.3.2 Pure Data

Pure Data is a programming language which is similar to Max developed by Miller Puckette for interactive computer music in 1996. It is an open source software designed to process audio data and control flow rate through a public application programming interface. Therefore the language C, python and Scheme could be used by developers to add their own audio control blocks (Farnell, 2006). As there are various external libraries available, the Pure Data is a common real-time programming environment for audio processing.

2.3.3 MATLAB

MATLAB is a modern programming language environment originally produced as matrix accessing software by Mathwork. It mainly focuses on the numerical computing achieving matrix manipulation, algorithm implementation, plotting functions and interfaces with other languages such as C and python. It also has collecting packages for specific applications as toolbox which is applied for simulation, symbolic computation, signals processing and engineering design (Houcque, 2005). Other functional package such as Simulink, is usually used for dynamic and embedded systems in graphical simulation field. Accordingly, it also includes the functions and toolbox for processing audio data and audio streaming packages for music synthesis.

2.3.4 Synthesis Toolkit

Synthesis Toolkit is a programming language for real-time musical synthesis introduced by Perry Cook in 1990. It can be compiled by any systems with C or C++ for audio digital signal processing in a fast prototyping environment. There are a variety of music synthesis algorithms compatible with STK for operating real-time sound and MIDI input signals on many platforms such as Tcl and Tk (Cook & Scavone, 1999). For example, there is a physical model of object named Mandolin for string instrument in STK. This model can be used to generate audio and for sound synthesis with C++ or Java language by programmers in real-time.

2.3.5 Super Collider

Super Collider is a programming language and audio programming environment initially published by James McCartney in 1996, mainly used for real-time sound synthesis and Algorithmic Composition. Since then it has gradually become a common system for sound development and operation used by scientists and artists. Its environmental structure has been divided into two parts since the release of the version 3, the server (scsynth) and the client (sclang). The newest version Super Collider 3.6.6 is released in November 2013 (Wang, 2008). Super Collider combines object oriented characteristics, features from functional programming

languages and syntax which is similar to C language. As the server supports simple C extension of application program interface, it makes this environment very simple to write efficient sound algorithm. It is also very easy to use other languages or applications to operate this language because all the operations of server would be through the external open source control, a standard for sending control messages for sound over the network.

2.3.6 Open Source Control

Open Source Control (OSC) is considered as a protocol for real-time control of computer music synthesizer. It has been widely applied in the design of hardware and software since it is firstly released by Matthew Wright in 1997. OSC is not only adaptable with IP transport in a LAN currently but also matches with other transporting methods, such as serial port, Wi-Fi and USB (Arslan et al., 2005).

2.3.7 MIDI

Musical Instrument Digital Interface (MIDI) is an electrical communication protocol defining various notes and playing codes for musical instruments. It could connect, adjust and synchronize the operation of computer, cellphone or electronic musical instrument for data exchange. It emits the control signals of musical volume, tremolo and phase instead of sound in the computer. MIDI standard is published by the Engineer Dave Smith in audio engineering association in 1983. It enable the multi-control and message exchange among computer, sound card, synthesizer, electrical musical instrument including electronic drum and keyboard, and synthesis of analog music realistically (Meister & Errede, 2011). Many musical formats are constructed based on MIDI. These files are usually only dozens of kilobytes, but enable play an intact piece of music by electronic musical instrument.

2.4 Related Researches

2.4.1 Music from Bio-signals

The project of 'Music from Bio-signals' was initially commenced by two engineering students Darian Tregenza and Robert Wood in 2011. In 2013, another student Nicholas Rose continued to conduct the previous research forcing on the hardware design of a data collection device (Rose, 2013). In 2014, an engineer student Semisi Finau went on this research concentrating on the hardware design of data acquisition, signal conversion and transmission from bio-signals for further music synthesis in the computer (Finau, 2014). So far these researches with the same title of this project are all talking about the techniques of acquiring data of bio-signals from human body and the approaches of analog-digital conversion. The majority of these designs are about hardware design while software design is rarely mentioned. Therefore, the music synthesis from bio-signals in the computer and the production of continuous music could fill the gap of these researches.

2.4.2 Interactive Instrument Technology in the Musical Performance

The BioMuse system developed by Knapp and Hugh in 1990 could transform the data of neuroelectric and myoelectric signals to MIDI directly for musical instrument performance (Knapp & Lusted, 1990). Such interactive instrument technology interfacing with the Max environment is a useful tool in musical composition and rehabilitation field.

2.4.3 Digital Musical Instruments Driven by Bio-signals from Musicians

Two bio-musicians built a 'Bio-Orchestra' using EEG and EMG signals based on MATLAB and Max/Msp through sound synthesis algorithm in 2010. The two digital musical instruments that driven by bio-signals are MIDI instrument and accordion (Arslan et al, 2010). The algorithms in this study are written in MATLAB while the data acquisition codes are made in C ++. In addition, a Simulink block is also used for adjusting variable objects and parameter from bio-signals and digital instruments.

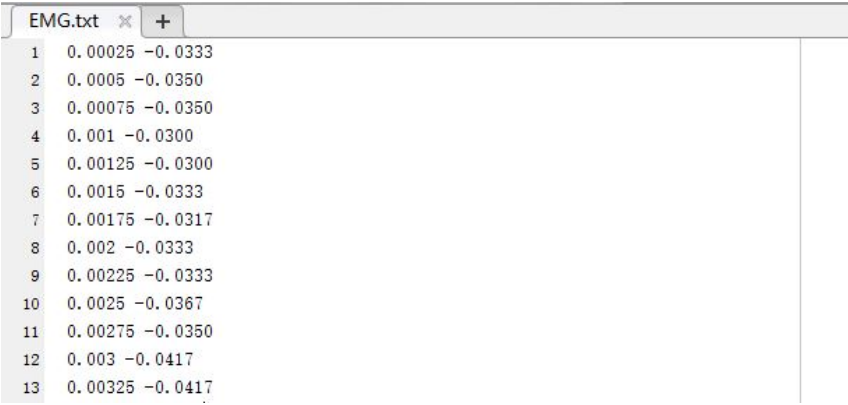
3.0 Project Design

3.1 Sound from static data

In order to investigate the characteristics of bio-signals, basically we can start to generate sound from static data. From various kinds of bio-signals, we choose EMG signals as the input data because EMG signal is non-periodic. The uncertain component in EMG signal is easier than the periodic signal to produce random notes, such as ECG signals during the process of music composition. Meantime, the software of MATLAB is used as the main programming tool for music synthesis, since it contains many useful functions and toolboxes for calculation and analysis.

3.1.1 Test EMG signals

A piece of normal EMG signals could be easily downloaded from the database of *PhysioBank*, which is a large online records of physiological signals used by biomedical research communities.



Line	Channel 1	Channel 2
1	0.00025	-0.0333
2	0.0005	-0.0350
3	0.00075	-0.0350
4	0.001	-0.0300
5	0.00125	-0.0300
6	0.0015	-0.0333
7	0.00175	-0.0317
8	0.002	-0.0333
9	0.00225	-0.0333
10	0.0025	-0.0367
11	0.00275	-0.0350
12	0.003	-0.0417
13	0.00325	-0.0417

Figure 9. A piece of digital EMG sample
(Overall data has 50860 sets of numbers)

The EMG sample in the testing file 'EMG.txt' has two channels. The left channel is the time with 12.715 seconds (sampling rate is 4000Hz), while the right channel is the corresponding voltage at that moment. The data from right channel could be picked out and played using 'sound' function in MATLAB to show the sound of raw digital EMG signals.

3.1.2 Create notes

Based on the characteristics of bio-signals, either the standard deviation or root mean square could be used as the algorithm to analysis digital data. In this step, root mean square is applied on the analysis. In the music theory, sine-wave is the simplest sound. The frequency of sine-wave determines the pitch of sound while the amplitude of sine-wave determines the volume. Each octave has a specific frequency range. For example, the frequency of 3rd octave range from 130.813 Hz to 246.942 Hz. Since there are twelve notes in each octave, the frequency increase from the first note to

the last one with a multiplier of $2^{\frac{1}{12}}$ between two adjacent notes. Figure 10 illustrates the flow diagram of making simple notes in MATLAB.

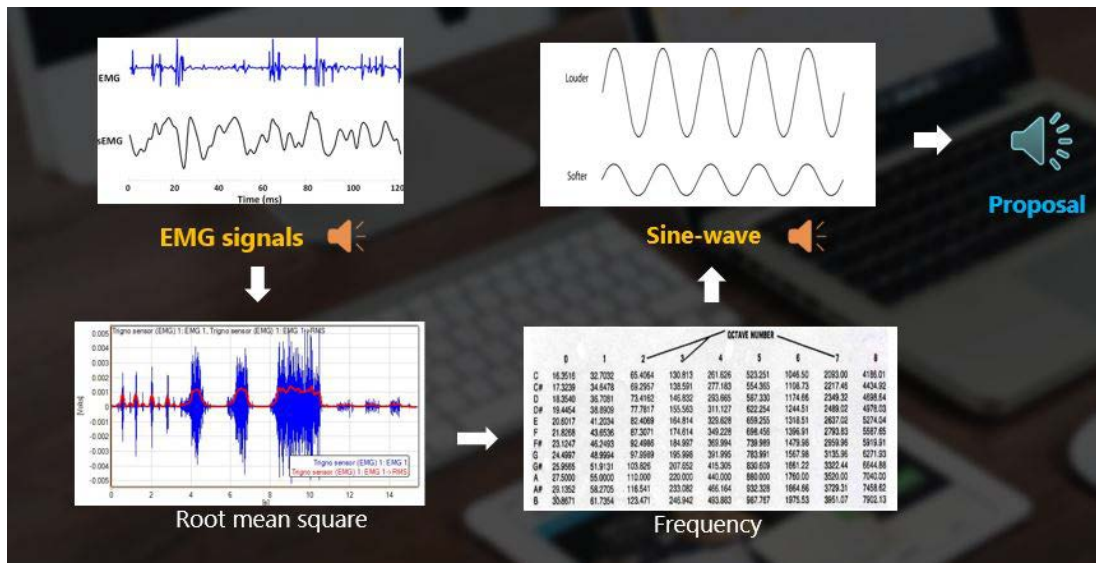


Figure 10. The flow diagram of making notes from EMG signals

Since root mean square is a very common mathematical algorithm, we use it directly in the project design, the formula is shown in Figure 11.

$$x_{rms} = \sqrt{\frac{1}{N} \sum_{i=1}^n x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \dots + x_n^2}{N}}$$

Figure 11. Calculating the root mean square

Firstly, in order to keep each musical note lasting for nearly one second, the digital EMG data in Figure 8 is divided into 16 average pieces because the overall dataset lasts for 12.715 seconds. It would be easier for human ears to distinguish the quality of sound in one second with each note. For each piece of data, the root mean square is used to calculate and derive a number. Since sixteen values would be achieved, a special number will be used to multiply by those sixteen values respectively to acquire final sixteen values which should be in the range of 20 to 20000. This special number is chosen through several computational tests depending on the final values which need to be in the frequency range of audible sound (the sound frequency ranges from 20Hz to 20 kHz). Secondly, the sixteen values of frequency would be added into sine-waves. It means the frequency of sine-wave will be manipulated by those numbers produced from the digital EMG signals. Since the audible sound has a large frequency range, the result sound is not expected to be very harmonious and beautiful, might be a little a bit sharp or uncomfortable for human ears.

3.2 Music synthesis from sound

Since the notes with different frequencies has been achieved from EMG signals, it is the time to simulation sound of musical instruments. In the music theory, harmonics and ADSR envelope is the two main factors to affect the timbre of sound and to distinguish the type of musical instrument. Therefore, some modifications could be implemented on the notes through programming in MATLAB.

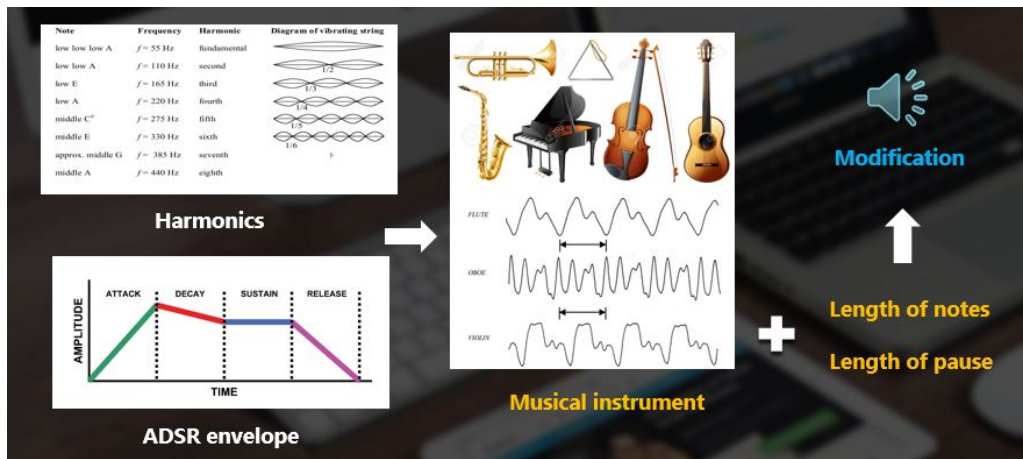


Figure 12. The overall process of music synthesis

3.2.1 Add harmonics

Each musical instrument has its particular harmonics while the harmonics in each octave of one instrument is also different. For example, the ratio of harmonics for three common instruments at Middle A (fundamental frequency is 440 Hz) is shown in Figure 13.

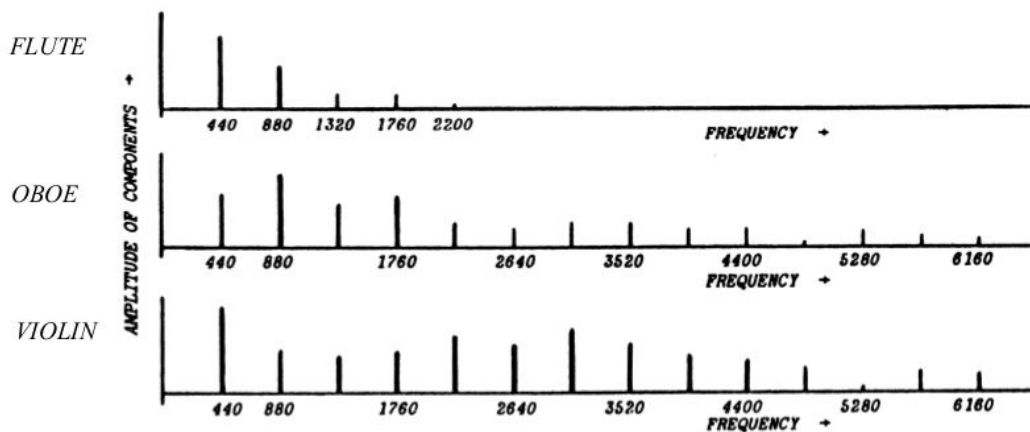


Figure 13. Ratio of harmonics for flute, oboe and violin at Middle A (440Hz)

(Petersen, 2001)

```

% make the basic note
t = ( 1:N) / fs;
note = zeros( 1, N);
for i = 1: numel( harmonics)
    note = note + harmonics( i) * sin( 2 * pi * freq * i * t);
end

```

Figure 14. Add harmonics on notes in a new function in MATLAB

Since there are sixteen notes that need to be modified, the calculation of harmonics for each note could not be completed in one formula. Therefore, a new function which is named as ‘*make_note (freq, harmonics, note_length, fs)*’, is made in MATLAB as shown in *Appendix 2*. Sometimes, the odd orders such as 3rd, 5th, and 7th should be ignored because those harmonics might affect the performance of simulation in the practice. To be specific, the harmonics with even and odd orders have different effect on the quality of sound through several simple sound test during the design. When the odd harmonics are strong while the even harmonics are weak, the sound is stiff and grotesque. When the even harmonics are strong while odd harmonics are weak, there is a sense of transparency and pureness in the sound. Therefore, only the harmonics with even orders would be useful in the codes.

3.2.2 Change ADSR envelope

The six notes in the proposal sound have no envelope. In order to add envelope into the notes, the feature of envelope in musical instruments needs to be analysis. Figure 15 shows the normal ADSR waveform and the envelope of strings. Generally, the strings such as guitar and violin, would take a while to reach the maximum volume, but the piano would reach to its peak volume in less than 125ms.

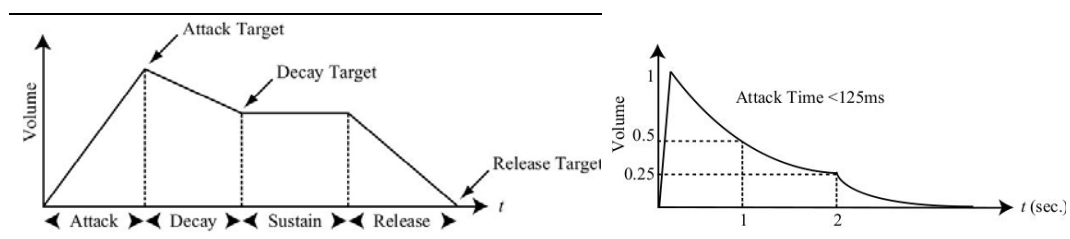


Figure 15. Sound analysis of different instruments (De Leon, 2000)

In MATLAB, ‘*linspace*’ is a common order to generate a specified number of points within a specified range. The calling method is ‘*linspace(x1, x2, N)*’, while *x1*, *x2* and *N* are the beginning value, the end value, the number of elements. Since the sampling numbers has been known as 4500 Hz, the amplitude of each stage of ADSR envelope could be set in the previous function ‘*make_note (freq, harmonics, note_length, fs)*’.

```

% impose envelope
A = linspace( 0,      0.7,      N * 0.1); %rise 10% of signal
D = linspace( 0.7,    0.5,      N * 0.2); %drop of 20% of signal
S = linspace( 0.5,    0.3,      N * 0.4); %delay of 40% of signal
R = linspace( 0.3,    0,        N * 0.3); %drop of 30% of signal
-note = note .* [ A D S R];

```

Figure 16. Usage of 'linspace' to simulate ADSR envelope
(Whiting, 2014)

3.2.3 Length of notes and pause

Since the notes of musical instruments has been made, it is necessary to define the rhythm of music. The rhythm could be set by changing the length of notes. In musical theory, notes could be divide into semibreve, minim, quarter, etc. Semibreve is the longest note in these notes while the length of minim is half of the semibreve. Other notes would continue to reduce in half than the previous one. In this part of design, three different lengths are used on the musical notes with 0.25 second, 0.5 second and 1 second. If the semibreve is 1 second, then minim is 0.5 second while the quarter is 0.25 second. These notes is arranged in a particular order in a piece of music. This order could be decided by several attempts in the design to make it pleasing to the ears. The final order is shown in Figure 17. In addition, length of pause could also be set up to create blank note between these notes in a music rhythm. However, the blank note is not applied on this section.

```

%set the length of notes
beat_length = 0.25;
note_length = beat_length * 2 .^[ 0 0 1 2 1 1 0 0 2 1 0 0 1 1 1 2];

```

Figure 17. The arrangement mode of music rhythm

3.3 Streaming real-time sound

From the previous design, music with the feature of musical instrument could be produced from static data of EMG signals. However, the aim of this project is to achieve the real-time control of music from bio-signals. Therefore, there is a task of seeking a way to stream out real-time sound in MATLAB.

3.3.1 Principle of DSP system toolbox

In MATLAB, there is a useful toolbox named as digital signal processing (DSP) toolbox that could be used for playing audio data using the audio device in the computer. The particular object in this toolbox used is in this toolbox is named as 'dsp.AudioPlayer'. The flow diagram of operation process of this object is shown in Figure 18.

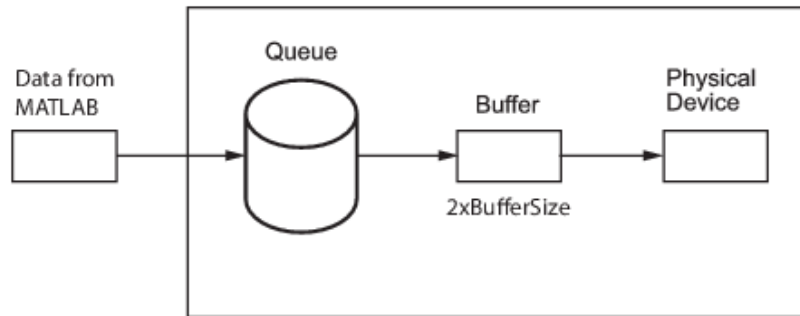


Figure 18. The flow diagram of 'dsp.AudioPlayer' object

The input data would be split into many frames before it enter into the object. Only one frame could be put into the 'Queue' before the frame in the queue is transmitted into the 'Buffer'. Once the first frame is sent out to the physical device through the 'Buffer', the 'Buffer' will be cleaned out while a new frame would enter into the 'Buffer'. Therefore, the 'Buffer' is used as a data memorizer in the object. The physical device could be a sound equipment or an audio device in the computer. Since there are a large numbers of data would be processed in the MATLAB, this object provides an efficiency method to stream audio data out to the audio devices.

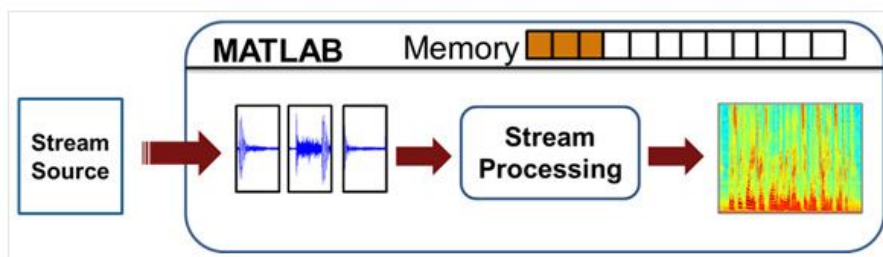


Figure 19. The conversion process of frames from digital data

```

AFR = dsp.AudioFileReader; % points to a default audio file
AP = dsp.AudioPlayer('SampleRate',AFR.SampleRate, ...
                    'QueueDuration',2, ...
                    'OutputNumUnderrunSamples',true);

while ~isDone(AFR)
    audio = AFR();
    nUnderrun = AP(audio);
    if nUnderrun > 0
        fprintf('Audio player queue underrun by %d samples.\n'...
                ,nUnderrun);
    end
end
pause(AP.QueueDuration); % wait until audio is played to the end
release(AFR);           % close the input file
release(AP);           % close the audio output device

```

Figure 20. A model of 'dsp.AudioPlayer' object in MATLAB
(MathWorks, 2016)

The definition of parameters in ‘*dsp.AudioPlayer*’ object is a critical part for the streaming process during the design. ‘*SampleRate*’ represents the number of samples per second while those samples would be sent to the audio device outside of MATLAB. Its default value is 44100 Hz while it is also a tunable value if it is predefined. ‘*BufferSize*’ is the size of buffer that is tunable when the ‘*BufferSizeSource*’ is set to be ‘*Property*’. It is an important parameter which affects the latency of the streaming data. The latency is the duration of device to empty the buffer when the buffer communicates with the physical device. Another parameter, ‘*QueueDuration*’ is also rated to the latency. It is the maximum length of signal which can be lagged by the data in seconds in the object. If the throughput rate of MATLAB is slower than that of physical device, there would be no sufficient data put into the buffer, then the underrun of buffer occurs. Therefore, the queue duration should be at least larger than buffer size. Moreover, ‘*OutputNumUnderrunSamples*’ could be used to monitor the underrun situation. A model of ‘*dsp.AudioPlayer*’ object is shown in Figure 19 while the application of this model is shown in *Appendix 4*.

3.3.2 Introduce ASIO driver

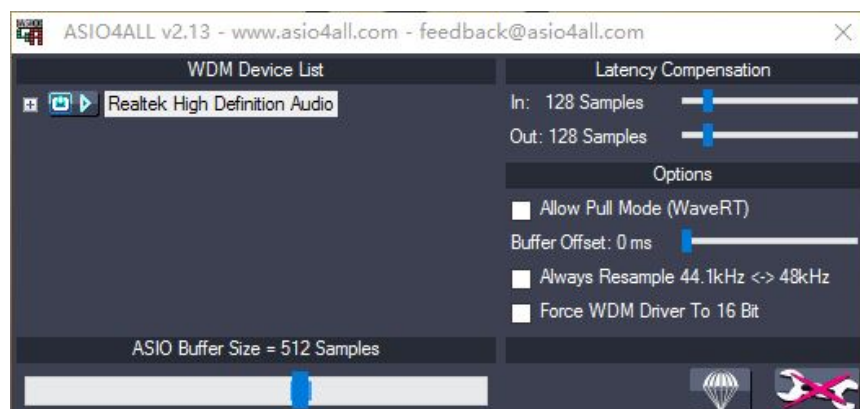


Figure 21. The main board of ASIO driver

The ASIO driver is a powerful interface between the software of sound synthesis and sound card in the computer. It could achieve a low latency and high-fidelity of sound streaming because it enables the sound card to access the physical device directly. In MATLAB, set the choice of ASIO driver instead of Windows’ direct sound driver by this path: *File > Preferences > DSP System Toolbox*, select *ASIO driver* in the list.

3.3.3 Test the function of DSP toolbox

Since the feature of ‘*dsp.AudioPlayer*’ object in the DSP toolbox has been illustrated while the interface of ASIO driver is ready, it is the time to test the performance of the audio streaming function.

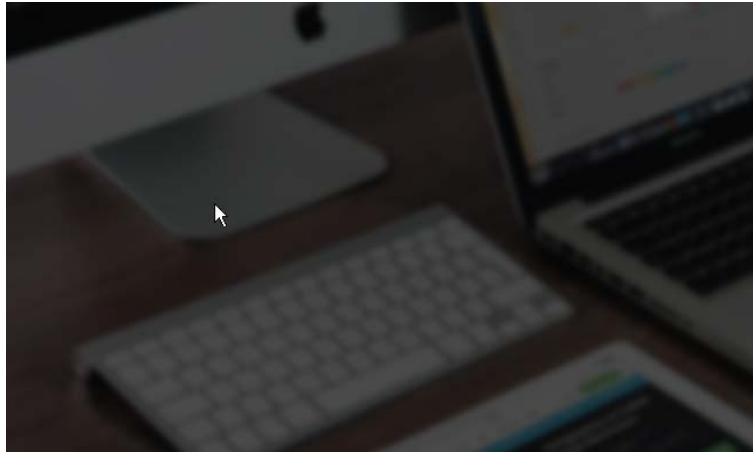


Figure 22. A mouse pointer in the computer screen

Firstly, a set of dynamic data is needed to simulate the real-time EMG signals from human body. Therefore the position of a mouse pointer in the computer screen is chosen as the streaming source to MATLAB. The order, '*get(0, 'ScreenSize')*' in MATLAB is used to calculate the length and width of the screen while the order, '*get(0, 'PointerLocation')*' is used to calculate the corresponding value of location of mouse pointer in the desktop. In the end, a 1×2 matrix could be acquired from the value of screen size divided by the value of pointer location. Therefore, the two numbers in his matrix ranges from zero to one. It could be considered that there is a coordinate system in the computer screen, the coordinate of left bottom point is the original point (0, 0), while the coordinate of right top point is (1, 1). When the mouse pointer moves in the screen, there will be two numbers recorded in MATLAB.

The location coordinate of mouse pointer, (x, y) ($0 \leq x \leq 1$, $0 \leq y \leq 1$) in the screen coordinate could be used to control the frequency range of sound in the test. For example, if the frequency range is defined from 131 Hz to 494 Hz (include 3rd and 4th octave), the output frequency is derived from $f = 131 + y \times (494 - 131)$. Since there are two numbers in the coordinate value, the other number could be used to control the volume of sound (amplitude value of volume).

As the controller has been defined well, the digital sine-wave is used as the input data to create simplest sound in MATLAB. The flow diagram of the test is shown in Figure 23. In the ASIO driver, the latency compensation is set to be 128 samples while the sampling rate is set to be 48000 Hz. The expected outcome of this test is that sound changes from low pitch to high pitch when the mouse pointer moves from bottom to the top of screen. Meanwhile, the sound changes from small volume to large volume when the mouse pointer moves from left to the right of screen.



Figure 23. Test the performance of DSP toolbox

3.3.4 Minimize the signal discontinuities of sound

The potential failure of the test might be that there are some signal discontinuities which make the output sound discontinuous in the streaming process.

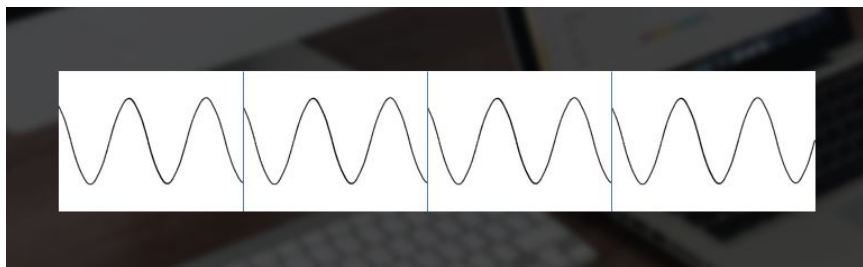


Figure 24. The signal discontinuities between different frames in the output sound

Since the first frame is taken out of buffer, another frame which is derived from the start of the queue is put into the buffer, such problem is shown in Figure 24. The sound would have the squeak or an uncomfortable noise. To solve this problem, the end value in the first frame needs to be calculated while the next frame of sine-wave would start from this value setting in the formula of output audio.

Another method is to chop the original data into many pieces. When the information of first frame is taken from the streaming source, the program will chop out that part of frame from the original data with same length of first frame. Then the output audio would be continuous when the second frame comes from the original data.

3.4 Music composition from real-time sound

As the real-time sound has been streamed out from the test of the function of DSP toolbox, the programming codes made for the composition from the static EMG data could be applied on this sound. According to the music theory, musical notes could be produced from the sine-wave and ADSR envelope.

3.4.1 Apply the previous model of music composition

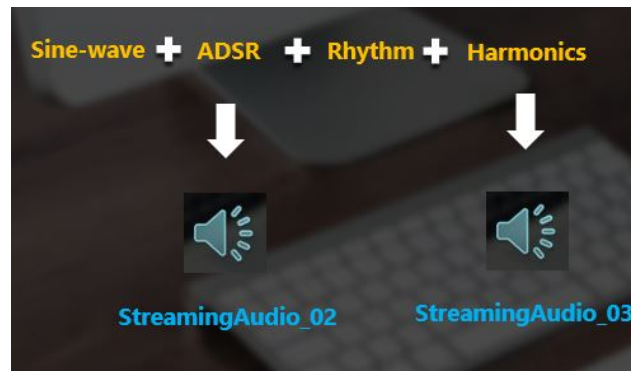


Figure 25. The application of previous music synthesis theory

Since the sine-wave is existing in the real-time sound, particular envelope could be added to making notes as shown in Figure 24. The controller of these notes is still the movement of mouse pointer. However, the difference of this sound from the previous one is expected to be that there will be a constant pitch when the mouse moves in range of a vertical axis in the computer screen. It means that those notes lies on the vertical axis in the screen with an increasing trend of pitch from the bottom to the top. Specifically, there are fourteen notes started from major C in 3rd octave (131Hz) to major B in 4th octave (494Hz).

```
- harmonics = [0.467 0.3 0.1 0.1 0.003];  
- beat_length = 0.5;  
- note_length = beat_length * 2 .^[2 1 1 0 0 1];
```

Figure 26. The definition of rhythm and harmonics

Regarding to the simulation of musical instrument, harmonics are also need to be defined as show in Figure 26. The five numbers are the ratio of every order of harmonics in the flute. Similar to the composition from static data of EMG signals, the note length is used to define the rhythm of music as show in Figure 26. Moreover, the ADSR envelope and harmonics are predefined in a new function named as '*frame_note (freq, harmonics, note_length, Fs, samplesPerFrame)*' shown in Appendix 5.

3.4.2 Introduce pentatonic scale

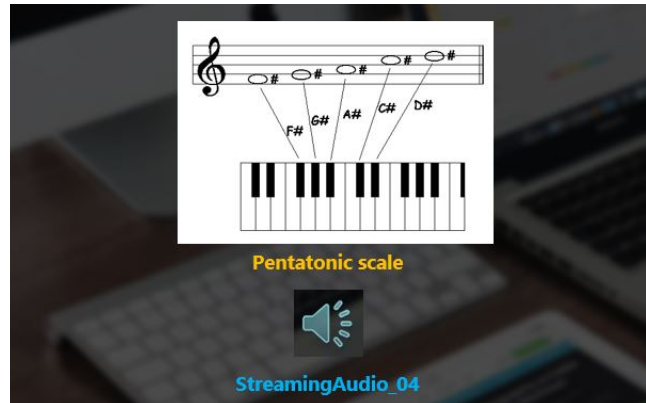


Figure 27. The pentatonic scale in piano keyboard

In the previous section of synthesizing music from static data, the order of different notes is preset before played. Meanwhile, the arrangement of those notes is the job of composer in the music field. However, those notes in this design would be originally produced from EMG signals in human body. According to the characteristic of EMG signals, there is no particular order for those signals. The random arrangement of these notes doesn't sound so harmonious. Therefore, pentatonic scale is introduced. The biggest feature of pentatonic scale is that it could stream a harmonious song when the notes are played randomly. Basically, the general twelve notes in one octave is expressed as C, C#, D, D#, E, F, F#, G, G#, A, A# and B. Pentatonic scale consists of five notes which belong to one of the octaves. It picks particular five notes from the twelve notes in one octave which allow the random arrangement to make a beautiful song. To be more perceptible, the pentatonic scale is the black notes in a keyboard of the piano as shown in Figure 26. It can be classified as two kinds, the major pentatonic and the minor pentatonic. For example, the notes of C major pentatonic notes are C, D, E, G, A while the notes of A minor pentatonic are A, C, D, E, G. With this useful scale, the random movement of mouse pointer could compose a euphonic song in the computer.

3.4.3 Implement multiple controls

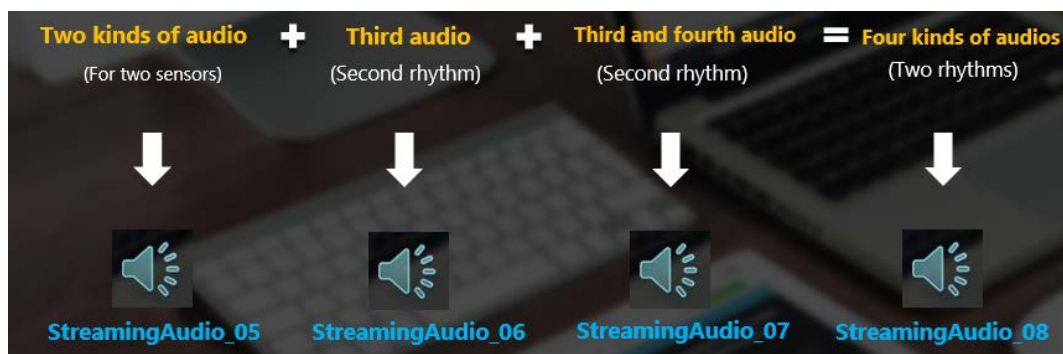


Figure 28. The design of multiple controls of music

From the previous study, one set of data could control the fundamental frequency of

one sound. It means that if there is one sensor attached on human body, the fundamental frequency of one sound could be manipulated by people. However, one sensor might not be enough for people to play while single sound could only be called as drum in the music. Therefore, multiple controls of sound is necessary to be designed in this project.

Since the sound of flute has been produced in the previous program, a similar sound of flute with different frequency range and note numbers could be easily generated in the same way. The combination of two similar sound could produce a special effect on the eardrum of human. The new sound could also rich the timbre of music. Moreover, a third sound of piano is produced based on the music theory of musical instruments. It is could be achieved by changing the ADSR envelope and harmonics of the parameters. Meanwhile, a new rhythm is used with a faster speed, which will make it be easier to distinguish from the sound of flute. In the same way, the fourth sound which is similar to the third one is produced, and combined together. However, the control mechanism is opposed to the first two sounds. The movement in horizontal axis in the screen is used to control the fundamental frequency of sound while the movement in vertical axis in the screen controls the volume.

3.5 Wireless transmission Test

A semi-finished hardware is produced by previous students in the study of 'Music from Bio-signals'. Two of that hardware are the Arduino which is used as a micro-controller and the Bluetooth shield which works for the wireless transmission to convert the data through Arduino to the computer. In order to test the connection between software and hardware, the setting of Bluetooth needs to be opened in the computer.

3.5.1 Read data from hardware

The program written in Arduino convert the voltage of two input pins on the Bluetooth shield to the computer. For receiving the data in the computer, the programming codes are written in MATLAB is shown in Figure 29.

```
%% Rececive data from arduino through bluetooth shield
bt = Bluetooth('HC-05',1);
fopen(bt);
```

Figure 29. Read data from Arduino

3.5.2 Stream out sound from hardware



Figure 30. Test the wireless transmission

Since the DSP toolbox has already been tested to stream out real-time music successfully, the program has been made could also be useful in this process. For the running of hardware, a power supply is needed to active the two devices. Firstly, the control of signal sound by one input pin is tested as shown in Figure 30. Secondly, the double control of sounds by two input pins is implemented to stream the real-time music out. Since there is no input signals onto the hardware terminals, the voltages on the two pins would be constant values. Therefore, the expected result is that the streaming music from one pin or two pins in the hardware terminals would has a constant pitch of sound.

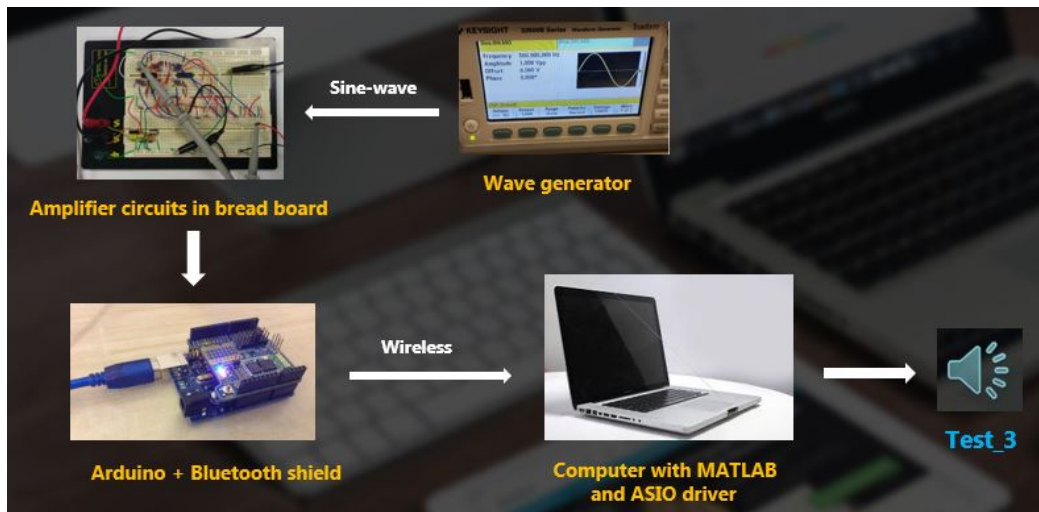


Figure 31. Test the controllability of hardware terminal for real-time music

In the end of this design, a tunable sine-wave is generated from a wave-generator to test the streaming music in the computer through hardware with the amplifier circuits, Arduino and Bluetooth shield as shown in Figure 31. The expected result is that the pitch of streaming music in the computer could be adjusted in real-time by the change of frequency in wave-generator.

4.0 Results

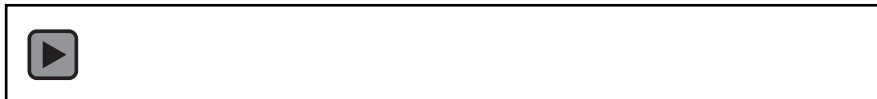
Since all the results of programming codes are sound and music, there is no picture that will be shown in the result. Those audio files from static EMG signals are produced by MATLAB codes in the computer while the real-time sound or music is recorded by the application software in a cell phone. All the audio files are the format of *.mp3* and have been embedded into the *.PDF* file.

Before the play of these audio files, a PDF reader software '*Adobe Acrobat XI Pro*' and an independent audio player software '*Adobe Flash Player*' need to be installed firstly in the computer.

4.1 Music from static data

4.1.1 Simple sound with notes

Firstly, the original piece of digital data from EMG signals shown in Figure 8 will be played in Audio 1:



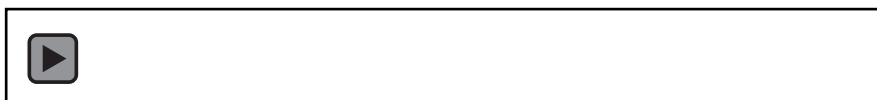
Audio 1. A piece of raw digital data of EMG signals (6.36 second)

It can be clearly heard that this piece of audio sounds very noisy, this audio could be used to produce a better sound.



Audio 2. A piece of sine-wave with a frequency of 440 Hz (5 second)

In MATLAB, a piece of sine-wave could be generated with specific duration and pitch, the sine-wave with a frequency of 440Hz in Audio 2 sounds like a whistle or a buzzer heard by human ears.



Audio 3. The proposal sound with simple notes (9 second)

In audio 3, 16 numbers are calculated from the digital data of EMG signals ((6.36 second) through the algorithm of root mean square. These numbers are used to control the frequency of the sine-wave in Audio 2. The length of every note is 0.5 second. The frequency of these sixteen notes are 297Hz, 165Hz, 392Hz, 357Hz, 141Hz, 802Hz, 1056Hz, 347Hz, 244Hz, 225Hz, 181Hz, 177Hz, 285Hz, 391Hz,

144Hz, 144Hz.

4.1.2 Simulating musical instruments

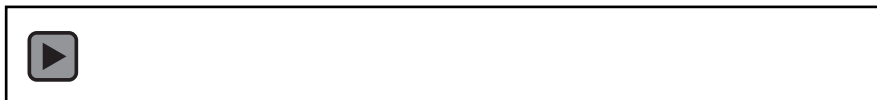


Audio 4. Simulating the sound of flute (9.6 second)

In audio 4, some modification is made from the proposal sound in Audio 3 for simulating the sound of flute. Specifically, the frequency range of these notes is limited into octave 5 (523Hz- 988Hz) while major scale is used to create seven keys in one octave. There are three kinds of note length in this audio (0.25 second, 0.5 second and 1 second). The ratio of harmonics of flute starting from the fundamental frequency (1st order) are 0.467, 0.3, 0.1, 0.1, and 0.003. The sampling frequency is set to be 8000Hz.

4.2 Music from real-time data

4.2.1 Streaming real-time sound

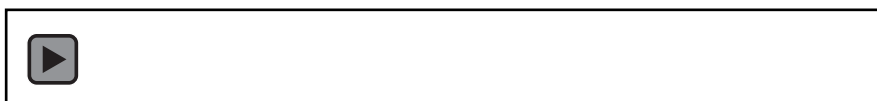


Audio 5. Streaming dynamic sound controlled by computer mouse pointer

This piece of sound is recorded from the running of MATLAB codes controlled by the mouse pointer in the computer. The movement of mouse pointer generates two dynamic numbers of the horizontal and vertical coordinates of pointer ranged from 0 to 1 in the computer screen. The two numbers are added on the sine-wave for the control of pitch and volume of sound.

From the Audio 5, firstly it can be clearly heard that the pitch of sound is gradually increased when the pointer is moved from the bottom to the top of computer screen. When it moves up and down faster, it sounds like a signaling whistle or the rolling wind. Secondly, the volume increases when the pointer is moved from the left to the right of the screen. Equally, the volume decreases when the pointer is moved from right to the left.

4.2.2 Streaming sound with notes



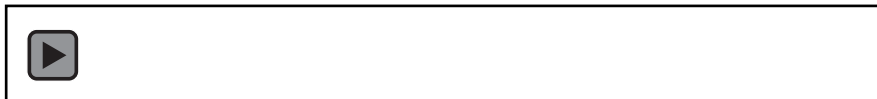
Audio 6. Real-time sound with notes

This piece of sound is modified based on the sound in Audio 5. A simple envelope

is add on the sine-wave illustrated in Audio 5 controlled by the coordinates of the mouse pointer. The control mechanism is same to the Audio 5.

From audio 6, it can be heard that different pitches of note can be clearly distinguished while the frequency ranges from 300Hz to 600Hz. In the envelope in every note, the attack time is $T_A = (500/48000) = 10.42\text{ms}$ while the decay time is $T_D = (5000/48000) = 104.17\text{ms}$. The sustain time which is the time when the volume is held on in a constant value is $T_S = 1 - T_A - T_D = 855.41\text{ms}$.

4.2.3 Simulating sound of flute

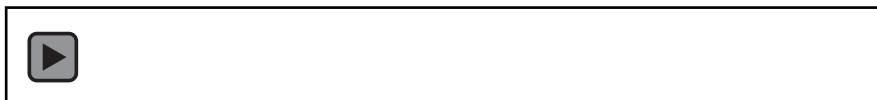


Audio 7. Simulating real-time sound of flute

In order to simulate the timbre of flute, specific ADSR envelope and harmonics are add into the sound with notes in Audio 6. The envelope. The envelope is made of straight lines which consist of 10% (0 – 0.4), 20% (0.4 - 1), 40% (1.0 – 0.9), 30% (0.9 – 0) (percentages represent horizontal axis while decimal numbers represent vertical axis). In addition, the ratio of different orders of harmonics in notes are 0.467(1st order), 0.3(2nd order), 0.1(3rd order), 0.1(4th order) and 0.003(5th order) respectively.

In Audio 7, it can be clearly heard that there is a rhythm looping in this piece of sound because the length of note is predetermined in the codes. The length notes in every six notes are 2 second, 1 second, 1 second, 0.5 second, 0.5 second and 1 second separately.

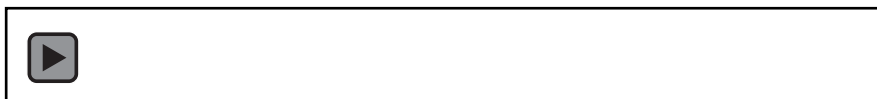
4.2.4 Streaming sound of flute in C major pentatonic



Audio 8. Streaming sound of flute in C major pentatonic

In Audio 8, the sound of flute is very harmonious compared with the Audio 7. Since the C major pentatonic is used in the simulation process, the timbre of notes are much more similar to the notes of real musical instruments.

4.2.5 Combination first and second audios

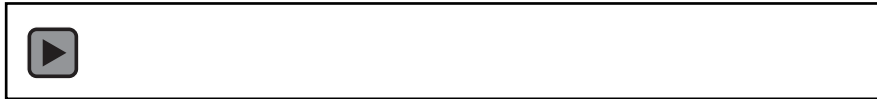


Audio 9. Combination of first and second sounds

In Audio 9, the two kinds of sound are all the simulation of flute with same ADSR envelope and harmonics but have a little difference in the definition of notes. The

frequency range of the first sound is from 130.81 Hz to 493.88 Hz (3rd octave and 4th octave) with ten notes while the frequency range of second sound is from 261.63 Hz to 493.88 Hz (4th octave) with 5 notes. However, they still have same length of notes which means that they have same rhythm. This combination is also an improvement optimization of timbre in Audio 8.

4.2.6 Introduce third sound of piano with a new rhythm



Audio 10. Third sound simulating the timbre of piano

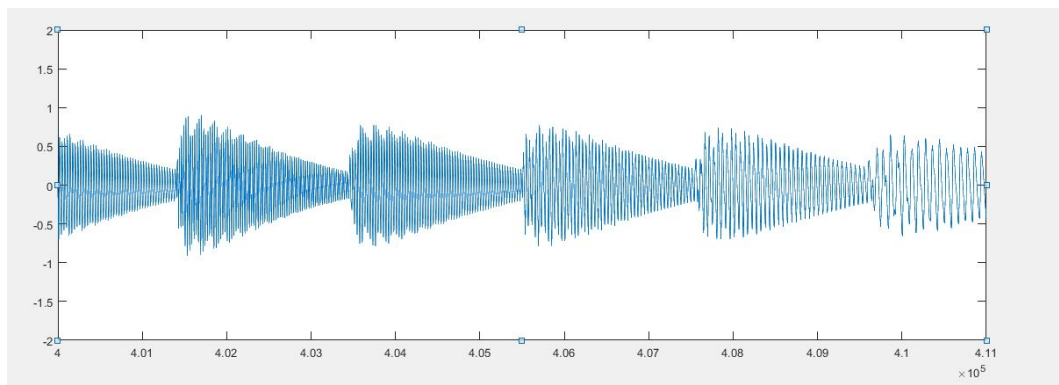


Figure 32. The expected spectrogram of piano

In Audio 10, the sound of piano is introduced for the further synthesis of music. This new sound also use the C major pentatonic to simulate the sound of real musical instrument of piano. The ratio of the harmonics of notes in different orders are 0.1842 (1st order), 0.1053 (2nd order), 0 (3rd order), 0.2256 (4th order) and 0 (5th order) 0.0827 (6th order), 0 (7th order), 0.0300 (8th order) respectively. The envelope of piano note is more complicated than the envelope of flute because of the special structure of piano. The data of envelope is shown in the codes in Appendix 8. Moreover, the control mechanism is also different from that of flute while vertical coordinate is used to control the fundamental frequency of note instead of the horizontal coordinate in the computer screen. Identically, the volume control is also exchanged from that of flute. The length of piano note is also halved, therefore this sound has a double rate of rhythm compared with the sound of flute.

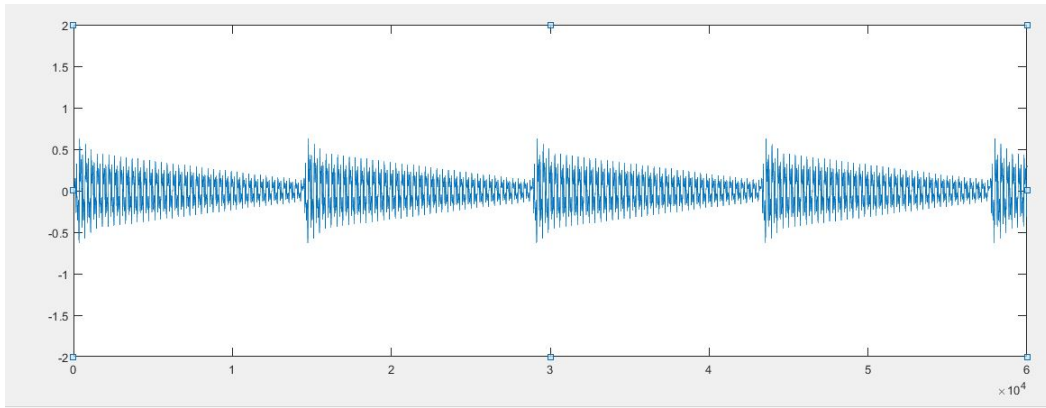
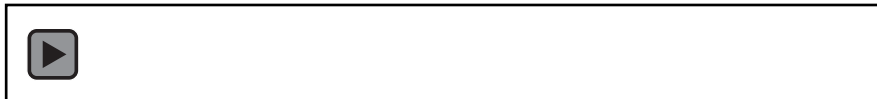


Figure 33. The simulating spectrogram of piano

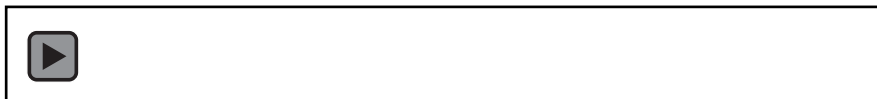
4.2.7 Combination third and fourth audios



Audio 11. Combination of third and fourth sounds

The Audio 11 is similar to the Audio 9 while two sounds of piano are combined together. Therefore, the two sounds of piano have same envelopes. However, the harmonics of the fourth sound is a little different in order to rich the timbre of the combination effect. The ratio of the harmonics of the fourth audio in different orders are 0.2078 (1st order), 0.1333 (2nd order), 0 (3rd order), 0.1255 (4th order) and 0 (5th order) 0.1804 (6th order), 0 (7th order), 0.0157 (8th order) respectively. Regarding to the pitch of sound, the frequency range of this combination is also different. In the note of the third sound, the frequency ranges from 65.4 Hz to 246.94 Hz (2nd octave and 3rd octave) with ten notes while the frequency ranges from 65.4 Hz to 123.47 Hz with five notes (2nd octave). Since the third and fourth sounds have same rhythm, the note length of the combination in every ten notes are 2, 2, 1, 1, 1, 1, 0, 0, 0, 0, 1 and 1 (unit: second).

4.2.8 Play the four audios with two rhythms



Audio 12. The combination of the four sounds

Now all the four sounds are combined together in Audio 12. This music have two musical instruments as flute and piano with two different rhythms. The timbre can be distinguished by human ears. The control mechanism is that the movement of mouse pointer in the vertical coordinate in computer screen controls the fundamental frequency of first and second sound of flute and volume of the third and fourth sound of piano while the movement in horizontal axis controls the volume of first and

second sound of flute and the fundamental frequency of third and fourth sound of piano.

4.3 Music test through wireless transmission

4.3.1 Test the program from one input pin in Bluetooth



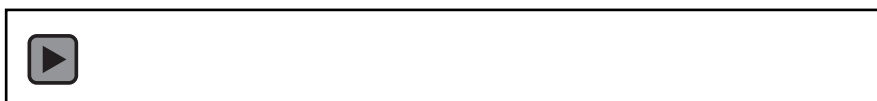
Audio 13. Test for one input pin on the Bluetooth

Since there is no input signals input pin, the voltage value is expected to be a constant number. From Audio 13, it can be clearly heard that the pitch of sound change a little bit at first, then doesn't change anymore.

4.3.2 Test the program from two input pins in Bluetooth

```
voltage0 = 1.99  outputvolt0 = 1.99
sensor1 = 404   output1 = 100
voltage1 = 1.97  outputvolt1 = 1.97
sensor0 = 410   output0 = 102
voltage0 = 2.00  outputvolt0 = 2.00
sensor1 = 403   output1 = 100
voltage1 = 1.97  outputvolt1 = 1.97
sensor0 = 412   output0 = 102
voltage0 = 2.01  outputvolt0 = 2.01
sensor1 = 406   output1 = 101
voltage1 = 1.98  outputvolt1 = 1.98
sensor0 = 412   output0 = 102
```

Figure 34. The dataset on the two input pins in Bluetooth



Audio 14. Test for two input pins on the Bluetooth

The result of the test of two input pin is expected to be the similar to Audio 13. In Audio 14, it can be heard that the sound of two different instruments are all steady at a particular pitch and don't change anymore.

4.3.3 Test the program using sine-wave input data through the bread board



Audio 15. Test for one input pin through the hardware

Since the bread board is connected to the A/D converter (Arduino) and Bluetooth shield with a sine-wave input produced by a wave generator, the pitch of sound should be tunable by the change of frequency through wave generator. However, the streaming audio in Audio 15 sounds like a random composition when the frequency is changed in the wave generator.

5.0 Discussion

5.1 Music from static data

The first step of this project is to explore the function of MATLAB whether it can be applied on the audio production through programming. Since the raw data of EMG signals inputted through microcontroller from sensors could be one or more set of digital numbers, such sample as a piece of static data is used for testing. From the proposal sound in Audio 3, the basic components of music, the musical notes are made from a piece of EMG signals. To be specific, the frequency of sound in music could be controlled by EMG signals. This function could be applied on the control of the pitch of sound in music composition of real-time data in the future. Therefore, it is a simple but really useful step for the design.

File number	Audio 1	Audio 2	Audio 3	Audio 4
File Name	EMG signals	Sine-wave	Sound from static data	Music synthesis from sound
Duration	6.36 second	5 second	9 second	9.6 second
Parameters	Random voltage	440Hz	Random frequency	Appropriate frequency range, rhythm and note length
Performance	Noisy	Constant buzzing	Simple notes	A piece of harmonious song

Table 1. Compare the sounds from static data in audio files

However, the simple notes are not sufficient to be called as music. So the concept of ADSR envelope and harmonics are introduced in the music theory while fundamental frequency is the 1st order of harmonic. Since the frequency of sound could be determined by EMG signals in Audio 3. This frequency is used as the fundamental frequency in the simulation process of flute, which is one of the normal musical instruments. Other parameters are predefined in the MATLAB codes, such as the ADSR envelope of flute and the note length which could affect the rhythm of music. Audio 4 shows good performance of the simulation of flute. Since the harmonics and ADSR envelope are the two main factors to distinguish different musical instruments while we have successfully prove it, any musical instruments could be simulated if the two parameters of them is known. The potential problem of the simulation for

other instruments might be that those parameters could not be achieved directly from existing articles. However, it is not difficult using MATLAB to analysis and acquire the ADSR envelope or harmonics from real musical instrument.

The Audio 3 and Audio 4 provide a significant preparation for the further research of music from real-time data, because theory of music composition from digital data in programming software such as MATLAB has been proved to be feasible.

5.2 Music from real-time data

The step of streaming continuous sound shown in Audio 5 is a critical part of this project, because sine-wave is the simplest sound in music theory. Since the sine-wave could be streamed out frame by frame through the DSP toolbox in MATLAB and ASIO driver, the programming codes we have created for music from static data could be applied on this model. However, the adjustment of parameters from the *dsp.AudioPlayer* function needs to be cautious because the latency and dropout of data need to be balanced when the buffer size and frame size are set. Both overrun and underrun of buffer should be prevented in the design process. As a result, the sound in Audio 5 is seamless without latency or dropout of data in the practical demonstration.

File number	Audio 5	Audio 6	Audio 7	Audio 8
File Name	Streaming real-time sound	Real-time sound with notes	Real-time sound of flute	Real-time sound of flute in C major pentatonic
Parameters	300 - 600Hz	300-600Hz 14 notes with simple envelope	Specific harmonics and envelope for flute	10 notes in pentatonic scale
Performance	Sound of rolling wind	Continuous simple notes	Random arrangement of flute notes with particular rhythm	Harmonious real-time sound of flute

Table 2. Compare the sounds from real-time data in audio files

Since the real-time audio could be streamed out in MATLAB, the programming codes of music from static data could be used on the DSP system model. The ADSR envelope and harmonics are created to simulate the sound of flute using the similar parameters in Audio 6 and Audio 7. However, the difference from the simulation from static data is that the pentatonic scale is introduced in Audio 8. As the input digital data could be random and unpredictable, such scale could make the random combination of notes sound like more harmonious than usage of the major or minor scale. The Audio 8 shows a good performance with the control of movement of mouse pointer in the computer screen. Such control could be used by one sensor in the hardware terminal, while the main control is about the fundamental frequency of sound and the supplementary control is about the volume of sound. However, the terminal sensor could not be only one, there might be three or sensors which could be attached on human body to manipulate on the composition. Therefore, multiple controls need to be built in the programming codes.

File number	Audio 9	Audio 10	Audio 11	Audio 12
File Name	Combination of first and second sounds	Third sound simulating the timbre of piano	Combination third and fourth sounds	The combination of the four sounds
Parameters	Different frequency range and note numbers	A new rhythm with an inverse control	Different frequency ranges and note numbers	Two sounds of flute and two sounds of piano
Performance	Richer timbre	A faster rhythm compared with that of flute	Richer timbre compared with signal sound	Real-time music with harmonious sound and rich timbre

Table 3. Compare the sounds from real-time data in audio files

For the music composition from real-time data, a few steps need to take attention in the design process. Firstly, another common musical instrument, piano is introduced into the design. Obviously, piano is a different type of instrument from flute. They belong to the percussion and woodwind respectively. Therefore, new envelope and harmonics are used for simulation. However, the envelope of piano sound could not be reproduced so well because of the complex structure of piano. The harmonics of piano are also complicated since the notes in different octave have different harmonics.

Therefore, only 2nd and 3rd octave are applied to serve as the bass part in the music synthesis. A new rhythm is also used to distinguish the timbre of from piano from flute. The different between the two instruments could be heard between the Audio 8 and Audio 10. Secondly, the combination of two similar sound with similar frequency but same harmonics and envelope has been proved to be effective to improve the quality of music if the Audio 9 and Audio 11 are compared. However, it is just a synthesis method of ‘beats’ in music theory, but the detailed mechanism haven’t been explored too much to control the performance of this technique. The only way implemented in this design is to combine the notes in 2nd and 3rd octave with 2nd octave for piano, then combine the notes in 4nd and 5rd octave with 4nd octave for flute. The result is surprisingly well but currently the operating mechanism of this method haven’t been controlled. Thirdly, as the four different kinds of sounds have already been obtained including two sound of flute and two sound of piano in Audio 12. The function of multiple controls have been achieved. When the hardware sends the data of EMG signals from four sensors, those data would be send into MATLAB with four digital numbers. In the programming codes in MATLAB, those four sets of numbers could be used to control the fundamental frequency of the four sounds as expected.

5.3 Music test through hardware in real-time

File number	Audio 13	Audio 14	Audio 15
File Name	Test for one input pin on the Bluetooth	Test for two input pins on the Bluetooth	Test for one input pin on the hardware
Parameters	No input signals with Arduino and Bluetooth shield	No input signals with Arduino and Bluetooth shield	Input sine-wave through amplifier, Arduino and Bluetooth shield
Performance	A signal sound with a constant pitch	Streaming out music with constant pitches	Streaming out music with random notes (sound is no tunable from hardware)

Table 4. Compare the music through hardware in real-time

Since the Arduino and Bluetooth shield have been built before the establishment of sensors, we firstly test the data commission from the input terminals on the Bluetooth

shield. Audio 13 and Audio 14 illustrate the same characteristic of the input pins on the Bluetooth shield. Both of the two output sounds have a constant pitch which means that the voltage in the input pins have a constant value. It is also a reasonable performance since we don't have any input signals yet.

On the next stage, the sine-wave is used as an input signals transmit through the circuit in bread board and Arduino as well as Bluetooth shield into the computer. However, the pitch of sound cannot be changed when frequency of sine-wave is changed in the wave-generator. The sound is still a random composition since we have an input of simple sine-wave with a fixed frequency. For a preliminary analysis of this problem, we know that the input data has thousands of digital numbers per one second but only two numbers are exported in the computer. Therefore, the output sound could not reflect the real feature of the input signals in the hardware terminal. For further research, the first problem needs to be solved in this project is the controllability between the input terminals and the output sound.

6.0 Conclusion

The project of “Music from bio-signals-software design for music synthesis” forces on the programming work for sound processing and music synthesis from both the static and real-time input signals using MATLAB and other related software and tools in the computer.

The project design is divided into five components: the sound from static data, the music synthesis from static sound, streaming real-time sound, music composition from real-time sound and wireless transmission test. Firstly, some simple notes are made from a piece of raw digital EMG signals in MATLAB. Then those notes are used to simulate the sound of musical instruments with the definition of ADSR envelope and harmonics. Secondly, the DSP toolbox in MATLAB is introduced to stream out the real-time sound manipulated by the mouse pointer in computer screen. Since the composition method has been illustrated during the synthesis of static data, that method is directed applied on the DSP toolbox to produce the sound of musical instruments. Thirdly, multiple controls are achieved by adding more sounds and rhythms in the existing codes. Then a beautiful and harmonious music with multiple controls and real-time inputs is completed. At last, some tests about the connection of hardware and software are carried on.

Currently, this project could generated four different types of sound simultaneously including two different musical instruments with flute and piano controlled by the coordinates of mouse pointer in the computer screen in real-time. This manipulation could be used for four input signals derived from four sensors in human body for further testing. The connection between software and hardware has been proved to be feasible while the data in the hardware terminals could be transmitted to MATLAB without dropout in the test.

In the future work, more tests about the reduction of latency and effective usage of the input data in the programming codes could be proceeded by other researches to achieve a better performance of the real-time control of music from bio-signals.

7.0 Recommendations

7.1 Software installation

To streaming the real-time data, the version of MATLAB should be at least MATAB 2012a or higher with the subject of DSP toolbox. The ASIO driver is also necessary to be installed for streaming process preliminarily. In addition, the Arduino IDE is used for the compiling and burning of Arduino codes.

7.2 Maneuverability and conciseness of programming

The maneuverability of programming codes is also important in the design process. All the parameters should be defined at the beginning of program because one parameter might be called many times in the following codes. With the pre-definition, the programmers don't need to waste time changing same parameter for too many times. In addition, some loop programs and calling programs are necessary to simplify the codes and provide a clear logic of the overall program.

8.0 Acknowledgment

I would like to express my heartfelt gratitude to those who helped me during the writing of my thesis. I gratefully acknowledge the help of my supervisor Kenneth Pope for his constant guidance and suggestions through the whole design process. Without his illuminating instruction and patient teaching, the completion of this thesis would not be possible. Also, I would like to thank Chris Hatswell, who kindly be patient to cooperate with me during the test of software. Last but not least, I am indebted to my parents for their continuous support and encouragement.

9.0 References

Arslan, B., Brouse, A., Castet, J., Filatriau, J.J., Léhembre, R., Noirhomme, Q. & Simon, C. 2005. 'From biological signals to music'. In *2nd International Conference on Enactive Interfaces*, Genoa, Italy, November 17th-18th, 2005.

Arslan, B., Brouse, A., Castet, J., Léhembre, R., Simon, C., Filatriau, J.J. & Noirhomme, Q. 2006. 'A real time music synthesis environment driven with biological signals', In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, Vol. 2, pp. II-II, IEEE.

Bain, R. 2003. 'The Harmonic Series-A path to understanding musical intervals, scales, tuning and timbre', University of South Carolina, Columbia, USA.

Brown, Z. & Gupta, B. 2008. 'Biological Signals and their Measurement', *Update in Anaesthesia*, pp 164-69.

Cook, P.R. & Scavone, G. 1999. 'The synthesis toolkit (stk)', In *Proceedings of the International Computer Music Conference*, October, 1999, pp 164-66.

Cram, J.R. & Kasman, G.S. 1998. The basics of surface electromyography. *Introduction to surface electromyography. USA: Aspen Publishers, Inc*, pp 1-8.

De Leon, P.L. 2000. 'Computer Music in Undergraduate Digital Signal Processing', *American Society for Engineering Education/Gulf Southwestern Region (Las Cruces, NM.)*.

De Luca, C.J., Adam, A., Wotiz, R., Gilmore, L.D. & Nawab, S.H. 2006. 'Decomposition of surface EMG signals', *Journal of neurophysiology*, 96(3), pp 1646-657.

DiCanio, C.T. 2015. 'Introduction to acoustic phonetics', University at Buffalo, New York, USA, 8 October.

Farnell, Andy. 2006. 'Practical synthetic sound design for film, games and interactive media using dataflow', *Designing Sound*, Applied Scientific Press, London, England,

Finau, S. 2014. 'Music from Bio-signals, making music from EMG signals', Flinders University, 28th November, 2014.

Hartling, L., Newton, A.S., Liang, Y., Jou, H., Hewson, K., Klassen, T.P. & Curtis, S. 2013. 'Music to reduce pain and distress in the pediatric emergency department: a randomized clinical trial', *JAMA pediatrics*, 167(9), pp.826-835.

Houcque, D. 2005. *Introduction to Matlab for engineering students*, version 1.2, Northwestern University, August 2005.

Kabal, P. 2004. 'Discrete Time Signal Processing', *Department of Electrical and Computer Engineering*, McGill University, Montreal , Canada, pp 5-12.

Knapp, R.B. & Lusted, H.S. 1990. 'A bioelectric controller for computer music applications', *Computer music journal*, 14(1), pp 42-7.

Mathew, T., Abraham, B.M. & Scaria, R., 'Music Synthesis using Sinusoid Generator, ADSR Envelope Generator and Composer Code', *International Journal of Scientific Engineering and Research (IJSER)*, ISSN: 2347-3878, Volume 3, Issue 2, 2015, pp 23-5.

Mathworks 2016, *dsp.AudioPlayer System object*, United States, viewed 27 October 2016, <<http://au.mathworks.com/help/dsp/ref/dsp.audioplayer-class.html>>.

Meister, J. & Errede, S.M. 2011. 'MIDI: A History and Technical Overview', *Physics of Music*, University of Illinois at Urbana-Champaign.

Muthuswamy, J., 2004. 'Biomedical Signal Analysis'. *Standard Handbook Of Biomedical Engineering And Design*, 14, pp.18-1.

Norali, A.N., Som, M.M. and Kangar-arau, J., 2009. 'Surface electromyography signal processing and application: A review', *In Proceedings of the International Conference on Man-Machine Systems (ICoMMS)*, pp 11-3.

Novotney, A., 2013. 'Music as medicine'. *American Psychological Association*, pp.10-46.

Olvera, F.E. 2006. 'Electrocardiogram waveform feature extraction using the matched filter', *Statistical Signal Processing II*, ECE SIO Stat Proc.

Petersen, M., 2001. 'Mathematical Harmonies' Mathematical Association of America College, *Mathematics Journal*, Vol. 35, No. 5, November 2004, pp 396-401.

Rose, N. 2013. 'Music from Bio-signals, Hardware Design of a Data Collection Device', Flinders University, 15th November, 2013.

Schmidt-Jones, C. & Jones, R. 2011. *Understanding basic music theory*. Connexions, Rice University, Houston, Texas.

Teplan, M. 2002. 'Fundamental of EEG Measurement', *Measurement Science Review*, Volume 2, Section 2, pp 1-11.

Wang, G. 2008. *The chuck audio programming language. A strongly-timed and on-the-fly environ/mentality*, Princeton University, Princeton, New Jersey, United States.

Whiting, C. 2014. 'Added necessary files for the music with matlab program'. Web log post, *Github*, 28 Aug, 2014, viewed 27 October 2016, retrieved from <https://github.com/mpro34/matlab/blob/master/matlab_music/Envelope.m>

Appendix

1. Codes for creating notes from static data (undigested draft)

```
clear all;
emg = load('EMG.txt');
left=emg(:,1);
right=emg(:,2);
X1=right(1:2500 ,1);
X2=right(2501:5000 ,1);
X3=right(5001:7500 ,1);
X4=right(7501:10000 ,1);
X5=right(10001:12500 ,1);
X6=right(12501:15000 ,1);
X7=right(15001:17500 ,1);
X8=right(17501:20000 ,1);
X9=right(20001:22500 ,1);
X10=right(22501:25000 ,1);
X11=right(25001:27500 ,1);
X12=right(27501:30000 ,1);
X13=right(30001:32500 ,1);
X14=right(32501:35000 ,1);
X15=right(35001:37500 ,1);
X16=right(35001:37500 ,1);
MS1= sum(X1.^2)/length(X1);
MS2= sum(X2.^2)/length(X2);
MS3= sum(X3.^2)/length(X3);
MS4= sum(X4.^2)/length(X4);
MS5= sum(X5.^2)/length(X5);
MS6= sum(X6.^2)/length(X6);
MS7= sum(X7.^2)/length(X7);
MS8= sum(X8.^2)/length(X8);
MS9= sum(X9.^2)/length(X9);
MS10= sum(X10.^2)/length(X10);
MS11= sum(X11.^2)/length(X11);
MS12= sum(X12.^2)/length(X12);
MS13= sum(X13.^2)/length(X13);
MS14= sum(X14.^2)/length(X14);
MS15= sum(X15.^2)/length(X15);
MS16= sum(X16.^2)/length(X16);
ff1=MS1*7500;
ff2=MS2*7500;
ff3=MS3*7500;
ff4=MS4*7500;
```

```

ff5=MS5*7500;
ff6=MS6*7500;
ff7=MS7*7500;
ff8=MS8*7500;
ff9=MS9*7500;
ff10=MS10*7500;
ff11=MS11*7500;
ff12=MS12*7500;
ff13=MS13*7500;
ff14=MS14*7500;
ff15=MS15*7500;
ff16=MS16*7500;
f1=ff1*10;
f2=ff2*10;
f3=ff3*10;
f4=ff4*10;
f5=ff5*10;
f6=ff6*10;
f7=ff7*10;
f8=ff8*10;
f9=ff9*10;
f10=ff10*10;
f11=ff11*10;
f12=ff12*10;
f13=ff13*10;
f14=ff14*10;
f15=ff15*10;
f16=ff16*10;
F= [f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11 f12 f13 f14 f15 f16];
N=16;
for i=1:N
    if          65<=F(1,i) & F(1,i)<130
                F(1,i)=4*F(1,i)
    elseif     130<=F(1,i) & F(1,i)<262
                F(1,i)=2*F(1,i)
    elseif     523<=F(1,i) & F(1,i)<1047
                F(1,i)=F(1,i)/2
    elseif     1047<=F(1,i) & F(1,i)<2097
                F(1,i)=F(1,i)/4
    end;
end
for i=1:N
    if          262<=F(1,i) & F(1,i)<294
                F(1,i)= 262
    end;
end

```

```

elseif 294<=F(1,i) & F(1,i)<329
    F(1,i)= 294
elseif 329<=F(1,i) & F(1,i)<349
    F(1,i)= 329
elseif 349<=F(1,i) & F(1,i)<392
    F(1,i)= 349
elseif 392<=F(1,i) & F(1,i)<440
    F(1,i)= 392
elseif 440<=F(1,i) & F(1,i)<493
    F(1,i)= 440
elseif 493<=F(1,i) & F(1,i)<523
    F(1,i)= 493
end;
end
f1= F(1,1);
f2= F(1,2);
f3= F(1,3);
f4= F(1,4);
f5= F(1,5);
f6= F(1,6);
f7= F(1,7);
f8= F(1,8);
f9= F(1,9);
f10= F(1,10);
f11= F(1,11);
f12= F(1,12);
f13= F(1,13);
f14= F(1,14);
f15= F(1,15);
f16= F(1,16);
fs=4500;
duration_1=1-1/fs;
n0= [0:fs*duration_1/2];
duration_1=1-1/fs;
n1 = [0:fs*duration_1];
duration_2=duration_1*2;
n2 = [0:fs*duration_2];
x1 = sin(2*pi*n2*f1/fs);
x2 = sin(2*pi*n0*f2/fs);
x3 = sin(2*pi*n1*f3/fs);
x4 = sin(2*pi*n2*f4/fs);
x5 = sin(2*pi*n0*f5/fs);
x6 = sin(2*pi*n1*f6/fs);
x7 = sin(2*pi*n1*f7/fs);

```

```

x8 = sin(2*pi*n1*f8/fs);
x9= sin(2*pi*n1*f9/fs);
x10 = sin(2*pi*n1*f10/fs);
x11 = sin(2*pi*n2*f11/fs);
x12 = sin(2*pi*n1*f12/fs);
x13 = sin(2*pi*n2*f13/fs);
x14= sin(2*pi*n0*f14/fs);
x15 = sin(2*pi*n1*f15/fs);
x16 = sin(2*pi*n1*f16/fs);
p0=sin(2*pi*n0*0/fs);
p1=sin(2*pi*n1*0/fs);
music=[x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,p0,x13,x14,x15,x16];
sound(music);

```

2. Codes for composition from static sound

```

%% definitions
file_in = 'EMG.txt';
Nblock = 2500;
Nnotes = 16;
ms2n = 340;
major_scale = 2 .^ ( [ 0 2 4 5 7 9 11] / 12);
f0 = 523;
harmonics = [ 0.467 0.3 0.1 0.1 0.003];
% fs = 44100;
fs = 8000;
%set the length of notes
beat_length = 0.25;
note_length = beat_length * 2 .^[ 0 0 1 2 1 1 0 0 2 1 0 0 1 1 1 2];
%% load the data
emg = load( file_in);
left=emg(:,1);
right=emg(:,2);
%% measure the input data
sright = right( 1:Nblock * Nnotes) .^ 2;
ms = mean( reshape( sright, Nblock, []), 1);
%% create the notes
notes = ceil( ms * ms2n);
freq = f0 * major_scale( notes)
music = [];
for i = 1:Nnotes
    note = make_note( freq( i), harmonics, note_length( i), fs);
    music = [ music, note];
end

```

```
%% play the music
sound( music, fs)
```

3. Codes for the function of ‘*make_note*’ in static composition

```
function note = make_note( freq, harmonics, note_length, fs)
% derived
N = note_length * fs;
% make the basic note
t = ( 1:N) / fs;
note = zeros( 1, N);
for i = 1:numel( harmonics)
    note = note + harmonics(i) * sin( 2 * pi * freq * i * t);
end
% impose envelope
A = linspace( 0,    0.7,    N * 0.1); %rise 10% of signal
D = linspace( 0.7,  0.5,    N * 0.2); %drop of 20% of signal
S = linspace( 0.5,  0.3,    N * 0.4); %delay of 40% of signal
R = linspace( 0.3,  0,      N * 0.3); %drop of 30% of signal
note = note .* [ A D S R];
```

4. Codes for streaming real-time sound from sine-wave

```
%% Definition
Fs=48000;
f0 = 300;
duration = 50;
samplesPerFrame = 128;           %Block_Size = samplesPerFrame/Fs =
2.7ms
frame_size = 0.125;             %queue_duration = 125ms
Buffer_Size = 512;              %Buffer = 11ms
%% Create Audio File
% t = linspace(0, duration, duration*Fs);
t = (1:samplesPerFrame) / Fs;
% data = sin(2 * pi * f0 * t);
% audiowrite('Test.wav',[t,data],Fs);
%% Create Audio File Read System Objects
%% Create File Player System Objects
AP = dsp.AudioPlayer('SampleRate',Fs, ...
    'BufferSizeSource','Property',...
    'BufferSize',Buffer_Size,...
    'QueueDuration',frame_size, ...
    'OutputNumUnderrunSamples',true);
%% Loop for Playing Input Audio
```

```

% while ~isDone(AFR)
%   audio = step(AFR);
scrsz = get(0,'ScreenSize');
pl = 1000;
theta = rand * 2 * pi;
while pl > 10
    pl = get( 0, 'PointerLocation');
    %   audio = audio * pl( 1) / scrsz( 3);
    freq = f0 * ( 1 + pl( 2) /scrsz( 4));
    audio = sin(2 * pi * freq * t + theta) * pl( 1) / scrsz( 3);
    theta = rem( theta + 2 * pi * freq * samplesPerFrame / Fs, 2 * pi);
    nUnderrun = step(AP,[ audio; audio]);
    if nUnderrun > 0
        fprintf('Audio player queue underrun by %d samples.\n'...
            ,nUnderrun);
    end
end
pause(AP.QueueDuration); % wait until audio is played to the
end
% release(AFR); % close the input file
release(AP); % close the audio output device
%%
%scrsz = get(0,'ScreenSize');
%pl = get( 0, 'PointerLocation');
%audio = audio * pl( 1) / scrsz( 3);

```

5. Codes for streaming real-time music with multiple controls

```

%% Definition
Fs = 48000;
f0 = 130.81;
f0_2 = 65.4;
f0_3 = 65.4;
f0_4 = 261.63;
samplesPerFrame = 0.5 * Fs;
samplesPerFrame_2 = samplesPerFrame / 2;
samplesPerFrame_3 = samplesPerFrame / 2 ;
samplesPerFrame_4 = samplesPerFrame ;
queue_duration = 0.002;
Buffer_Size = 640;
harmonics = [0.38 0.2 0 0.18 0 0.22 0 0.22];
harmonics_2 = [0.1842 0.1053 0 0.2256 0 0.0827 0 0.0300];
harmonics_3 = [0.2078 0.1333 0 0.1255 0 0.1804 0 0.0157];
harmonics_4= [0.467 0.3 0.1 0.1 0.003];

```

```

beat_length = 0.5;
beat_length_2 = beat_length;
note_length = beat_length * 2.^[2 1 1 0 0 1];
note_length_2 = beat_length * 2.^[2 2 1 1 1 1 0 0 0 0 1 1];
note_length_3 = beat_length * 2.^[2 2 1 1 1 1 0 0 0 0 1 1];
note_length_4 = beat_length * 2.^[2 1 1 0 0 1];
major_scale = 2.^([0 2 4 7 9 12 14 16 19 21] / 12);
major_scale_2 = 2.^([0 2 4 7 9 12 14 16 19 21] / 12);
major_scale_3 = 2.^([0 2 4 7 9] / 12);
major_scale_4 = 2.^([0 2 4 7 9] / 12);
Nnotes = length ( note_length);
Nscale = length ( major_scale);
Nscale_2 = length ( major_scale_2);
Nscale_3 = length ( major_scale_3);
Nscale_4 = length ( major_scale_4);
Blance_1 = 0.4; %0.2;0.4
Blance_2 = 1.6; %0.8;1.6
Blance_3 = 2.0; %1.4;2.0
Blance_4 = 0.4; %0.2;0.4
%% Create File Player System Objects
AP = dsp.AudioPlayer('SampleRate',Fs, ...
    'BufferSizeSource','Property',...
    'BufferSize',Buffer_Size,...
    'QueueDuration',queue_duration, ...
    'OutputNumUnderrunSamples',true);
%% Loop for Playing Input Audio
pl = 1000;
scrsz = get(0,'ScreenSize');
ind = 0;
r = 1:( queue_duration * Fs);
while pl > 10
    ind = rem( ind, Nnotes) + 1;
    a = 2 * ind - 1;
    b = 2 * ind;
    %pause(0.95 * AP.QueueDuration); % wait until audio is nearly all played
    %AP.QueueDuration
    %% first_audio
    pl = get( 0, 'PointerLocation');
    notes = ceil( Nscale * pl( 2) /scrsz( 4));
    freq = f0 * major_scale( notes);
    note = frame_note( freq, harmonics, note_length( ind), Fs, samplesPerFrame);
    audio = Blance_1 * note * pl( 1) / scrsz( 3);
    %% second_audio
    notes_2 = ceil( Nscale_2 * pl( 1) / scrsz( 3));

```

```

    freq_2 = f0_2 * major_scale_2( notes_2);
    note_2_a = frame_note_second_audio( freq_2, harmonics_2, note_length_2( a),
Fs, samplesPerFrame_2);
    note_2_b = frame_note_second_audio( freq_2, harmonics_2,
note_length_2( b), Fs, samplesPerFrame_2);
    note_2 = [note_2_a note_2_b];
    audio_2 = Blance_2 * note_2 * pl( 2) /scrsz( 4);
    %% third_audio
    notes_3 = ceil( Nscales_3 * pl( 1) / scrsz( 3));
    freq_3 = f0_3 * major_scale_3( notes_3);
    %note_3 = frame_note_second_audio_A2( freq_3, harmonics_3,
note_length_3( ind), Fs, samplesPerFrame);
    note_3_a = frame_note_second_audio( freq_3, harmonics_3, note_length_3( a),
Fs, samplesPerFrame_3);
    note_3_b = frame_note_second_audio( freq_3, harmonics_3,
note_length_3( b), Fs, samplesPerFrame_3);
    note_3 = [note_3_a note_3_b];
    audio_3 = Blance_3 * note_3 * pl( 2) /scrsz( 4);
    %% forth_audio
    notes_4 = ceil( Nscales_4 * pl( 2) / scrsz( 4));
    freq_4 = f0_4 * major_scale_4( notes_4);
    %note_4_a = frame_note_4( freq_4, harmonics_4, note_length_4( a), Fs,
samplesPerFrame_4);
    %note_4_b = frame_note_4( freq_4, harmonics_4, note_length_4( b), Fs,
samplesPerFrame_4);
    %note_4 = [note_4_a note_4_b];
    note_4 = frame_note_4( freq_4, harmonics_4, note_length_4( ind), Fs,
samplesPerFrame_4);
    audio_4 = Blance_4 * note_4 * pl( 1) /scrsz( 3);
    %% loop_for_one_frame
    while size( audio, 2) > 0
        frame = [ audio( r) + audio_2( r) + audio_3( r) + audio_4( r);audio( r) +
audio_2( r) + audio_3( r) + audio_4( r) ];
        nUnderrun = step(AP, frame);
        audio( r)=[];
        audio_2( r)=[];
        audio_3( r)=[];
        audio_4( r)=[];
        if nUnderrun > 0
            fprintf('Audio player queue underrun by %d samples.\n'...
,nUnderrun);
        end
    end
end
end
end

```



```

pause(AP.QueueDuration);           % wait until audio is played to the
end
release(AP);                       % close the audio output device

```

6. Codes for the function of *'frame_note'* in multiple controls (first flute)

```

function note = frame_note( freq, harmonics, note_length, Fs, samplesPerFrame)
% derived
N = note_length * samplesPerFrame;
% make the basic note
t = ( 1:N) / Fs;
note = zeros( 1, N);
for i = 1:numel( harmonics)
    note = note + harmonics( i) * sin( 2 * pi * freq * i * t);
end
% impose envelope
A = linspace( 0,    0.4,    N * 0.2); %rise 10% of signal
D = linspace( 0.4,  1.0,    N * 0.1); %drop of 20% of signal
S = linspace( 1.0,  0.9,    N * 0.4); %delay of 40% of signal
R = linspace( 0.9,  0.0,    N * 0.3); %drop of 30% of signal
note = note .* [ A D S R];

```

7. Codes for the function of *'frame_note_4'* in multiple controls (second flute)

```

function note = frame_note_4( freq_4, harmonics_4, note_length_4, Fs,
samplesPerFrame_4)
% derived
N = note_length_4 * samplesPerFrame_4;
% make the basic note
t = ( 1:N) / Fs;
note = zeros( 1, N);
for i = 1:numel( harmonics_4)
    note = note + harmonics_4( i) * sin( 2 * pi * freq_4 * i * t);
end
% impose envelope
A = linspace( 0,    0.4,    N * 0.2); %rise 10% of signal
D = linspace( 0.4,  1.0,    N * 0.2); %drop of 20% of signal
S = linspace( 1.0,  1.0,    N * 0.4); %delay of 40% of signal
R = linspace( 1.0,  0.0,    N * 0.2); %drop of 30% of signal
note = note .* [ A D S R];

```

8. Codes for the function of *'frame_note_second_audio'* in multiple

controls (piano)

```
function note = frame_note_second_audio( freq_2, harmonics_2, note_length_2,
Fs, samplesPerFrame_2)
% derived
N = note_length_2 * samplesPerFrame_2;
% make the basic note
t = ( 1:N) / Fs;
note = zeros( 1, N);
for i = 1:numel( harmonics_2)
    note = note + harmonics_2(i) * sin( 2 * pi * freq_2 * i * t);
end
% impose envelope
A = linspace( 0,      0.8,      N * 0.02); %rise 10% of signal
D = linspace( 0.8,    0.6,      N * 0.04); %drop of 20% of signal
A_1 = linspace( 0.6,  0.9,      N * 0.02); %rise 10% of signal
D_1 = linspace( 0.9,  0.7,      N * 0.04); %drop of 20% of signal
A_2 = linspace( 0.7,  0.9,      N * 0.02); %rise 10% of signal
D_2 = linspace( 0.9,  0.6,      N * 0.04); %drop of 20% of signal
A_3 = linspace( 0.6,  0.7,      N * 0.02); %rise 10% of signal
D_3 = linspace( 0.7,  0.6,      N * 0.04); %drop of 20% of signal
S = linspace( 0.6,    0.2,      N * 0.66); %delay of 40% of signal
R = linspace(0.2,     0.0,      N * 0.1); %drop of 30% of signal
note = note .* [ A D A_1 D_1 A_2 D_2 A_3 D_3 S R];
```