第七屆旺宏科學獎

成果報告書

參賽編號：SA7-025

作品名稱：軟體調音器

姓　　名：林冠汝

關 鍵 字： 聲音頻率偵測、自相關函數、平均振幅差函數

研究題目

軟體調音器

研究摘要

本篇研究分析了現有音高偵測技術中的自相關函數（ACF）、平均振幅差函數（AMDF）法，加以組合而發展了一個更為精準的音高偵測函數，為了達到即時顯示目前聲音的音高供使用者方便於吉他的調音，在實作時採用採用二個緩衝區的機制及 Wiener-Khinchin 定理（使用了快速傅立葉轉換與反快速傅立葉轉換）來大幅提升程式的運算速度，從實作與測試中得知，本研究的方法可以在很短的時間內（0.13 秒至 0.15 秒）完成每個緩衝區中音框的音高辨識，以即時地方式呈現目前聲音的音高，且其在音高偵測上的平均誤差僅有 0.39%。

研究動機

吉他在彈奏前都需經過調音這個步驟，對於初學者來說，就算藉助了調音器，仍然很難憑著音感調出準確的音高，若是能夠有一個視覺化的調音器，可以直接看出音調準了沒，相信調音這個工作一定變的簡單又愉快。電腦已是日常生活中的必備物品，若能利用電腦程式設計出視覺化的調音器，這樣就不用花錢去購買調音器，而且在保存與取得上會方便許多。

研究目的

希望藉由本研究設計出一個能夠準確地分析吉他聲音頻率的程式，並能夠即時地以視覺化的介面來呈現，方便使用者能夠輕易地上手做吉他調音的工作。

研究過程

1. 器材及軟體

   個人電腦、筆記型電腦、Visual Basic 6、Speech Analyzer 3.01、Praat 5.0.09、Solo Explorer 1.0、SFS 4.7、古典吉他。

2. 音高簡介

   樂音三要素中的音高（Pitch）就是指人類心理對於聲音基本頻率之感受，基本頻率越高音高也就越高，在樂理上，每個全音階依照十二平均率分為 12 個半音，每個半音均有其對應的基本頻率，可用下列的式子來表示兩者間的轉換關係[1][2]：

   $$半音 = 69 + 12 \times \log_2(基本頻率 / 440)$$

例如基本頻率是 440Hz 時，其對應到的半音是 69，基本頻率為 220Hz 時，其對應到的半音是 57，一般來說，利用數字來表示音高不太容易明瞭與記憶，通常都以科學音高記號法來替代，表格 1 是吉他所能彈奏音域範圍的音高頻率表，頻率的單位為赫茲 Hz，括號內的數字為半音。

音名 / 全音階	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
2					第 6 弦 82.4(40)	87.3(41)	92.5(42)	98.0(43)	103.8(44)	第 5 弦 110.0(45)	116.5(46)	123.5(47)
3	130.8(48)	138.6(49)	第 4 弦 146.8(50)	155.6(51)	164.8(52)	174.6(53)	185.0(54)	第 3 弦 196.0(55)	207.7(56)	220.0(57)	233.1(58)	第 2 弦 246.9(59)
4	261.6(60)	277.2(61)	293.7(62)	311.1(63)	第 1 弦 329.6(64)	349.2(65)	370.0(66)	392.0(67)	415.3(68)	440.0(69)	466.2(70)	493.9(71)
5	523.3(72)	554.4(73)	587.3(74)	622.3(75)								

表格 1 吉他所能彈奏音域範圍的音高頻率表

3. 現有的音高偵測方法

音高偵測的方法與技術主要分為時域(Time Domain)及頻域(Frequency Domain)兩大類型[1][2][4]。在時域上運算的方法有零交叉、自相關函數（AutoCorrelation Function）、平均振幅差函數（Average Magnitude Difference Function）等，此類型的方法是直接分析聲音波形資料而得到聲音的頻率，所以運算起來較為簡單，在單一頻率的單音（Monophony）辨識上，辨識率還蠻高的，不過若是用來辨識兩個頻率以上所組成的複音（Polyphony），其辨識能力則會大打折扣。

在頻域運算的方法有調和產品頻譜（HPS, Harmonic Product Spectrum）、倒頻譜（Cepstrum）、最大化相似度（ML, Maximum Likelihood）等，這類型的方法先利用傅立葉轉換（Fourier Transform）將聲音波形資料轉換至頻域，然後藉由分析頻域資料而得到聲音的頻率，由於是得到聲音的整個頻譜，所以可用來辨識複音，但是運算量卻也多了許多。

4. 相關軟體介紹

可用來偵測音高的軟體相當多，比較常用的軟體有 Speech Filing System（SFS）[5]、Praat[6]、Solo Explorer[7]、Speech Analyzer[8]，不過這些軟體僅能偵測與分析已經存在的音訊檔，無法以即時（Real Time）的形式來顯示目前聲音的音高。

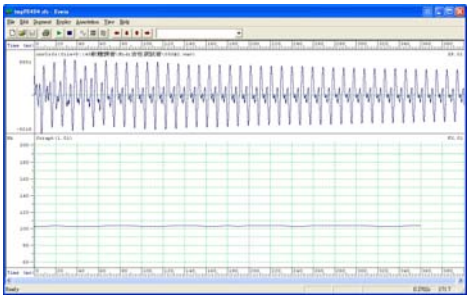

圖 1-1 Speech Filing System

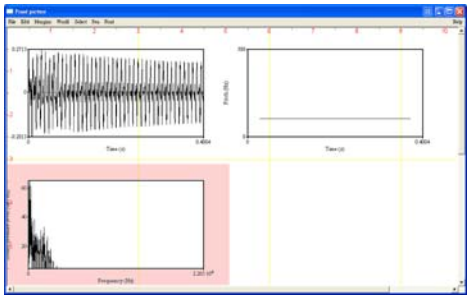

圖 1-2 Praat

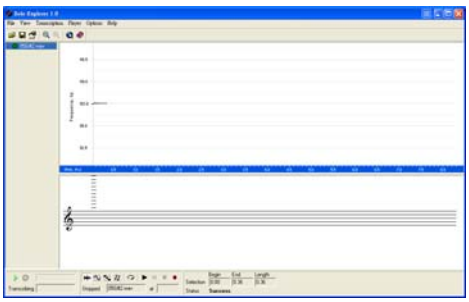

圖 1-3 Solo Explorer

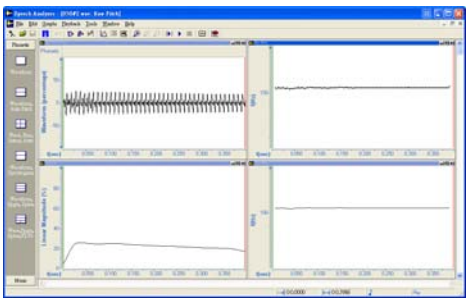

圖 1-4 Speech Analyzer

5. 軟體調音器的要求

由於電腦的日益普及，軟體式的調音器無論在取得與保存上，應該都會比硬體式的調音器方便許多，一個好的軟體調音器在設計上必需滿足下列的要求：反應時間要短，能夠即時的顯示出目前聲音的音高；能夠精準的偵測出目前聲音的音高；能將周圍雜音的干擾降至最低；親和性佳的使用者介面，讓初學者也能輕易上手。

為了達到上述的要求，所設計的軟體調音器主要可分為聲音接收與音高偵測等兩大部分，除了可以連續不斷的接收外界的聲音外，同時也要能夠快速且精準的偵測出目前聲音的音高，並在使用者介面上顯示出來，另外，在處理的過程中也必須考慮到外界雜音的問題，務使干擾降到最低。

6. 聲音接收部分的設計

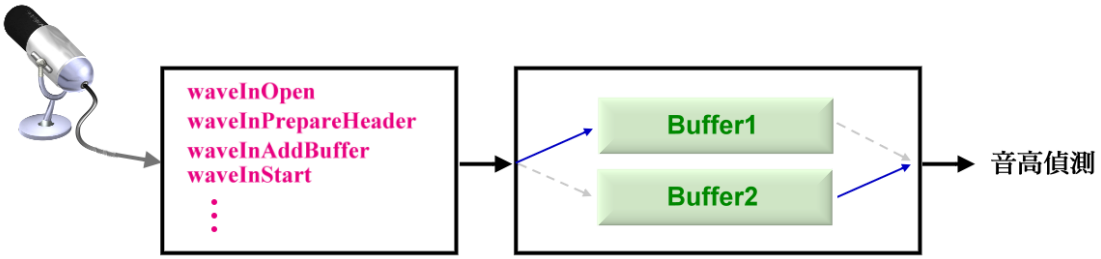

圖 2 聲音接收部分設計示意圖

為了能夠連續不斷的接收外界的聲音，同時也要能夠快速的偵測出聲音音高，可以採用 2 個緩衝區的機制，麥克風陸續接收進來的聲音先存放在第 1 個緩衝區（Buffer1），當存滿後，換到第 2 個緩衝區（Buffer2）存放，並將第 1 個緩衝區的資料進行音高偵測，完成後將其資源回收留待第 2 個緩衝區填滿後使用，如此交替循環直到程式終止。要達成這樣的機制必須使用 Windows 最底層的 API 函式，包括管理緩衝區的 GlobalAlloc、GlobalLock、GlobalFree（使用 kernel32.dll 動態連接函式庫），以及接收聲音的 waveInOpen、waveInPrepareHeader、waveInAddBuffer、waveInStart、waveInStop、waveInUnprepareHeader、waveInClose（使用 wimm.dll 動態連接函式庫），整個設計示意圖如圖 2 所示。

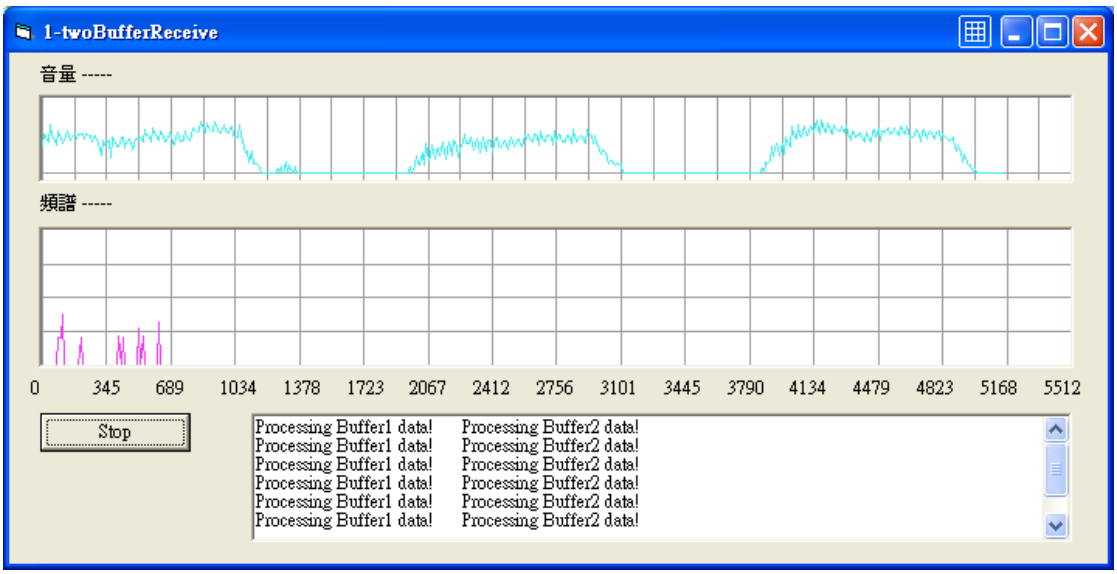

圖 3 以 VB 實作兩個具有兩個緩衝區機制的聲音接收程式

具有兩個緩衝區機制的聲音接收程式使用 Visual Basic 6 來實作，相關設定為：取樣頻率 44100Hz、單聲道、每個取樣點為 2bytes、每個緩衝區可放置 4096 個取樣點，執行時的畫面如圖 3 所示，除了可以即時顯示目前聲音的音量與頻譜外，另外加了個 TextBox 控制項，可用來追蹤緩衝區的使用情形。

7. 音高偵測函數的測試

音高偵測的方法與技術主要分為時域及頻域兩種類型，每一種方法都有其存在的價值與優缺點，調音軟體主要是用來辨識單一頻率的單音，所以打算採用時域類型的偵測方法，現將兩種常用的時域類方法自相關函數 ACF 與平均振幅差函數 AMDF 介紹如下[1][4]：

- 自相關函數（ACF, AutoCorrelation Function）

  $ACF(k)=\sum_{n=1}^{N-k}x(n)x(n+k)$

  把音訊 $x(n)$ 平移一段取樣點 k，再和未平移前的音訊做內積，即可得到 1 個值 $ACF(k)$，上列式子中 N 表示音訊中取樣點總數；$k=0\sim n-1$。兩個向量做內積時，若是這兩個向量越相像，得到的內積值就越大，所以找出 $ACF(k)$ 極大值的位置，就可以得到音訊的週期，進而求出其頻率與音高。

- 平均振幅差函數（AMDF, Average Magnitude Difference Function）

  $AMDF(k)=\sum_{n=1}^{N-k}\left|x(n)-x(n+k)\right|$

  與自相關函數類似，只是將 $ACF(k)$ 算式中的乘法改成減法後取絕對值而得到 $AMDF(k)$，若其值越小，表示兩個向量間的差異也越小，所以找出 $AMDF(k)$ 極小值的位置，就可以得到音訊的週期，進而求出其頻率與音高。

觀察自相關函數與平均振幅差函數，發現他們在運算上極為相似，是否會有其他更好的函數呢？所以除了上述兩種方法外，同時也實作了下列六個函數以及頻域的快速傅立葉轉換法（FFT），並用準備好的 56 個音訊檔來測試，以便瞭解這些方法的效能與精確度。

- $Function3(k)=\sum_{n=1}^{N-k}(x(n)-x(n+k))^2$
- $Function4(k)=\sum_{n=1}^{N-k}\left|x(n)+x(n+k)\right|$

- $Function5(k) = \sum_{n=1}^{N-k} (x(n) + x(n+k))^2$
- $Function6(k) = \sum_{n=1}^{N-k} x(n)x(n+k) \Big/ \sum_{n=1}^{N-k} |x(n) - x(n+k)|$
- $Function7(k) = \sum_{n=1}^{N-k} x(n)x(n+k)(x(n) + x(n+k))$
- $Function8(k) = \sum_{n=1}^{N-k} x(n)x(n+k) \Big/ \sum_{n=1}^{N-k} (x(n) - x(n+k))^2$

測試用的音訊檔分為兩大類，第一類是利用 Guitar Pro 5 軟體所產生模擬吉他音色的 Midi 音訊檔，第二類是真實古典吉他所產生的音訊檔，每一類型各有 28 個不同音高的音訊檔，其取樣頻率為 44100Hz、每個取樣點為 16 位元、單聲道，音高由 E2 至 G4，其對應的吉他弦及把位如圖 4 所示。每個音訊檔分別取出 4096 個取樣點來供音高偵測函數測試，測試結果如表格 2 所示（詳細資料見附錄，圖 5 為測試程式執行畫面），在沒有精密儀器輔助的情況下，其實很難得知每個音訊檔的真實音高，所以採用 Speech Filing System（SFS）、Praat、Solo Explorer、Speech Analyzer 等 4 種知名軟體所測得的平均值當做真實音高。

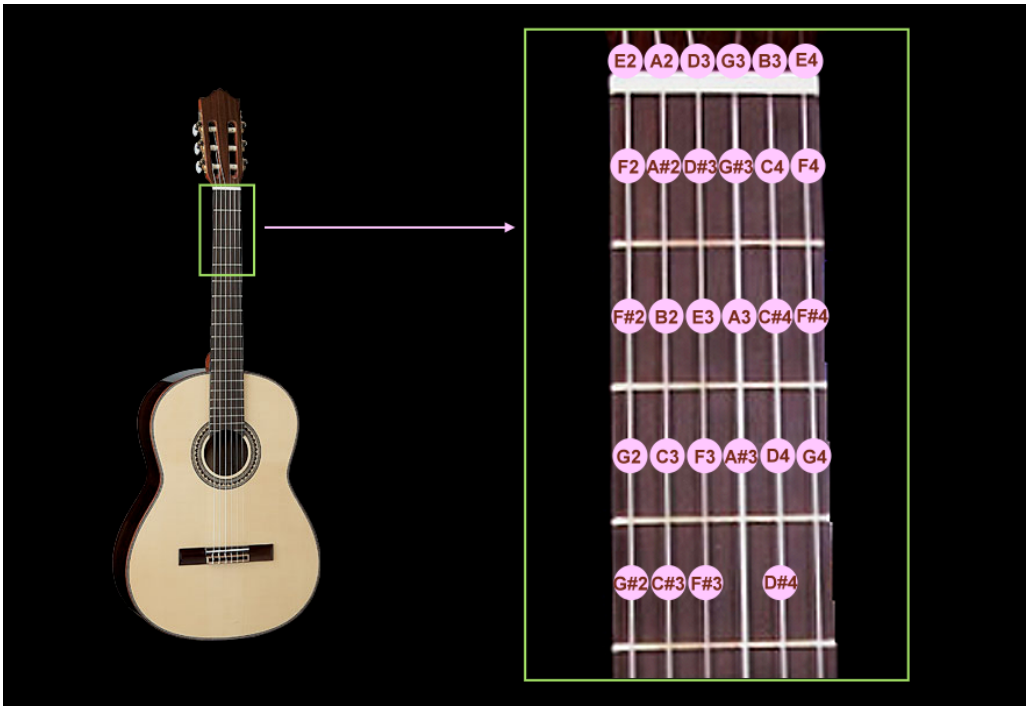

圖 4 測試用音訊檔的音高與其對應的弦及把位

誤差與效能 / 偵測函數	音高偵測平均誤差	平均花費時間
ACF	2.02%	1.772 秒
AMDF	3.06%	1.787 秒
Function3	1.99%	2.201 秒
Function4	37.53%	1.917 秒
Function5	14.78%	2.194 秒
Function6	0.21%	4.538 秒
Function7	2.06%	4.468 秒
Function8	0.20%	8.342 秒
FFT	12.70%	0.048 秒

表格 2 各種音高偵測函數的平均誤差與平均花費時間

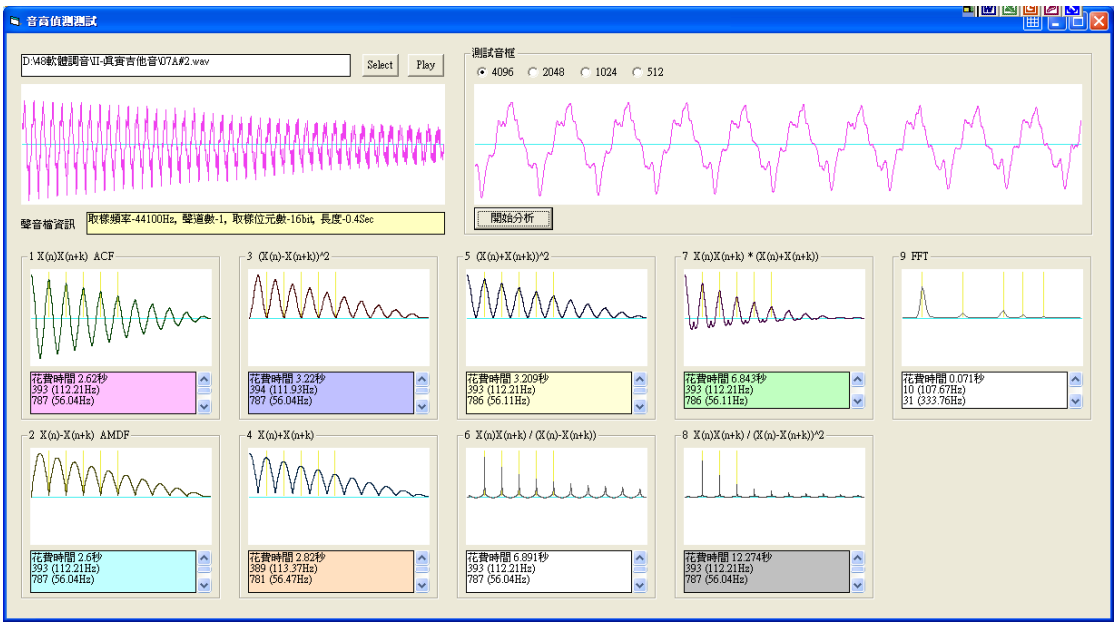

圖 5 各種音高偵測函數測試

8. 偵測函數運算效能的提升

由音高偵測函數的測試結果得知，快速傅立葉轉換法具有最快的運算速度（時間複雜度為 $O(N\log N)$），但是其音高辨識能力確顯不足，而 *Function*8 則有最精確的音高辨識能力，但是運算起來卻是非常地耗費時間（時間複雜度為 $O(N^2)$），若是能加快 *Function*8 的運算速度，那就可以得到一個非常有效能的音高偵測函數。由下列的轉換可以看出，其實 *Function*8 內含有自相關函數 ACF：

$$\begin{aligned} Function8(k) &= \sum_{n=1}^{N-k} x(n)x(n+k) \Big/ \sum_{n=1}^{N-k} (x(n)-x(n+k))^2 \\ &= \sum_{n=1}^{N-k} x(n)x(n+k) \Big/ (\sum_{n=1}^{N-k} (x^2(n)+x^2(n+k)-2x(n)x(n+k)) \\ &= ACF(k) \Big/ (\sum_{n=1}^{N-k} x^2(n) + \sum_{n=1}^{N-k} x^2(n+k) - 2\times ACF(k)) \end{aligned}$$

另外，Wiener-Khinchin 定理得知[9]，自相關函數 ACF 可以利用快速傅立葉轉換 FFT 與反快速傅立葉轉換 IFFT 求得 $ACF(k) = IFFT(|FFT(x(n))|^2)$，時間複雜度為 $O(N\log N)$，而 $\sum_{n=1}^{N-k} x^2(n)$ 與 $\sum_{n=1}^{N-k} x^2(n+k)$ 可用累加的方式求得，時間複雜度為 $O(N)$，所以整個 *Function*8 的時間複雜度可由 $O(N^2)$ 變成 $O(N\log N)$。經過實作的測試，可以看出音高偵測函數 *Function*8 確實在運算速度上增快了許多。

誤差與效能 / 偵測函數	音高偵測平均誤差	平均花費時間
Function8	0.20%	8.342 秒
Enhance Function8	0.39%	0.115 秒

表格 3 *Function8* 加快速度前後的比較

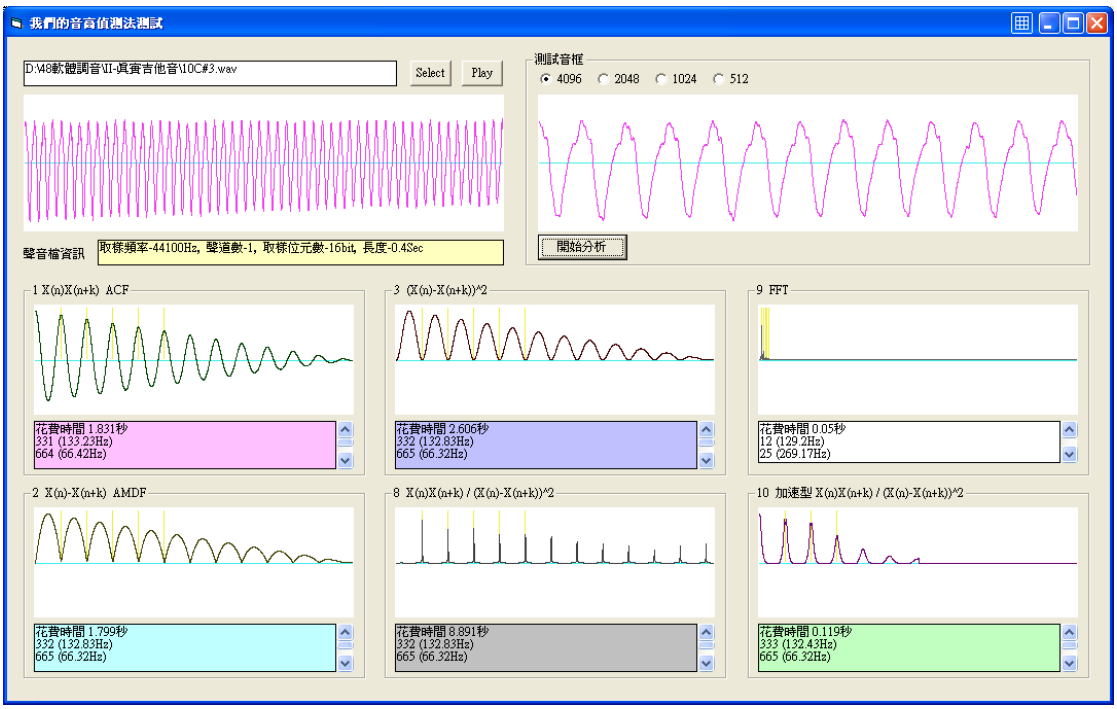

圖 6 音高偵測函數 *Function8* 的加快速度測試

9. 雜訊與靜音的處理

利用麥克風收錄外界聲音時，多少會有雜訊混在聲音訊號中，雜訊可能除了來自於周遭環境外，也可能是麥克風或是音效卡本身所產生的，為了避免雜訊會影響到音高的偵測，最好能夠消除雜訊。利用 GoldWave 軟體來觀察錄製好的音訊檔，發現雜訊多是在中線附近上下 5%的範圍內震盪，所以可以設定好一個門檻 $noise_{TH}$，利用下列算式來消除音訊中的雜訊。

$$x'(n) = x(n) - noise_{TH} \quad ，當\ x(n) > noise_{TH}$$

$$x'(n) = 0 \quad ，當\ |x(n)| \le noise_{TH}$$

$$x'(n) = x(n) + noise_{TH} \quad ，當\ x(n) < -noise_{TH}$$

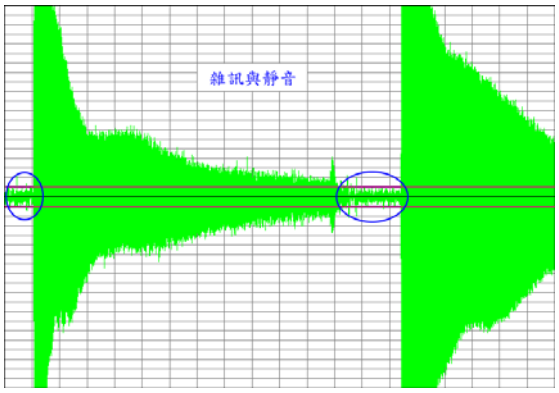

圖 7-1 雜訊在中線附近上下震盪

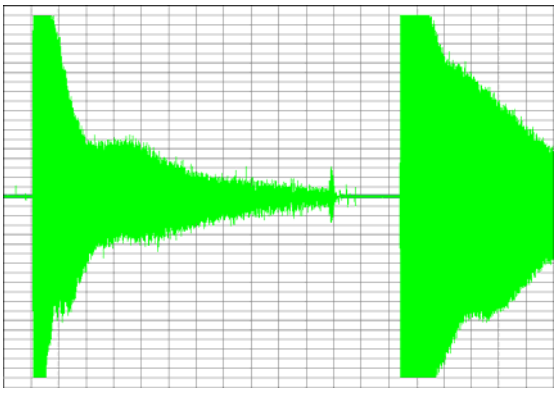

圖 7-2 消除中線附近上下震盪的雜訊

調音器是用來偵測聲音的音高，若是在音訊中包含有靜音部分，必須避開以免影響音高的偵測，可以採用能量法來抽取出音訊中的聲音，實作時也可以設定好一個能量門檻，高於此門檻的聲音才需要進行音高的偵測，圖 8 為實作的測試程式執行畫面，可以避開靜音部分與去除雜訊，實驗結果顯示，去除雜訊並不會影響到音高偵測的準確度，平均誤差仍為 0.39%。

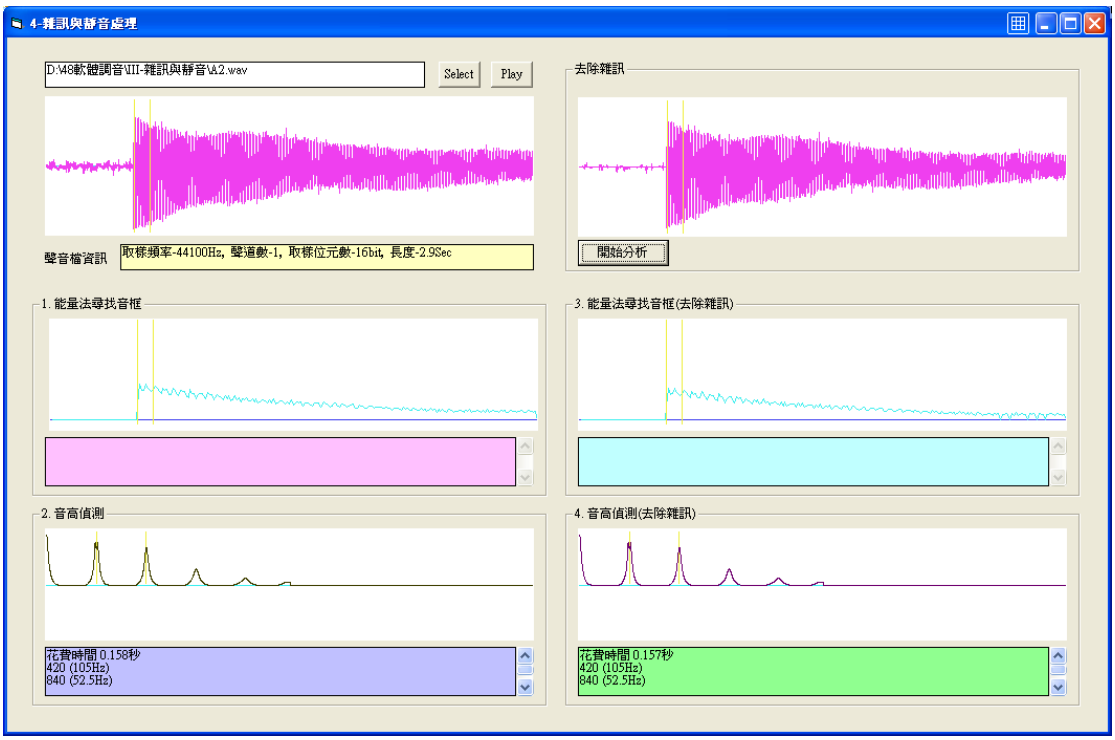

圖 8 去除雜訊與抽取高於能量門檻的音訊

10. 軟體調音器的設計架構

有了前面的測試與實驗經驗，軟體調音器的運作流程如圖 9 所示，聲音接收部分的相關設定為取樣頻率 44100Hz、單聲道、每個取樣點為 16bits、每個緩衝區放置 8192 個取樣點（約 0.1858 秒），每當負責接收聲音的緩衝區填滿後，就改由另一個緩衝區去接收聲音，剛被填滿的緩衝區資料則進行去除雜訊（雜訊門檻設為 $noise_{TH}$ ＝4%）、抽取聲音（能量門檻設為 $Volume_{TH}$ ＝12%）、音高偵測（抽取聲音中 4096 個取樣點）等處理，最後將音高偵測結果即時顯示出來，整個處理過程必須在 0.1858 秒內完成，以免延遲了下一個被填滿的緩衝區之資料處理。

圖 9 調音軟體運作流程

結論與成果

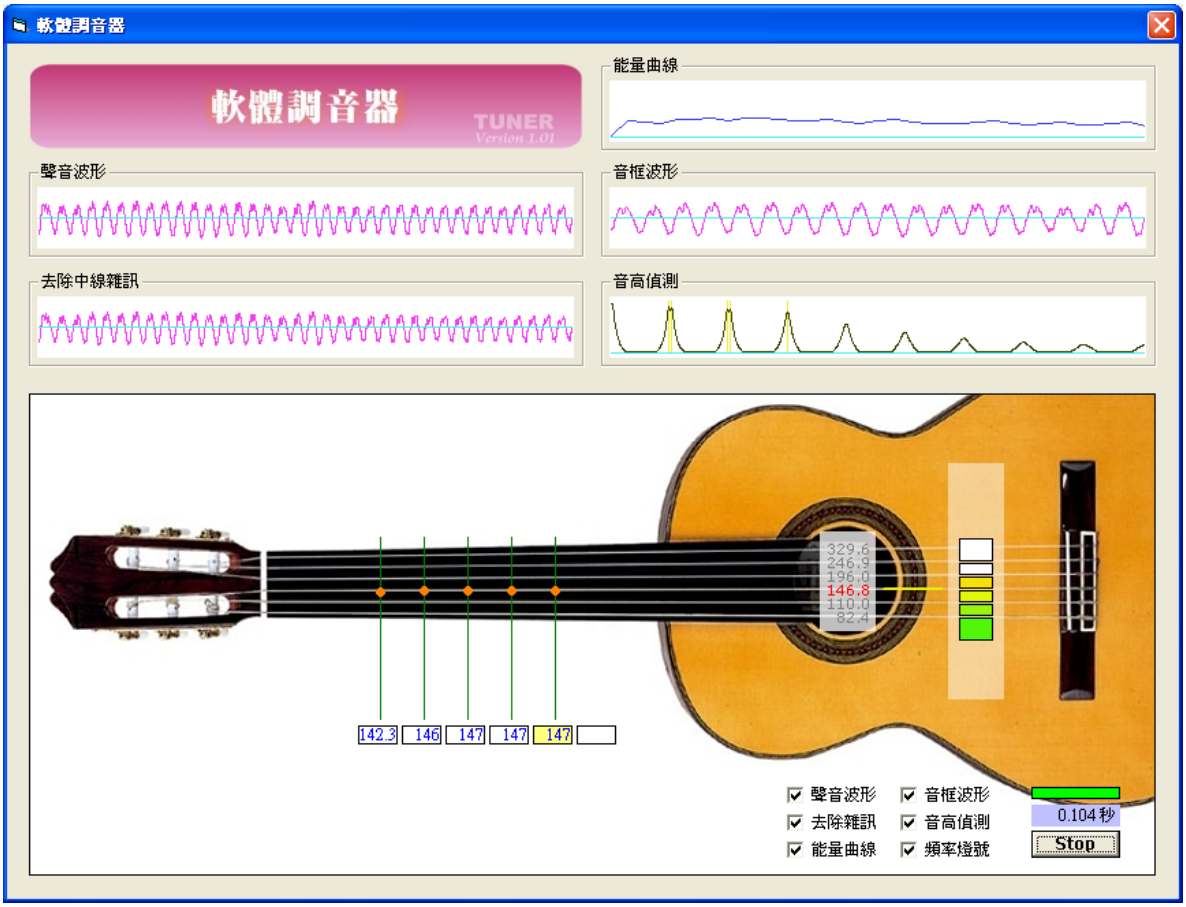

圖 10 軟體調音器運作畫面

利用 Visual Basic 6 來實作軟體調音器，在圖 10 的畫面中，為了方便使用者能夠即時地觀察目前的聲音波形，共使用了 6 個圖形方塊 PictureBox，第 1 個用來顯示緩衝區所接收到的 8192 個取樣點的原始聲音波形，第 2 個顯示去除中線附近雜訊後的聲音波形，並將其能量變化繪製於第 3 個圖形方塊中，第 4 個顯示經過靜音處理，抽取出來 4096 個取樣點的音框波形，音框進行音高偵測的的函數波形則顯示在第 5 個圖形方塊中，最後音高偵測的結果就顯示在第 6 個型方塊中。在全部圖形介面都顯示的情況下，每個音框的音高偵測大約需要 0.35 秒至 0.39 秒的處理時間（CPU 為 Intel Core 2 Duo T5500 1.66GHz），若是關閉第 1 個至第 5 個圖形介面，每個音框的音高偵測則僅需 0.13 秒至 0.15 秒，可以確實達到即時音高偵測的功能。

討論及應用

本篇研究分析了現有音高偵測技術中的自相關函數（ACF）、平均振幅差函數（AMDF）法，加以組合而發展了一個更為精準的音高偵測函數，為了達到即時顯示前聲音的音高供使用者方便於吉他的調音，在實作時採用採用二個緩衝區的機制及 Wiener-Khinchin 定理(利用快速傅立葉轉換與反快速傅立葉轉換來完成 ACF 的運算)來大幅提升程式的運算速度，從實作與測試中得知，本研究的方法可以在很短的時間內（0.13 秒至 0.15 秒）完成每個音框的音高辨識，以即時地方式呈現目前聲音的音高，且其在音高上的平均誤差僅有 0.39%。

無論多麼精準的音高偵測函數都一定會有誤差的產生，因為音高偵測函數所偵測出來的音訊的週期均為整數，再以此整數的音訊週期計算出來的音訊頻率，未必能與吉他各弦的頻率完全吻合。以第 4 弦為例，在取樣頻率為 44100Hz 的情況下，其理論週期的為 300.342，但是音高偵測函數能夠偵測出來的週期不是 300 就是 301，相對應的頻率分別為 147.000Hz 與 146.512Hz，都無法與正確的頻率 146.832Hz 相吻合。表格 4 為音高偵測函數在取樣頻率為 44100Hz 時，偵測吉他各弦頻率一定會產生的誤差，其平均值為 0.21%。

吉他弦編號	6	5	4	3	2	1
正確頻率（Hz）	82.407	110.000	146.832	195.998	246.942	329.628
理論週期（取樣 44100 Hz）	535.149	400.909	300.342	225.003	178.585	133.787
音高偵測函可能的偵測週期	535 或 536	400 或 401	300 或 301	225 或 226	178 或 179	133 或 134
音高偵測函的偵測頻率（Hz）	82.430 82.276	110.250 109.975	147.000 146.512	196.000 195.133	247.753 246.369	331.579 329.104
誤差百分比（%）	0.09%	0.13%	0.17%	0.22%	0.28%	0.38%

表格 4 音高偵測函數必定會產生的誤差

從理論上來看，接收聲音時的取樣頻率越高，偵測出來音高的解析度越好也越精準，但是相對的在同樣時間內資料量會大很多，處理的時間也會越久，為了達到即時顯示音高偵測的結果，取樣頻率與音框大小需作搭配上的考量。未來，希望持續地增加軟體調音器的功能，使其能夠適用於任何弦樂器的調音，例如大、中、小提琴、二胡、琵琶、古箏、豎琴等。

參考資料

1. http://neural.cs.nthu.edu.tw/jang/books/audioSignalProcessing/ 國立清華大學 資訊工程學系 張智星教授網站。
2. 王小川著, “語音訊號處理,” 全華科技, 2004。
3. 彭明柳著, “Visual Basic 6.0 中文專業版徹底研究,” 博碩文化, 1999。
4. S. Uppgard, “Implementation and Analysis of Pitch Tracking Algorithms,” Report for Master of Science Thesis Project, KHT, Stockholm, Sweden, 2001.。
5. http://www.phon.ucl.ac.uk/resource/sfs/。
6. http://www.fon.hum.uva.nl/praat/。
7. http://www.recognisoft.com/。
8. http://www.sil.org/computing/sa/index.htm。
9. Wolfram MathWorld 網站 http://mathworld.wolfram.com/Wiener-KhinchinTheorem.html。