中華民國第 63 屆中小學科學展覽會

作品說明書

國小組　生活與應用科學科(一)

探究精神獎

082811

看得見聲音？—以傅立葉轉換應用聲紋辨識及判讀警示音訊息

學校名稱：新北市板橋區埔墘國民小學

作者：	指導老師：
小六 傅冠勝	陳怡君 黃靜蘋

關鍵詞：傅立葉轉換、聲紋辨識、警示音訊息

摘要

沒注意到警報聲往往使人錯過黃金逃命時間，對聽障者或聽力衰退的老人而言，更是不容忽視的問題。本研究嘗試編寫辨識說話者程式，先用人聲找出辨識特徵方法，以傅立葉轉換得到音檔頻率，再用兩種方法擷取特徵來辨識說話者，**得知不同參數對準確率的影響，進而實際應用於辨識警示音。**

本研究錄製資料庫和測試資料音檔，建立說話特徵後，分析測試資料特徵，與每人資料庫特徵做比對，判斷說話者是誰。結果顯示，一個音檔中取出越多頻率為特徵對準確率沒有明顯影響。將音檔切為小段分析，對準確率影響較明顯，且**每個小段音檔長 0.02 秒到 0.05 秒有最高的準確率**。找出合適的變因後，在辨識警示音上準確率達 100%，並將以 Arduino 連接 LED 與 LCD 螢幕，讓聽障者**看得見聲音**。

壹、前言

一、研究動機

2022 年 6 月媽媽買了 Apple 的產品。我們發現裡面的 Siri 可以設定某個特定的人講話，才有反應並回答。我覺得很有趣，所以就常常對著媽媽的手機說話，卻發現 Siri 有個問題：我對著媽媽的手機和手錶喚醒 Siri 時，Siri 也會回答，並不是只有當初設定的特定使用者講話才會回答。如此可能會啟動到別人手機上的 Siri，或是被別人啟用，造成使用上的困擾。

而因為這個困擾，所以我想到可以用去年參加市賽科展的作品「神奇的傅立葉你到底是誰」[13]中，學習到用單純波合成原複雜波時，用越多的單純波來合成，就能合成出與原複雜波越相似的複雜波，以及經傅立葉轉換分析得出的單純波合起來的合成波有週期的知識。把家人說話的聲音分解、找出特徵，來探討如何辨識說話者，並探討不同方法對辨識準確率的影響。

當我了解變因對辨識準確率的影響後，我發現學校裡面有一位聽障的工友叔叔，他在視訊通話時總是對著螢幕比手語。有一次與老師討論到這位工友叔叔時，老師跟我分享工友叔叔因為是聽障者的緣故，所以購買的新車是車廠特別改裝過，像是會把一些像倒車雷達這種聽覺回饋改成視覺提示。

聽完老師的分享後，我產生了一個疑惑，那就是當聽障者在開車時，旁邊發出了警告的聲音，或是日常生活中火災警鈴響了，他可能因為沒有聽到，而阻礙了救護車的通行，或是發現火災的時候已經太晚，錯過了黃金逃命時間。因此，假如可以透過程式辨識出周遭的一

些警示音，並提供視覺提示，或許可以解決聽障者的困擾，而其他如有重聽的老人家或有需求的人也可受惠。

二、文獻回顧

有了想做辨識警示音的初步想法後，我便開始透過網路以聲紋辨識為關鍵字搜尋相關研究閱讀，並將相關文獻研究結果整理於表 1。

表 1 相關文獻整理表

研究主題	研究方法	研究結論
應用聲紋分析原理於建築物隱藏管路探測之研究[10]	錄音->進行聲紋分析->選出特徵頻率->分析比較，	**管徑**：管徑增加頻率下降。 **材質**：鐵管頻率較高；PVC 管頻率較低。 **管路判別**：低頻特徵頻率高於背景雜訊，就判定內含管路。
識破你的「蛙」言巧語[11]	**建立特徵資料庫：**萃取出蛙聲的音節->計算出梅爾倒頻譜參數 **辨識蛙聲：**待辨識蛙聲計算梅爾倒頻譜參數->與資料庫比對->得到結果。	從實驗結果得知，這個方法的辨識效能極高，系統反應也能在很短的時間完成。
老年人居家使用之聲音辨識與偵測系統開發[12]	**語音辨認**：由最精簡次數輸入訓練。 **非語音之辨認**：以梅爾倒頻譜係數與即時變率作為特徵向量，並運用高斯混和模型分辨聲響。	**詞彙辨認**：本團隊目前未著力。 **聲紋辨認**：準確率有改進空間。 **音源辨認**：未來可繼續開發。

綜合表 1，我從以上三篇文獻中，發現有人運用聲紋辨識偵測隱藏管路、辨識青蛙叫聲、辨識老人出事時可能會發出的聲音，然而，未有研究針對聽障者給予偵測警報的研究發表！因此，讓我更想要來嘗試辨識警示音，以協助有需求的人能夠適時判斷緊急狀況。

三、研究目的

(一)以 Python 語言建構出錄音、分析、比對的程式。

(二)以「取出的頻率數目」來探討對辨識說話者準確率的影響。

(三)以「切為小段後每段小音檔的長度」來探討對辨識說話者準確率的影響。

(四)應用辨識程式判讀生活常見的警示音，並以視覺提示警示音來源。

(五)將辨識警示音的結果以 Arduino 的 LED、LCD 顯示

四、名詞釋義

(一) Siri [5][6]：

由 Apple 公司開發的語音助理，可以用語音控制 iPhone 來完成日常工作。

(二) Python [1]：

一種直譯式、物件導向的程式語言，具有可擴充的完整函式庫。

(三)Arduino [2]：

一種價格低廉、開發容易的控制板，只要以 USB 線連接電腦，就能開始進行微電腦互動實驗。

(四) 辨識說話者 [7]：

判斷不明語音的說話者是在一組已註冊的說話者中的哪一個人，**可將裝置個人化**。

(五) 準確度 [8][9]：

準確度是一個統計學上的概念，指在每一次獨立的測量之間，其平均值與已知的理論值之間的差距。若多次測量平均值接近理論值，我們就可以說數據具有「高準確度」。

(六) 複雜波、單純波 [4]：

單一個頻率的 sin、cos 波是**單純波**；說話、樂器等的聲波是**複雜波**，如圖 1。

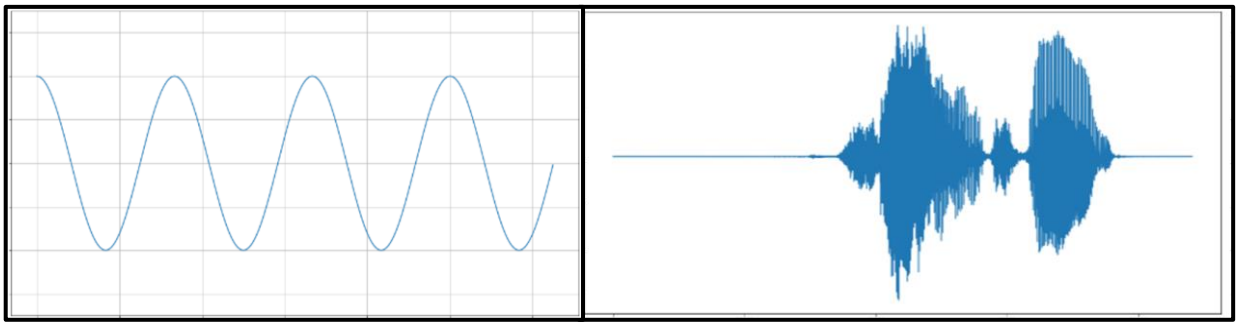


圖 1 複雜波和單純波的圖形

(七) 傅立葉級數 [4]：

「任何函數都可以由無限多個 sin、cos 函數加總而得」，這種由無限多個 sin、cos 函數組合而成的數學式稱作「傅立葉級數」，如以下公式：

$$f(x) = \frac{a_0}{2} + a_1 \cos x + a_2 \cos 2x + a_3 \cos 3x \ldots + b_1 \sin x + b_2 \sin 2x + b_3 \sin \sin 3x \ \ldots$$

其中 f(x)為原函式，級數中每個 sin 與 cos 函數的倍數稱作傅立葉係數(如上式a_1、a_2、$a_3 \ldots$ b_1、b_2、b_3 ...)。原函式可以被等號右邊各個不同頻率(上式中x、$2x$、$3x$等部分)和強度(上式中$a1$、$b1$)的 cos 函式和 sin 函式加總而得。**此公式可在本研究中應用於分解複雜波為數個不同頻率的單純波的加總**。

(八) 傅立葉轉換 [4]：

將一個函數寫成傅立葉級數的過程，稱為「傅立葉轉換」。如果將聲音的複雜波視為函數，將其傅立葉轉換分解成許多 sin、cos 波，就可以知道這段聲波裡包含了「**多高的音**」、「

這些音的**音量有多大**」，再藉此分析出聲音特徵。所以，進行傅立葉轉換後的複雜波可分析出「**由哪些單純波組成**」、「**單純波的頻率是多高**」、「**頻率的強度有多大**」。

貳、研究設備及器材

本研究的研究設備有：筆記型電腦、anaconda 軟體、Arduino 軟體、Tinkercad 軟體、錄音語詞卡、Arduino UNO 版、LED 燈、LCD 顯示裝置 (I2C 1602)、杜邦線數條。

參、研究過程或方法

本研究共分為四項子研究，為了找出聲紋辨識可能的關鍵變因，我們先從最易取得且聲紋最接近的家人開始探究，期許找到辨識說話者的方式後，應用到警示音的辨識與實際提供視覺提示，本段落針對研究架構、程式設計、前置作業與實驗設計分說如下：

一、研究架構圖

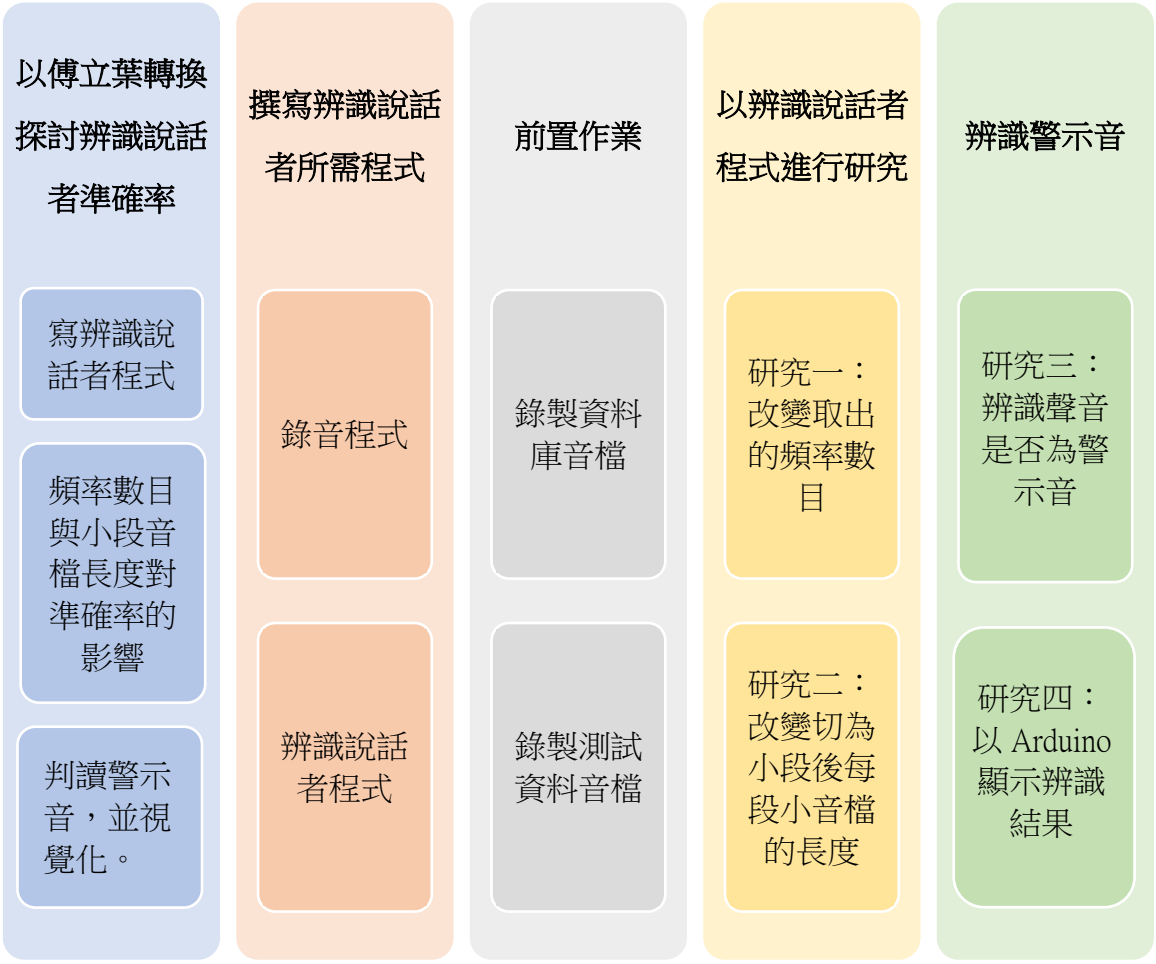


圖 2 研究架構圖

二、前置作業－建立資料庫及測試資料音檔建立

建立說話者資料庫及測試資料的音檔，用來分析音檔中所含單純波的頻率及強度，並比

對出說話者。

(一)錄「資料庫」音檔：

1.使用表 2 的 prefix 變數，選擇要錄**哪個人。**

表 2 資料庫音檔檔名前綴表

資料庫	第一人	第二人	第三人	第四人
prefix	one_	two_	three_	four_

2.使用表 3 的 index 變數，選擇要錄**哪個語詞**。

表 3 資料庫音檔檔名編號表

資料庫	早安	午安	晚安	起床	睡覺	上車	下車	開門	關門	你好
index	1	2	3	4	5	6	7	8	9	10

3.音檔檔名的組合方式：prefix + index + .wav

4.執行圖 3 程式，進行錄音及存檔。

```
import sounddevice as sd
from scipy.io.wavfile import write
import matplotlib.pyplot as plt

sample_rate = 44100  # 取樣率
seconds = 2   # 錄幾秒  #一句話的長度有待想想

prefix = 'one_'
index = '1'
filename = prefix + index + '.wav'  #音檔存檔檔名
"""錄音"""
print('start1')  #預備
data = sd.rec(int(seconds * sample_rate), samplerate=sample_rate, channels=1, dtype='int16')
print('start2')   #開始
sd.wait()  # 等待錄音結束
print('record finished')  #結束
write(filename, sample_rate, data)  # 存成WAV檔
print('save finished')  #儲存完成
```

圖 3 錄音程式

4.重複步驟 2～3，依序將 index 變數換成下一個語詞，並進行錄音及存檔，直到此人十個語詞都錄好，完成「**第一人資料庫**」。

5. 重複步驟 1～4，依序將 prefix 變數換成第二人至第四人，執行錄音及存檔，完成所有人的**說話者資料庫**音檔之建立。

(二) 錄「測試資料」音檔：

1.改變表 4 和表 5 prefix 和 index 變數。

2.執行與錄資料庫音檔時一樣的步驟，完成四個人的 15 句語詞測試資料音檔。

3.建立第一人至第四人的**測試資料**音檔。

表 4 測試資料音檔檔名前綴表

測試資料	第一人	第二人	第三人	第四人
prefix	new_1	new_2	new_3	new_4

表 5 測試資料音檔檔名編號表

測試資料	早安	起床	睡覺	上車	開門	你好	國語	數學	人類	蚊子	秋分	寒露	蘋果	香蕉	公車
index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(三)依照(一)、(二)步驟進行錄音：

資料庫第一人的「早安」音檔存檔檔名呈現為"one_1.wav"。

三、用 Python 寫出辨識說話者程式進行研究：

(一)建立「單人說話特徵資料庫」－以第一人為例

1.從建立好的**第一人資料庫** 10 句音檔中，讀取第一個音檔"one_1.wav"，做傅立葉轉換，即可得到這筆音檔所含的**單純波頻率**。

2.取強度前 3 強的頻率記錄。

3.重複步驟 1～2 轉換完第一人的 10 句音檔。

4.將第 3 步記錄的第一人 10 句音檔**前三強**的共 30 個頻率資料，**以 10Hz 為組距**分組，統計每組頻率計數次數。

舉例：

有[10,15,20,22,25,31,35,50,52] Hz，分組時

[10,15]會被分到 10Hz 那組，

[20,22,25]會分到 20Hz 那組，

[31,35]會分到 30Hz 那組，

[50,52]會分到 50Hz 那組。如圖 4：

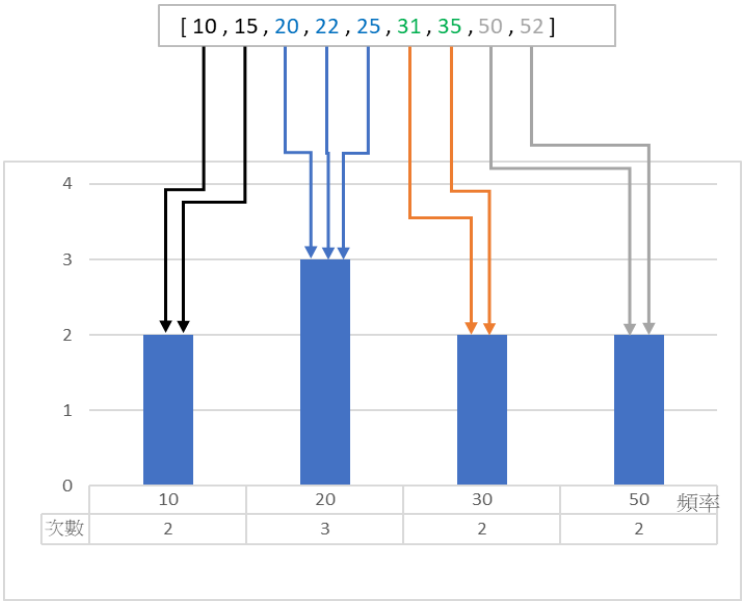


圖 4 單人說話特徵資料頻率前三強分組統計圖

5.第 4 步得到的頻率組別統計結果就是第一人說話時**最常出現的頻率組別以及出現次數**。將此統計表放入說話特徵資料庫中，即代表第一人的說話特徵。

6. 重複步驟 1～5，處理其他三個人的說話特徵資料音檔，建立完所有人**說話特徵資料庫**。

(二) 如何辨識說話者 －以第一人的「早安」做為測試資料舉例

1. 從建立好的**第一人測試資料** 15 句音檔中，讀取第一人的早安測試音檔，做傅立葉轉換，即可得到第一人的「早安」所含的**單純波頻率**。

2. 取強度前 3 強的頻率記錄。

3. 將前 3 強的頻率以 **10Hz 為組距分組**，統計每組頻率計數次數，得到第一人的早安測試音檔**最常出現的頻率組別與出現次數**：

舉例：

取前 3 強頻率[19,22,25] Hz 做分組統計，分組時：

[19]會被分到 10Hz 那組，[22,25]會分到 20Hz 那組。如圖 5：

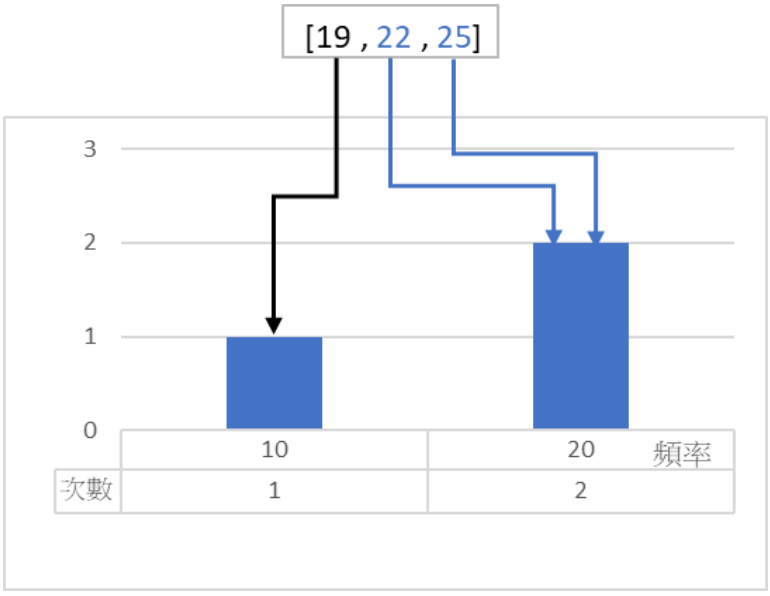


圖 5 測試資料一句話說話頻率特徵前三強分組統計圖

4.以**第一人早安說話特徵**與**說話特徵資料庫**中四個人的說話特徵**做比對** ，即可推論出說話者是誰。

(三)比對的方式－以第一人的「早安」做為測試資料舉例

比對是**判斷**一句測試資料的特徵和說話特徵資料庫中**誰的說話特徵最相似**。

我們使用以下計算得分的方式來判斷兩個資料的相似度。

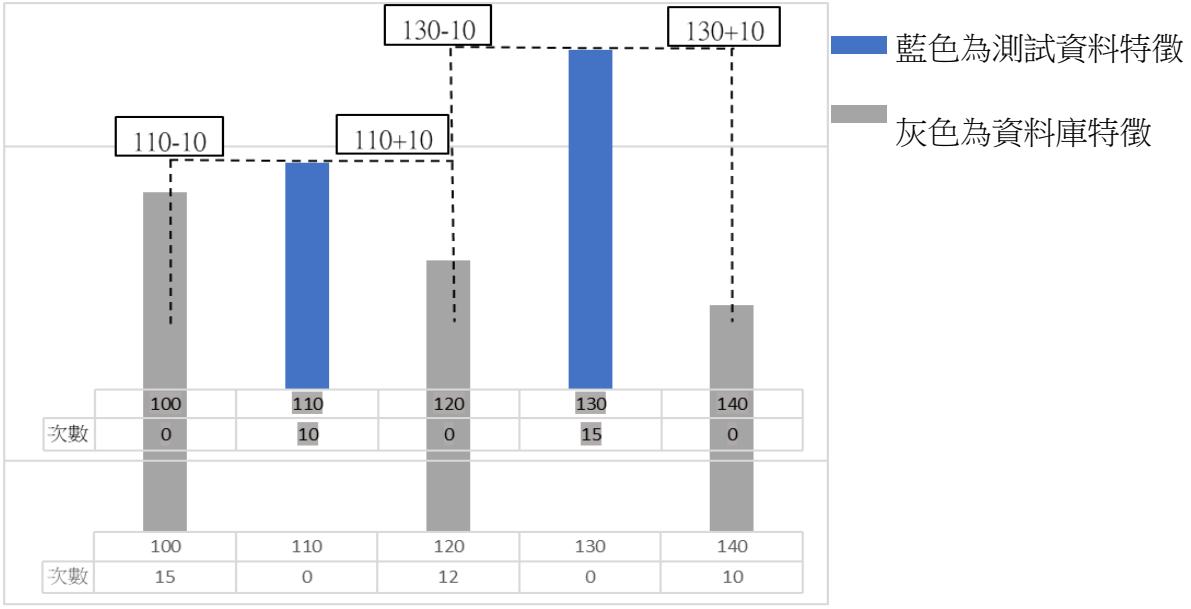
1.在**說話特徵資料庫**中讀出第一人的說話頻率。

2.在**測試語音的特徵**中取出第一個**頻率與其次數**。

3.若該頻率是在這人說話特徵中**某筆頻率加減 10Hz 範圍內**，就判定這兩個頻率**有對應到**，則計算此次得分，計算方式是將此**測試語音特徵頻率的次數**和對應到的說話特徵資料庫中的**頻率的次數相乘**，並將得到的數值**加入那人的總分**。

4.重複步驟 2～3，將「**早安**」**的頻率一個個取出來與第一人說話特徵資料庫(所有說話特徵頻率)比對**，最後會得到第一人的**總分**。

比對方式如圖 6：



1. 110Hz 和 100Hz 有比到所以將兩個的次數相乘，得到 150。
2. 110Hz 和 120Hz 有比到所以將兩個的次數相乘，得到 120。
3. 130Hz 和 120Hz 有比到所以將兩個的次數相乘，得到 180。
4. 130Hz 和 140Hz 有比到所以將兩個的次數相乘，得到 150。

將四次得到的分數相加，得到600，這個數即為此人的總分。

圖 6 加減 10Hz 解釋圖

5.重複步驟 1～4，將早安的每個頻率與第二人到第四人的說話特徵依序比對，會得到所有人的總分。

6.比較所有人的總分，**分數最高**的就是**說話者**。

四、研究設計

研究一：改變傅立葉轉換後資料庫和測試資料音檔取出的「頻率數目」，為前三強、前六強、前九強，分別探討對辨識說話者準確率的影響

(一)研究構想

在上次的科展中學到，用單純波合成原始波形時，使用越多單純波來合成，就能合成出與原始波形**越相似**的複雜波。所以這次想知道進行說話者比對時，使用的語音特徵資料**數目多或少**對辨識準確率的影響。

(二)研究目的

觀察改變分析後取出的頻率數目，對辨識說話者準確率的影響。

(三)研究步驟

1.將圖 7、圖 8 的 **max_n** 變數設為 3。

2.使用表 6 的資料改變圖 7 中 **this_man** 變數，選擇要讀取**哪個人的資料庫音檔**：

表 6 讀取資料庫音檔變數表

第一人	第二人	第三人	第四人
one_man	two_man	three_man	four_man

3.使用表 7 的資料改變圖 7 中 **this_man_str** 變數(組成特徵存檔檔名的字串)，可以改變此人說話特徵存檔的檔名：

表 7 資料庫特徵儲存檔名表

第一人	第二人	第三人	第四人
one_man	two_man	three_man	four_man

4.執行圖 7， 得到經傅立葉轉換和分組統計後此人的**資料庫說話特徵**，並依第 3 步產生的資料庫說話特徵存檔檔名存檔。

```
import numpy as np
from scipy.fft import fft, fftfreq
import soundfile as sf
import matplotlib.pyplot as plt

the_all_three_first_1=[] #宣告前三強的串列
one_man=['one_1.wav','one_2.wav','one_3.wav','one_4.wav','one_5.wav',\
         'one_6.wav','one_7.wav','one_8.wav','one_9.wav','one_10.wav']  #檔名串列
two_man=['two_1.wav','two_2.wav','two_3.wav','two_4.wav','two_5.wav',\
         'two_6.wav','two_7.wav','two_8.wav','two_9.wav','two_10.wav']  #檔名串列
three_man=['three_1.wav','three_2.wav','three_3.wav','three_4.wav',\
           'three_5.wav','three_6.wav','three_7.wav','three_8.wav',\
           'three_9.wav','three_10.wav']  #檔名串列
four_man=['four_1.wav','four_2.wav','four_3.wav','four_4.wav','four_5.wav',\
          'four_6.wav','four_7.wav','four_8.wav','four_9.wav','four_10.wav']  #檔名串列

this_man = one_man
this_man_str = 'one_man'
n_man_feature_save = '資料庫的txt\\' + this_man_str +'_feature.txt'

max_n = 3                       #找尋前n大
```

```
for filename in this_man:
    # 讀取資料，和取樣率
    data, sample_rate = sf.read(filename, dtype='int16')
    data_size = data.size
    sampling_rate = sample_rate
    max_time = int(data_size / 10) #調整搜尋資料的比例

    data_fft = fft(data)
    freq_point = fftfreq(data_size, 1/sampling_rate)
    mag = (2/data_size) * np.abs(data_fft) # Magnitude
    for i in range(len(mag)):
        mag[i] = round(mag[i], 2)

    fig , ax = plt.subplots()

    ax.plot(freq_point[0:max_time], mag[0:max_time])
    ax.set_xlabel('Hz')
    ax.set_ylabel('amp')
```

```
#分組統計
import json
old_three_first_1={}  #dict
for the_three_first_1 in the_all_three_first_1:  #*the_three_first_1*

    level = the_three_first_1//10*10

    if level in old_three_first_1:
        old_three_first_1[level] = old_three_first_1.get(level)+1
    else:
        old_three_first_1.setdefault(level,1)
print(old_three_first_1)

#把特徵存檔
with open(n_man_feature_save, 'w') as file:
     file.write(json.dumps(old_three_first_1))
```

圖 7 資料庫傅立葉轉換和分組統計的程式

5.重複步驟 2～ 5 做完第一人到的四人的資料庫的說話特徵。

6.使用表 8 的資料改變圖 8 中 this_man 變數，選擇要讀取哪個人的**測試資料**的音檔：

表 8 讀取測試資料音檔變數表

第一人	第二人	第三人	第四人
new_1_man	new_2_man	new_3_man	new_4_man

7.使用表 9 的資料改變圖 8 中 this_man_str 變數，以改變分組完的測試資料存檔檔名：

表 9 測試資料特徵儲存檔名表

第一人	第二人	第三人	第四人
new_1_man	new_2_man	new_3_man	new_4_man

8.執行圖 8，得到經傅立葉轉換和分組統計後此人的**測試資料的說話特徵**，並存檔。

```
import numpy as np
from scipy.fft import fft, fftfreq
import soundfile as sf
import matplotlib.pyplot as plt

the_all_three_first_1=[] #宣告前三強的串列
new_1_man=['new_1_1.wav','new_1_2.wav','new_1_3.wav','new_1_4.wav','new_1_5.wav','new_1_6.wav',\
           'new_1_7.wav','new_1_8.wav','new_1_9.wav','new_1_10.wav','new_1_11.wav',\
           'new_1_12.wav','new_1_13.wav','new_1_14.wav','new_1_15.wav'] #(F)
new_2_man=['new_2_1.wav','new_2_2.wav','new_2_3.wav','new_2_4.wav','new_2_5.wav','new_2_6.wav',\
           'new_2_7.wav','new_2_8.wav','new_2_9.wav','new_2_10.wav','new_2_11.wav',\
           'new_2_12.wav','new_2_13.wav','new_2_14.wav','new_2_15.wav'] #(D)
new_3_man=['new_3_1.wav','new_3_2.wav','new_3_3.wav','new_3_4.wav','new_3_5.wav','new_3_6.wav',\
           'new_3_7.wav','new_3_8.wav','new_3_9.wav','new_3_10.wav','new_3_11.wav',\
           'new_3_12.wav','new_3_13.wav','new_3_14.wav','new_3_15.wav']  #(M)
new_4_man=['new_4_1.wav','new_4_2.wav','new_4_3.wav','new_4_4.wav','new_4_5.wav','new_4_6.wav',\
           'new_4_7.wav','new_4_8.wav','new_4_9.wav','new_4_10.wav','new_4_11.wav',\
           'new_4_12.wav','new_4_13.wav','new_4_14.wav','new_4_15.wav']  #(G)
AA=0
```

```
this_man_str = 'new_3_man'
this_man = new_3_man
n_man_fifteen_feature_save = '測試資料的txt\\' + this_man_str +'_feature.txt'

max_n = 3                       #找尋前n大
```

```
for filename in this_man:
    AA=AA+1
    # 讀取資料，和取樣率
    data, sample_rate = sf.read(filename, dtype='int16')
    data_size = data.size
    sampling_rate = sample_rate
    max_time = int(data_size / 10) #調整搜尋資料的比例 *16*

    data_fft = fft(data)
    freq_point = fftfreq(data_size, 1/sampling_rate)

    mag = (2/data_size) * np.abs(data_fft) # Magnitude
    for i in range(len(mag)):
        mag[i] = round(mag[i], 2)
    print("NO.",AA)
    ll = sorted(zip(mag[0:max_time], freq_point[0:max_time]), reverse=True)[:max_n] #找尋前n強

    temp_mag ,temp_freq = zip(*ll)  #此行的temp_mag ,temp_freq僅為暫時 , (unzip), *temporary*
    freq_point=list(temp_freq)    #由tuple轉為list
    print(freq_point)   #僅剩前n強的freq
    the_all_three_first_1.append(freq_point)

print("最終的前三強:")
print(the_all_three_first_1)
print("前三強的數量:",len(the_all_three_first_1))
```

```
#分組統計
import json
old_three_first_list = []
old_three_first_1={}  #dict
for three_first in the_all_three_first_1:
    old_three_first_1={}
    for the_three_first_1 in three_first:  #*the_three_first_1*

        level = the_three_first_1//10*10

        if level in old_three_first_1:
            old_three_first_1[level] = old_three_first_1.get(level)+1
        else:
            old_three_first_1.setdefault(level,1)
    print(old_three_first_1)
    old_three_first_list.append(old_three_first_1)
print(old_three_first_list)
#把特徵存檔
with open(n_man_fifteen_feature_save, 'w') as file:
     file.write(json.dumps(old_three_first_list))
```

圖 8 測試資料傅立葉轉換和分組統計的程式

9. 重複步驟 6 ～ 11 做完所有人測試資料的特徵。

10. 使用表 10 資料改變圖 9 中 tester 變數，以選擇要讀取哪個人的**測試資料特徵**：

表 10 讀取測試資料特徵人選變數表

第一人	第二人	第三人	第四人
new_1_man	new_2_man	new_3_man	new_4_man

11.執行圖 9，讀取所有人的**資料庫說話特徵、讀取 tester 變數指定測試者之測試資料特徵。**

```
import json

#第1人的資料 (D)
with open('資料庫的txt\\one_man_feature.txt') as file:
    old_three_1 = json.load(file)
    print(old_three_1)
#第2人的資料 (F)
with open('資料庫的txt\\two_man_feature.txt') as file:
    old_three_2 = json.load(file)
    print(old_three_2)
#第3人的資料 (M)
with open('資料庫的txt\\three_man_feature.txt') as file:
    old_three_3 = json.load(file)
    print(old_three_3)
#第4人的資料 (G)
with open('資料庫的txt\\four_man_feature.txt') as file:
    old_three_4 = json.load(file)
    print(old_three_4)

#新的人的資料
tester = 'new_3_man'
tester_file = '測試資料的txt\\' + tester + '_feature.txt'
with open(tester_file) as file:
    new_man = json.load(file)
    print(new_man)
print(len(new_man))
#分數
score = [0,0,0,0]
end_score=[0,0,0,0]
#次數
X=0
```

圖 9 讀取資料庫特徵和測試資料特徵程式

12. 執行圖 10，以 tester **變數**的測試資料特徵與所有人的資料庫說話特徵**進行比對**，推論 15 句話的**每一句話說話者是誰**。

```
X=0
end_score=[0,0,0,0]
for new_three in new_man:
    X=X+1
    print('現在執行第',X,end=" ")
    print('個新的資料，正與資料庫裡的四個人比對')
    score = [0,0,0,0]
    #第一人
    for new in new_three:
        for old in old_three_1:
            if(float(new) <= float(old)):
                if(float(new) >= float(old)-10):
                    A=new_three[new]
                    B=old_three_1[old]
                    C=A*B
                    score[0]=score[0]+C

            if(float(new) >= float(old)):
                if(float(new) <= float(old)+10):
                    A=new_three[new]
                    B=old_three_1[old]
                    C=A*B
                    score[0]=score[0]+C
```

```
#第二人
for new in new_three:
    for old in old_three_2:
        if(float(new) <= float(old)):
            if(float(new) >= float(old)-10):
                A=new_three[new]
                B=old_three_2[old]
                C=A*B
                score[1]=score[1]+C

        if(float(new) >= float(old)):
            if(float(new) <= float(old)+10):
                A=new_three[new]
                B=old_three_2[old]
                C=A*B
                score[1]=score[1]+C

#第三人
for new in new_three:
    for old in old_three_3:
        if(float(new) <= float(old)):
            if(float(new) >= float(old)-10):
                A=new_three[new]
                B=old_three_3[old]
                C=A*B
                score[2]=score[2]+C
        if(float(new) >= float(old)):
            if(float(new) <= float(old)+10):
                A=new_three[new]
                B=old_three_3[old]
                C=A*B
                score[2]=score[2]+C
 #第四人
for new in new_three:
    for old in old_three_4:
        if(float(new) <= float(old)):
            if(float(new) >= float(old)-10):
                A=new_three[new]
                B=old_three_4[old]
                C=A*B
                score[3]=score[3]+C

        if(float(new) >= float(old)):
            if(float(new) <= float(old)+10):
                A=new_three[new]
                B=old_three_4[old]
                C=A*B
                score[3]=score[3]+C
print('本次分數為：')
print('第一人的分數:',score[0])
print('第二人的分數:',score[1])
print('第三人的分數:',score[2])
print('第四人的分數:',score[3])
print('')
temp_score = sorted(score,reverse=True)
if(score[0]==score[1]==score[2]==score[3]):
    print('無勝利者')
elif(temp_score[0]==score[0]):
    print('說話者為第一人')
elif(temp_score[0]==score[1]):
    print('說話者為第二人')
elif(temp_score[0]==score[2]):
    print('說話者為第三人')
elif(temp_score[0]==score[3]):
    print('說話者為第四人')
print('')
```

圖 10 比對的程式

13.改變圖 7、圖 8 的 max_n 變數，**將 max_n 變數依序改為 6、9**，代表取前六強的頻率特徵和前九強的頻率特徵。執行步驟 2 ～ 12。

研究二：將資料庫和測試資料音檔切為小段音檔後，以「每段小音檔」為單位做傅立葉轉

換。改變音檔切的長度，探討對辨識說話者準確率的影響。

(一)研究構想

1.上次的科展結果發現：經傅立葉轉換後分析出來的單純波合起來得到的合成波**會有週期**，但原則上聲波是沒有明顯週期的，如果像研究一那樣將整段聲波做傅立葉轉換，分析出的頻率會不準確。

2.說話時每個字發聲的時間內，聲音頻率較為穩定，推測應該有較為穩定的週期，所以我把音檔**切成多個小段音檔後再做分析，將每段小音檔分析的結果合併，統計音檔特徵**，看**小段音檔的長度不同**對辨識說話者準確率的影響。以圖 11 說明：切成小音檔->分析->合併

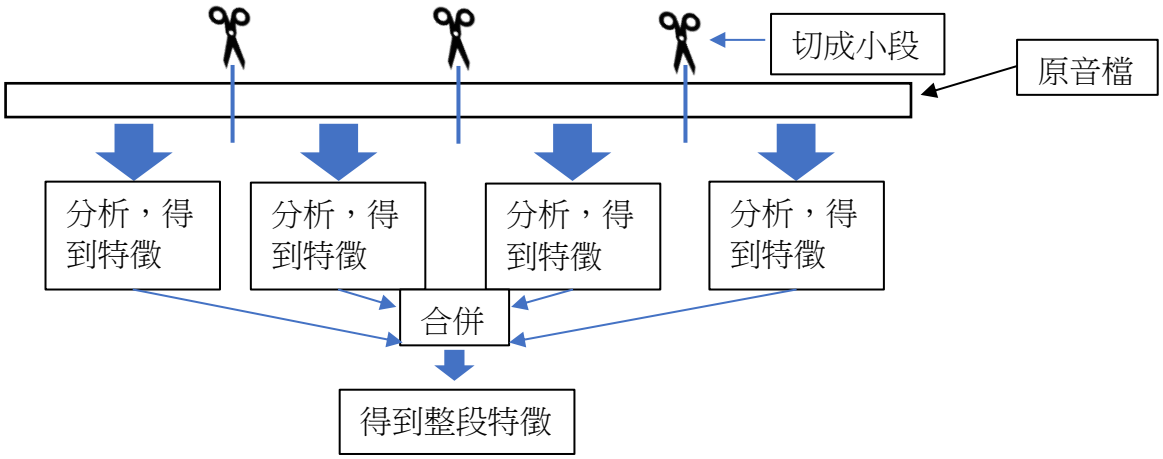


圖 11 切段、分析、合併

(二)研究目的

探討觀察改變每個小段音檔的長度，對辨識說話者準確率的影響。

(三)研究步驟

1.將圖 12、圖 13 的 slices_time 變數設為 0.5。

2.將圖 12、圖 13 的 max_n 變數設為 3。

3.使用表 11 改變圖 12 中 this_man 變數，以選擇要讀取哪個人的**資料庫音檔**(.wav 檔)：

表 11 讀取資料庫音檔變數表

第一人	第二人	第三人	第四人
one_man	two_man	three_man	four_man

4.使用表 12 改變圖 12 中 this_man_str 變數，以改變此人說話特徵存檔檔名：

表 12 資料庫特徵儲存檔名表

第一人	第二人	第三人	第四人
one_man	two_man	three_man	four_man

5.執行圖 12， 得到經傅立葉轉換和分組統計後此人的**資料庫說話特徵**，並存檔。

```
import numpy as np
from scipy.fft import fft, fftfreq, rfft, rfftfreq
import soundfile as sf
import matplotlib.pyplot as plt

the_all_three_first_1=[] #宣告前三強的串列
one_man=['one_1.wav','one_2.wav','one_3.wav','one_4.wav','one_5.wav','one_6.wav','one_7.wav',\
         'one_8.wav','one_9.wav','one_10.wav'] #檔名串列
two_man=['two_1.wav','two_2.wav','two_3.wav','two_4.wav','two_5.wav','two_6.wav','two_7.wav',\
         'two_8.wav','two_9.wav','two_10.wav'] #檔名串列
three_man=['three_1.wav','three_2.wav','three_3.wav','three_4.wav','three_5.wav','three_6.wav',\
           'three_7.wav','three_8.wav','three_9.wav','three_10.wav'] #檔名串列
four_man=['four_1.wav','four_2.wav','four_3.wav','four_4.wav','four_5.wav','four_6.wav',\
          'four_7.wav','four_8.wav','four_9.wav','four_10.wav'] #檔名串列

max_n = 3                    #找尋前n大
slice_time = 0.5  #秒  #500毫秒
this_man = one_man
this_man_str = 'one_man'
n_man_feature_save = '資料庫的txt\\' + this_man_str +'_feature.txt'

for filename in this_man:
    #filename = 'one_1.wav' 此行在for時已經給filename
    # 讀取資料，和取樣率
    data, sample_rate = sf.read(filename, dtype='int16')

    data_size = data.size
    sampling_rate = sample_rate
    max_time = int(data_size / 10) #調整搜尋資料的比例 *16*
    print('音檔:',filename)
    print('聲音共有多少資料點:',data_size)
    print('sampling_rate',sampling_rate)
    print('共幾塊小切片',int(data_size/(sample_rate*slice_time)))

    for i in range(0,int(data_size/(sample_rate*slice_time)),1): #算幾個切片
        print(i)
        data_cut = data[i*int(sample_rate*slice_time):(i+1)*int(sample_rate*slice_time)]
        data_cut_len = len(data_cut)
        print(data_cut_len)

        data_fft = rfft(data_cut)
        freq_point = rfftfreq(data_cut_len, 1/sampling_rate)
        print("len:")
        print(len(freq_point))
        print("頻率:")
        print(freq_point[0], freq_point[1],freq_point[2], freq_point[3])

        mag = (2/data_cut_len) * np.abs(data_fft) # Magnitude

        for i in range(len(mag)):
            mag[i] = round(mag[i], 2)

        ll = sorted(zip(mag[0:max_time], freq_point[0:max_time]), reverse=True)[:max_n] #找尋前3強度
        temp_mag ,temp_freq = zip(*ll)  #此行的temp_mag ,temp_freq僅為暫時 , (unzip), *temporary*
        freq_point=list(temp_freq)    #由tuple轉為list
        print(freq_point)   #僅剩前三強的freq
        print(ll)
        for power,freq in ll:#*range*
            if(power>=100):
                the_all_three_first_1.append(freq)

print("最終的前三強:")
print(the_all_three_first_1)  #應該有30項
print("前三強的數量:",len(the_all_three_first_1))
```

```
#分組統計
import json
old_three_first_1={}  #dict
for the_three_first_1 in the_all_three_first_1:  #*the_three_first_1*

    level = the_three_first_1//10*10

    if level in old_three_first_1:
        old_three_first_1[level] = old_three_first_1.get(level)+1
    else:
        old_three_first_1.setdefault(level,1)
print(old_three_first_1)

#把特徵存檔
with open(n_man_slices_feature_save, 'w') as file:
     file.write(json.dumps(old_three_first_1))
```

圖 12 資料庫傅立葉轉換和分組統計的程式

6.重複步驟 1 ～ 5 完成第一人至第四人的資料庫的說話特徵。

7.使用表 13 改變圖 13 中 **this_man** 變數，以選擇要讀取哪個人的**測試資料的音檔**：

表 13 讀取測試資料音檔變數表

第一人	第二人	第三人	第四人
new_1_man	new_2_man	new_3_man	new_4_man

8.使用表 14 改變圖 13 中 **this_man_str** 變數，以改變分組完的測試資料存檔檔名：

表 14 測試資料特徵儲存檔名表

第一人	第二人	第三人	第四人
new_1_man	new_2_man	new_3_man	new_4_man

9.執行圖 13，得到經傅立葉轉換和分組統計後此人的**測試資料的說話特徵**，並依步驟 8 產生的測試資料存檔檔名存檔。

```
import numpy as np
from scipy.fft import fft, fftfreq, rfft, rfftfreq
import soundfile as sf
import matplotlib.pyplot as plt  #**
slices_three_first = []
the_all_three_first_1 = [] #宣告前三強的串列

new_1_man=['new_1_1.wav','new_1_2.wav','new_1_3.wav','new_1_4.wav','new_1_5.wav','new_1_6.wav',\
           'new_1_7.wav','new_1_8.wav','new_1_9.wav','new_1_10.wav','new_1_11.wav','new_1_12.wav',\
           'new_1_13.wav','new_1_14.wav','new_1_15.wav'] #(F)
new_2_man=['new_2_1.wav','new_2_2.wav','new_2_3.wav','new_2_4.wav','new_2_5.wav','new_2_6.wav',\
           'new_2_7.wav','new_2_8.wav','new_2_9.wav','new_2_10.wav','new_2_11.wav','new_2_12.wav',\
           'new_2_13.wav','new_2_14.wav','new_2_15.wav'] #(D)
new_3_man=['new_3_1.wav','new_3_2.wav','new_3_3.wav','new_3_4.wav','new_3_5.wav','new_3_6.wav',\
           'new_3_7.wav','new_3_8.wav','new_3_9.wav','new_3_10.wav','new_3_11.wav','new_3_12.wav',\
           'new_3_13.wav','new_3_14.wav','new_3_15.wav']  #(M)
new_4_man=['new_4_1.wav','new_4_2.wav','new_4_3.wav','new_4_4.wav','new_4_5.wav','new_4_6.wav',\
           'new_4_7.wav','new_4_8.wav','new_4_9.wav','new_4_10.wav','new_4_11.wav','new_4_12.wav',\
           'new_4_13.wav','new_4_14.wav','new_4_15.wav']  #(G)
AA=0
max_n = 3                       #找尋前n大
slice_time = 0.5  #秒
this_man_str = 'new_3_man'
this_man = new_3_man
n_man_fifteen_feature_save = '測試資料的txt\\' + this_man_str +'_feature.txt'
```

```
for filename in this_man:
    AA=AA+
    # 讀取資料，和取樣率
    data, sample_rate = sf.read(filename, dtype='int16')

    data_size = data.size
    sampling_rate = sample_rate
    max_time = int(data_size / 10) #調整搜尋資料的比例
    print('音檔:',filename)
    print('聲音共有多少資料點:',data_size)
    print('sampling_rate',sampling_rate)
    print('共幾塊小切片',int(data_size/(sample_rate*slice_time)))
    slices_three_first = []
    for i in range(0,int(data_size/(sample_rate*slice_time)),1): #算幾個切片
        print(i)
        data_cut = data[i*int(sample_rate*slice_time):(i+1)*int(sample_rate*slice_time)]
        data_cut_len = len(data_cut)
        print(data_cut_len)

        data_fft = rfft(data_cut)
        freq_point = rfftfreq(data_cut_len, 1/sampling_rate)    #441

        mag = (2/data_cut_len) * np.abs(data_fft) # Magnitude
        for i in range(len(mag)):
            mag[i] = round(mag[i], 2)
        print("NO.",AA)
        ll = sorted(zip(mag[0:max_time], freq_point[0:max_time]), reverse=True)[:max_n] #找尋前n強度

        temp_mag ,temp_freq = zip(*ll)  #此行的temp_mag ,temp_freq僅為暫時 , (unzip), *temporary*
        freq_point_n=list(temp_freq)    #由tuple轉為list
        print("freq",freq_point_n)   #僅剩前n強的freq
        print("ll",ll)
        for power,freq in ll:#*range*
            if(power>=100):
                slices_three_first.append(freq)
        print("slices_three_first",slices_three_first)
    the_all_three_first_1.append(slices_three_first)

print("最終的前三強:")
print(the_all_three_first_1)
print("前三強的數量:",len(the_all_three_first_1))
```

```
#分組統計
import json
old_three_first_1={}  #dict
for three_first in the_all_three_first_1:
    old_three_first_1={}
    for the_three_first_1 in three_first:  #*the_three_first_1*

        level = the_three_first_1//10*10

        if level in old_three_first_1:
            old_three_first_1[level] = old_three_first_1.get(level)+1
        else:
            old_three_first_1.setdefault(level,1)
    print(old_three_first_1)
#把特徵存檔
with open(n_man_fifteen_slices_feature_save, 'w') as file:
     file.write(json.dumps(old_three_first_list))
```

圖 13 測試資料傅立葉轉換和分組統計的程式

10. 重複步驟 7 ～ 11 做完所有人的測試資料的特徵。

11. 使用表 15 改變圖 14 中 tester 變數，選擇要讀取哪個人的**測試資料特徵**：

表 15 讀取測試資料特徵人選變數表

第一人	第二人	第三人	第四人
new_1_man	new_2_man	new_3_man	new_4_man

12.執行圖 14，讀取所有人的資料庫說話特徵、以 tester 變數**改變要讀取誰的測試資料特徵**。

```
import json

#第1人的資料 (D)
with open('資料庫的txt\\one_man_feature.txt') as file:
    old_three_1 = json.load(file)
    print(old_three_1)
#第2人的資料 (F)
with open('資料庫的txt\\two_man_feature.txt') as file:
    old_three_2 = json.load(file)
    print(old_three_2)
#第3人的資料 (M)
with open('資料庫的txt\\three_man_feature.txt') as file:
    old_three_3 = json.load(file)
    print(old_three_3)
#第4人的資料 (G)
with open('資料庫的txt\\four_man_feature.txt') as file:
    old_three_4 = json.load(file)
    print(old_three_4)

#新的人的資料
tester = 'new_3_man'
tester_file = '測試資料的txt\\' + tester + ' feature.txt'
with open(tester_file) as file:
    new_man = json.load(file)
    print(new_man)
print(len(new_man))
#分數
score = [0,0,0,0]
end_score=[0,0,0,0]
#次數
X=0
```

圖 14 讀取資料庫特徵和測試資料特徵程式

13. 執行圖 15，以 tester 的測試資料特徵與所有人的資料庫說話特徵**進行比對**，推論出 15 句音檔的**每一句話說話者是誰**。

```
X=0
end_score=[0,0,0,0]
for new_three in new_man:
    X=X+1
    print('現在執行第',X,end=" ")
    print('個新的資料，正與資料庫裡的四個人比對')
    score = [0,0,0,0]
    #第一人
    for new in new_three:
        for old in old_three_1:
            if(float(new) <= float(old)):
                if(float(new) >= float(old)-10):
                    A=new_three[new]
                    B=old_three_1[old]
                    C=A*B
                    score[0]=score[0]+C

            if(float(new) >= float(old)):
                if(float(new) <= float(old)+10):
                    A=new_three[new]
                    B=old_three_1[old]
                    C=A*B
                    score[0]=score[0]+C
```

```
#第二人
for new in new_three:
    for old in old_three_2:
        if(float(new) <= float(old)):
            if(float(new) >= float(old)-10):
                A=new_three[new]
                B=old_three_2[old]
                C=A*B
                score[1]=score[1]+C

        if(float(new) >= float(old)):
            if(float(new) <= float(old)+10):
                A=new_three[new]
                B=old_three_2[old]
                C=A*B
                score[1]=score[1]+C
#第三人
for new in new_three:
    for old in old_three_3:
        if(float(new) <= float(old)):
            if(float(new) >= float(old)-10):
                A=new_three[new]
                B=old_three_3[old]
                C=A*B
                score[2]=score[2]+C

        if(float(new) >= float(old)):
            if(float(new) <= float(old)+10):
                A=new_three[new]
                B=old_three_3[old]
                C=A*B
                score[2]=score[2]+C
 #第四人
for new in new_three:
    for old in old_three_4:
        if(float(new) <= float(old)):
            if(float(new) >= float(old)-10):
                A=new_three[new]
                B=old_three_4[old]
                C=A*B
                score[3]=score[3]+C

        if(float(new) >= float(old)):
            if(float(new) <= float(old)+10):
                A=new_three[new]
                B=old_three_4[old]
                C=A*B
                score[3]=score[3]+C
print('本次分數為：')
print('第一人的分數:',score[0])
print('第二人的分數:',score[1])
print('第三人的分數:',score[2])
print('第四人的分數:',score[3])
print('')
temp_score = sorted(score,reverse=True)
if(score[0]==score[1]==score[2]==score[3]):
    print('無勝利者')
elif(temp_score[0]==score[0]):
    print('說話者為第一人')
elif(temp_score[0]==score[1]):
    print('說話者為第二人')
elif(temp_score[0]==score[2]):
    print('說話者為第三人')
elif(temp_score[0]==score[3]):
    print('說話者為第四人')
print('')
```

圖 15 比對的程式

14.改變圖 12、圖 13 的 slices_time 變數，將 slices_time 變數依序改為 0.1、0.05、0.02，代表每一個小段的長度為 0.1 秒；每一個小段的長度為 0.05 秒；每一個小段的長度為 0.02 秒，並執行步驟 3 ～ 13。

研究三：以研究二的方法，辨識出周遭的聲音，是否為警笛聲

(一)研究構想

當聽障者在開車時，旁邊發出了警告的聲音，或是日常生活中火災警鈴響了，他可能因為沒有聽到，而阻礙了救護車的通行，或是發現火災的時候已經太晚，錯過了黃金逃命時間。因此，假如可以透過程式辨識出周遭的警笛一些，並提供視覺的提示，或許可以解決聽障者和有重聽的老人家的人的困擾。

(二)研究目的

使用研究二的方法，判斷能否辨識出救護車、警車、火災警報器等的警笛聲，而其他非警笛的聲音，不會被辨識為警笛。

(三)研究步驟

1.錄資料庫音檔

本研究的資料庫音檔要錄救護車聲、警車聲、火災警報器聲、喇叭聲、電風扇運轉聲和引擎聲，每種各錄 10 次，並依據檔名代碼依次建檔。

表 16 辨識警示音的資料庫音檔命名

聲音種類	音檔來源	建置流程	檔名命名規則
救護車聲	街上錄	手機在街上錄完後，回家撥放給電腦	ambulance_次序 (次序為 1～10)
警車聲	plxabay 網站的 sound effectsc 查 police 下載音檔	下載音檔後撥放	police_car_次序 (次序為 1～10)
火災警報器聲	plxabay 網站的 sound effectsc 查 fire alarm 下載音檔	下載音檔後撥放	fire_alarm_次序 (次序為 1～10)
喇叭聲	在車前錄	手機在車前錄完後，回家撥放給電腦	horn_次序 (次序為 1～10)
電風扇運轉聲	在電風扇旁錄	電腦放在電風扇旁，錄音	fan_次序 (次序為 1～10)
引擎聲	在車前錄	手機在車前錄完後，回家撥放給電腦	engine_次序 (次序為 1～10)

2.錄測試資料音檔

本研究的測試資料音檔要錄救護車聲、警車聲、火災警報器聲、喇叭聲、電風扇運轉聲和引擎聲，每種各錄 3 次。測試資料錄製的來源與資料庫音檔的來源皆一致。而本研究在錄製測試資料時分別使用表 17 的資料將每種聲音的 3 個音檔命名。

表 17 辨識警示音的測試資料音檔命名

音源	救護車聲	警車聲	火災警報器聲	喇叭聲	電風扇運轉聲	引擎聲
prefix	new_	new_	new_fire_alarm_	new_horn_	new_fan_	new_engine_

	ambulance_	police_car_				
index	1	1	1	1	1	1
	2	2	2	2	2	2
	3	3	3	3	3	3

3.錄有車流背景雜音的測試資料音檔

本研究的測試資料音檔還要錄有背景雜音的救護車聲、警車聲、火災警報器聲，每種各錄 3 次。錄製的方法是將**背景雜音**和單純的**警示音聲**一起播放，背景雜音是在文化路錄製的車流聲。而本研究在錄製測試資料時分別使用表 18 的資料將每種聲音的 3 個音檔命名。

表 18 辨識警示音的有背景雜音測試資料音檔命名

音源	有背景雜音的救護車聲	有背景雜音的警車聲	有背景雜音的火災警報器聲
prefix	new_2_ambulance_	new_2_police_car_	new_2_fire_alarm_
index	1	1	1
	2	2	2
	3	3	3

4.分析特徵

將資料庫音檔與測試資料音檔**全部以 max_n 為 3；slices_time 為 0.05** 的方式**分析出特徵**，並存檔。會得到救護車、警車、火災警報器、喇叭、電風扇、引擎的資料庫特徵，以及 6 種聲音各 3 個的測試資料特徵。

5.沒背景雜音的警示音測試資料辨識比對

將救護車、警車、火災警報器的測試資料特徵依序與六個資料庫特徵進行**比對，並統計辨識準確率**。其中，辨識正確的條件為**測試資料的救護車要辨識成救護車**；測試資料的警車要辨識成警車；測試資料的火災警報器要辨識成火災警報器，才算辨識正確。

6.有背景雜音的警示音測試資料辨識比對

將有車流背景雜音的救護車、警車、火災警報器的測試資料特徵依序與六個資料庫特徵進行**比對，並統計辨識準確率**。辨識正確的條件：有雜音**的警示音也要辨識為該警示音種類**，才算辨識正確。

7.非警示音的測試資料辨識比對

將喇叭、電風扇、引擎的測試資料特徵依序與六個資料庫特徵進行**比對，並統計辨識準確率**。辨識正確的條件：不能辨識成**救護車、警車、火災警報器**，才算辨識正確。

研究四：將警示音辨識結果以 Arduino 顯示

(一)研究目的

1.以 Arduino 的 LED 燈，顯示出辨識結果是否為警示音

2.以 Arduino 的 LED 燈與 LCD 螢幕，顯示出辨識結果是哪個聲音

(二)研究步驟

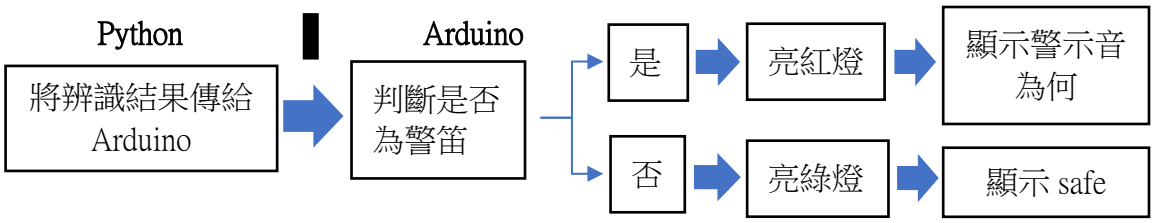


圖 16 研究四的流程圖

1.改變 Python 程式，在比對程式後面改為可將結果傳送給 Arduino 的程式，如圖 17。

```
elif(temp_score[0]==score[0]):
    print('聲音為救護車')
    led = '0'
elif(temp_score[0]==score[1]):
    print('聲音為警車')
    led = '1'
elif(temp_score[0]==score[2]):
    print('聲音為火災警報器')
    led = '2'
elif(temp_score[0]==score[3]):
    print('聲音為電風扇')
    led = '3'
elif(temp_score[0]==score[4]):
    print('聲音為喇叭')
    led = '4'
elif(temp_score[0]==score[5]):
    print('聲音為引擎')
    led = '5'
print('')

ser = serial.Serial('COM8', 9600, timeout=1)
ser.write(led.encode('ascii'))
time.sleep(2)
ser.write(led.encode('ascii'))
readin = ser.read()
while readin:
    print(readin)
    time.sleep(0.01)
    readin = ser.read()
ser.close()
```

圖 17　Python 與 Arduino 通訊的程式

程式會依照表 19 的資料更改傳送給 Arduino 的變數 led。

表 19 依照不同結果傳送的 led 變數表

辨識結果	救護車	警車	火災警報器	電風扇	喇叭	引擎
led	0	1	2	3	4	5

2. 寫出接收便是結果 Arduino 程式，並可依照傳送的值，決定**要亮哪個燈，並使用 LCD 螢幕顯示警示音為何**，如圖 18。

alarm_serial

```
#include <Arduino.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);

char val = 0;
int alarm_led = 12;
int other_led = 11;
int sda = A4;
int scl = A5;
```

```
void setup() {
  Serial.begin(9600);
  pinMode(alarm_led, OUTPUT);
  pinMode(other_led, OUTPUT);
   lcd.init();
   lcd.backlight();
   lcd.clear();
}
void loop() {
  if(Serial.available())
  {
    digitalWrite(alarm_led, LOW);
    digitalWrite(other_led, LOW);
    val = Serial.read();

    if(val == '0' || val == '1' || val == '2')
    {
      digitalWrite(alarm_led, HIGH);
      Serial.println("Warning");
      if(val == '0')
      {
        lcd.clear();
        lcd.print("It is ambulance.");
      }
      if(val == '1')
      {
        lcd.clear();
        lcd.print("It is police car.");
      }
      if(val == '2')
      {
        lcd.clear();
        lcd.print("It is fire alarm.");
      }
    }
    else
    {
      digitalWrite(other_led, HIGH);
      Serial.println("safe");
      lcd.clear();
      lcd.setCursor(6,0);
      lcd.print("safe");
    }
  }
}
```

圖 18 控制 LED、LCD 的程式

程式會依照接收到的資料，決定要亮哪個燈以及顯示的字串，如表 20。

表 20 選擇要亮的燈和要顯示的字

接收到的資料	0	1	2	3	4	5
要亮的燈	alarm_led (紅燈)			other_led (綠燈)		
顯示的字串	It is ambulance	It is police car	It is fire alarm	safe		

3.組裝 Arduino，配線呈現如圖 19。

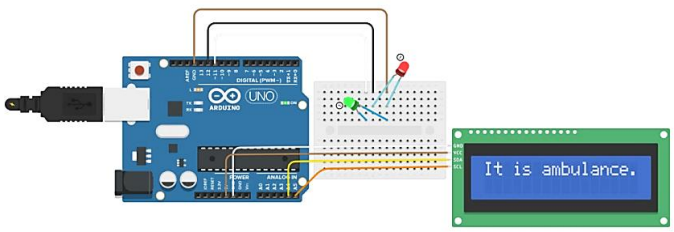


圖 19 研究四 Arduino 配線圖

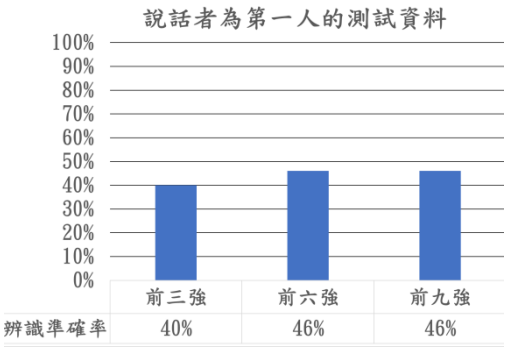
4.將 Arduino 程式(圖 18)傳送給 UNO 版，並執行 Python 程式(圖 17)，就會依照辨識結果顯示是警示音或非警示音。

肆、研究結果

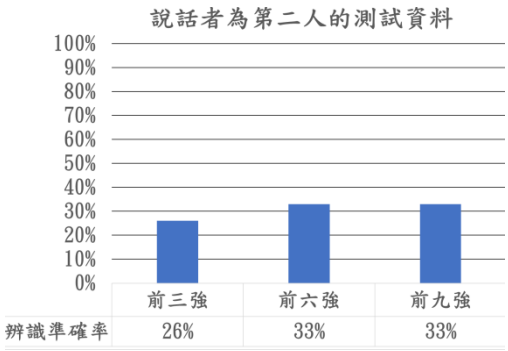
一、研究一：改變取出的頻率數目對辨識說話者準確率的影響

表 21 研究一的結果

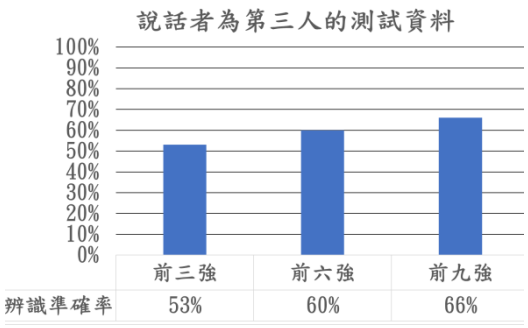
第一人 15 句測試資料辨識的結果			
	前三強	前六強	前九強
辨識到第一人的次數	6 次	7 次	7 次
辨識到第二人的次數	3 次	4 次	4 次
辨識到第三人的次數	1 次	1 次	1 次
辨識到第四人的次數	1 次	1 次	1 次
沒有辨識到人的次數	4 次	2 次	2 次
辨識準確率	40%	46%	46%



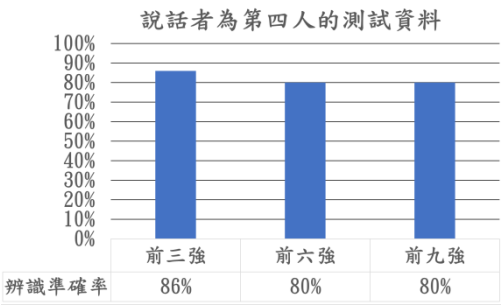
第二人 15 句測試資料辨識的結果			
	前三強	前六強	前九強
辨識到第一人的次數	2 次	0 次	1 次
辨識到第二人的次數	4 次	5 次	5 次
辨識到第三人的次數	0 次	2 次	2 次
辨識到第四人的次數	6 次	7 次	7 次
沒有辨識到人的次數	3 次	1 次	0 次
辨識準確率	26%	33%	33%



第三人 15 句測試資料辨識的結果			
	前三強	前六強	前九強
辨識到第一人的次數	0 次	0 次	0 次
辨識到第二人的次數	3 次	2 次	1 次
辨識到第三人的次數	8 次	9 次	10 次
辨識到第四人的次數	4 次	4 次	4 次
沒有辨識到人的次數	0 次	0 次	0 次
辨識正確率	53%	60%	66%



第四人 15 句測試資料辨識的結果			
	前三強	前六強	前九強
辨識到第一人的次數	0 次	1 次	1 次
辨識到第二人的次數	2 次	2 次	2 次
辨識到第三人的次數	0 次	0 次	0 次
辨識到第四人的次數	13 次	12 次	12 次
沒有辨識到人的次數	0 次	0 次	0 次
辨識準確率	86%	80%	80%



研究一的平均準確率：

表 22 研究一的平均準確率

取出的頻率數目	前三強	前六強	前九強
平均準確率	51%	55%	56%

研究一小結：

(一)當測試資料為第一人、第二人、第三人時，增加取出的頻率數目後，辨識準確率都有緩緩升高。

(二)而當測試資料為第四人時，增加取出的頻率數目後，辨識準確率則稍微下降一點。

二、研究二：改變每段小音檔長度對辨識說話者準確率的影響

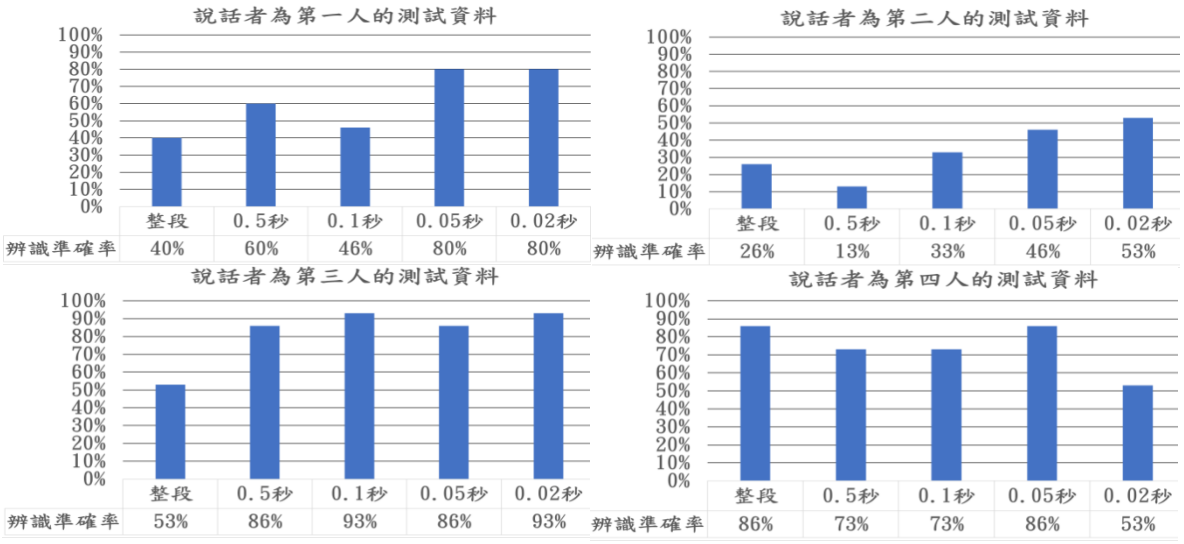
表 23 研究二的結果

第一人 15 句測試資料辨識的結果					
	整段分析	0.5 秒	0.1 秒	0.05 秒	0.02 秒
辨識到第一人的次數	6 次	9 次	7 次	12 次	12 次
辨識到第二人的次數	3 次	3 次	1 次	1 次	2 次
辨識到第三人的次數	1 次	3 次	7 次	2 次	1 次
辨識到第四人的次數	1 次	0 次	0 次	0 次	0 次
沒有辨識到人的次數	4 次	0 次	0 次	0 次	0 次
辨識準確率	40%	60%	46%	80%	80%

第二人 15 句測試資料辨識的結果					
	整段分析	0.5 秒	0.1 秒	0.05 秒	0.02 秒
辨識到第一人的次數	2 次	3 次	0 次	4 次	7 次
辨識到第二人的次數	4 次	2 次	5 次	7 次	8 次
辨識到第三人的次數	0 次	3 次	0 次	0 次	0 次
辨識到第四人的次數	6 次	7 次	10 次	4 次	0 次
沒有辨識到人的次數	3 次	0 次	0 次	0 次	0 次
辨識準確率	26%	13%	33%	46%	53%

第三人 15 句測試資料辨識的結果					
	整段分析	0.5 秒	0.1 秒	0.05 秒	0.02 秒
辨識到第一人的次數	0 次	1 次	1 次	2 次	1 次
辨識到第二人的次數	3 次	1 次	0 次	0 次	0 次
辨識到第三人的次數	8 次	13 次	14 次	13 次	14 次
辨識到第四人的次數	4 次	0 次	0 次	0 次	0 次
沒有辨識到人的次數	0 次	0 次	0 次	0 次	0 次
辨識準確率	53%	86%	93%	86%	93%

第四人 15 句測試資料辨識的結果					
	整段分析	0.5 秒	0.1 秒	0.05 秒	0.02 秒
辨識到第一人的次數	0 次	1 次	1 次	1 次	3 次
辨識到第二人的次數	2 次	2 次	2 次	1 次	4 次
辨識到第三人的次數	0 次	1 次	1 次	0 次	0 次
辨識到第四人的次數	13 次	11 次	11 次	13 次	8 次
沒有辨識到人的次數	0 次	0 次	0 次	0 次	0 次
辨識準確率	86%	73%	73%	86%	53%



研究二的平均準確率：

表 24 研究二的平均準確率

小段音檔長度	整段	0.5 秒	0.1 秒	0.05 秒	0.02 秒
平均準確率	約 51%	約 58%	約 61%	約 75%	約 70%

研究二小結：

(一)當測試資料為第一人時，從整段分析到 0.5 秒辨識準確率有上升的趨勢，但到 0.1 秒時辨識準確率則下降了一些，直到 0.05 秒、0.02 秒時辨識準確率上升到比其他長度高。

辨識準確率：0.02 = 0.05 > 0.5 > 0.1 > 整段分析

(二)當測試資料為第二人時，從整段分析到 0.5 秒辨識準確率有下降了一些，而 0.1 秒到 0.02 秒辨識準確率都有上升的趨勢。

辨識準確率：0.02 > 0.05 > 0.1 > 整段分析 > 0.5

(三)當測試資料為第三人時，從整段分析到 0.5 秒辨識準確率有上升的趨勢，而 0.5 秒到 0.02 秒辨識準確率則是一直在 13、14 次徘徊。

辨識準確率：0.02 = 0.1 > 0.05 = 0.5 > 整段分析

(四)當測試資料為第四人時，從整段分析到 0.5 秒辨識準確率有下降了一些，但到 0.05 秒

時辨識準確率有上升到與整段分析一樣，而到 0.02 秒時辨識準確率卻下降了蠻多。

辨識準確率：0.05 = 整段分析 > 0.1 = 0.5 > 0.02

三、研究三：以研究二的方法，辨識出周遭的聲音，是否為警示音

本研究透過出外錄製與網路上下載後得到的音檔進行辨識，得到的結果如表 24～表 26。

表 24 聲音為沒雜音警示音時的辨識準確率

聲音為沒雜音警示音時的辨識準確率			
測試資料音檔	救護車	警車	火災警報器
辨識準確率	100%	100%	100%

表 25 聲音為有雜音警示音時的辨識準確率

聲音為有雜音警示音時的辨識準確率			
測試資料音檔	救護車	警車	火災警報器
辨識準確率	100%	100%	100%

表 26 聲音並非警示音時的辨識準確率

聲音為非警示音時的辨識準確率			
測試資料音檔	電風扇	喇叭	引擎
辨識準確率	100%	100%	100%

研究三小結：

(一)當聲音為**警示音**時，有雜音與沒雜音的各個測試資料的辨識準確率皆為 100%

(二)當聲音為**雜訊**時，各個測試資料的辨識準確率皆為 100%

四、研究四：將警示音辨識結果以 Arduino 顯示

本研究將辨識警示音的結果傳送至 Arduino 並顯示，得到的結果照片，如圖 20。

聲音為救護車：(亮紅燈)

聲音為警車：(亮紅燈)

聲音為火災警報器：(亮紅燈)

聲音為引擎：(亮綠燈)

圖 20 研究四的結果

研究四小結：

不論聲音是警示音，還是非警示音皆可成功的亮燈，並顯示警示音是哪一個。

伍、討論

一、研究一：改變取出的頻率數目對辨識說話者準確率的影響

(一)當取出的頻率為前三強且辨識準確率較低時，增加取出的頻率數目，準確率會上升。

(二)當取出的頻率為前三強且辨識準確率較高時，增加取出的頻率數目，準確率無改善。

二、研究二：改變每段小音檔長度對辨識說話者準確率的影響

(一)當語音資料為整段分析且辨識準確率較低時，將每個小段音檔的長度變短，再進行分析後，準確率會明顯上升。

(二)當語音資料為整段分析且辨識準確率較高時，將每個小段音檔的長度變短，再進行分析後，準確率無改善。

三、研究三：以研究二的方法，辨識出周遭的聲音，是否為警示音

從研究結果可知，運用**取出的頻率數目為 3、小段音檔的長度為 0.05 秒**的方法，能辨識出警示音與非警示音，且有雜音的警示音仍可正確辨識警示音種類。運用此方法，可以解決聽障者聽不見警示音的問題，也可以在緊急事件發生時的第一時間，用智慧型裝置在螢幕上提醒，並顯示在此緊急事件時該如何應對，更可以讓臺灣高齡化社會中許多有聽覺問題的老人受到視覺的方式告訴他們身邊發生了緊急事件與如何處理，造福更多民眾與家庭。

四、研究四：將警示音辨識結果以 Arduino 顯示

從研究結果可知，用研究四的方法，可以**將辨識的結果傳送到 Arduino 顯示，做出了以智慧型裝置顯示的第一步**。未來可縮小硬體，並改成不需以 USB 連接電腦的方式……

五、建議

由以上結果可知，如果想要讓辨識準確率變高的話，可增加取出的頻率數目，並將小段長度設為 0.05 秒或 0.02 秒。而因為 0.05 秒和 0.02 秒兩個長度在所有組別中準確率都是最高的，所以在分析不知道說話者是誰的語音時，可將兩個長度都進行分析、比對，取出總分高的那個人做為說話者，以表 27 說明

表 27 以不同長度音檔比對表

	0.05 秒	0.02 秒
辨識出來的說話者是誰	第一人	第二人
說話者的總分	700 分	900 分
判斷說話者是誰？		V

陸、結論與建議

一、研究結論

1.「**取出的頻率數目**」並不是影響準確率的主要變因，但增加取出的頻率數目，還是可以讓辨識準確率微微增加。

2.「切為小段後**每個小音檔的長度**」在切到 0.05 秒、0.02 秒的時候，是各組中準確率最高的兩個長度，所以可以依照討論中建議的方法，來選擇要取哪個人做為辨識結果，提高辨識準確率。

3.運用的研究二的方法，取出的頻率數目為 3；小段音檔的長度為 0.05 秒。可以清楚的辨識出**警示音與非警示音**的聲音。

4.運用研究四的方法，可以**將辨識的結果傳送到 Arduino** 顯示，並不會出錯。

二、研究建議

1. 增加辨識度的方法：改變辨識準確率的因素不只上述兩個，所以未來還是可以繼續尋找其他因素，探究變因，例如：小段音檔的切法與取樣的方式調整，或者是語詞資料來源的改變，也許都會改變辨識準確率，可以更進一步的研究與學習。

2.未來實際應用方向：如果讓準確率變高的話，可以增加資訊的安全，更可以使裝置個人化，甚至能應用此技術來判斷機械故障或動物叫聲等。**也可嘗試改善視覺提示裝置**。

柒、參考文獻資料

1. 洪錦魁 (2019)。Python 最強入門邁向頂尖高手之路王者歸來。深智數位。
2. 趙英傑 (2020)。超圖解 Arduino 互動設計入門。旗標科技。
3. 洪維恩 (2020)。C 語言教學手冊。旗標科技。
4. 日本 Newton Press (2020)。三角函數 sin、cos、tan。人人出版。
5. iPhone 使用手冊。在 iPhone 上使用 Siri。https://support.apple.com/zh-tw/guide/iphone/iph83aad8922/ios
6. iPhone 使用手冊。瞭解 iPhone 上的 Siri 可以做哪些事。https://support.apple.com/zh-

tw/guide/iphone/ipha48873ed6/16.0/ios/16.0

7. 數位時代(2017 年 4 月 8 日)。蘋果新專利：Siri 將能設定「暗語」，並辨識主人的聲音。數位時代。https://www.bnext.com.tw/article/43958/apple-siri-new-patent-can-identify-its-owner
8. 張育唐、陳藹然(2011 年 04 月 18)。準確度和精確度。科學 online 。https://highscope.ch.ntu.edu.tw/wordpress/?p=24512
9. 輔仁大學物理系。數據誤差處理－輔仁大學物理系。輔仁大學物理系。http://www.phy.fju.edu.tw/uploads/asset/data/613c0bf4aea57e057c000b72/error_analysis.pdf
10. 朱嗣雍、林意欣、顏伯蒼(2011)。中華民國第 51 屆中小學科學展覽會。應用聲紋分析原理於建築物隱藏管路探測之研究。https://twsf.ntsec.gov.tw/activity/race-1/51/pdf/030810.pdf
11. 陳尹安、蘇奕鴒、徐于康、張廖曉蓉(2006)。中華民國第 46 屆中小學科學展覽會。識破你的「蛙」言巧語。https://twsf.ntsec.gov.tw/activity/race-1/46/senior/0408/040810.pdf
12. 劉奕汶、康仕仲、江秉穎(2011)。GRB 政府研究資訊系統。老年人居家使用之聲音辨識與偵測系統開發。https://www.grb.gov.tw/search/planDetail?id=2333393
13. 作者本人(2022)。新北市 110 學年度中小學科學展覽會。神奇的傅立葉你到底是誰。

【評語】082811

本作品主要發展辨識說話者程式，先用人聲找出辨識特徵方法，以傅立葉轉換得到音檔頻率，再用兩種方法擷取特徵來辨識說話者，得知不同參數對準確率的影響，進而實際應用於辨識警示音。針對聽障者給予偵測警報的研究作品。

研究方法與過程及討論，符合應研究目的需求。

1. 基於 arduino 利用聲音頻率特徵來分辨聲音 對於特殊聲音警示音與非警示音可以非常正確的分辨。

2. 用四個人約十句的短詞做實驗, 以前三強頻率判斷人聲平均正確率約五成。

3. 理解傅立葉專換大致原理, 並能以製作實際系統應用相當難得。

4. 未來可以更加強人聲聲紋比對之正確率。

作品海報

看得見聲音？

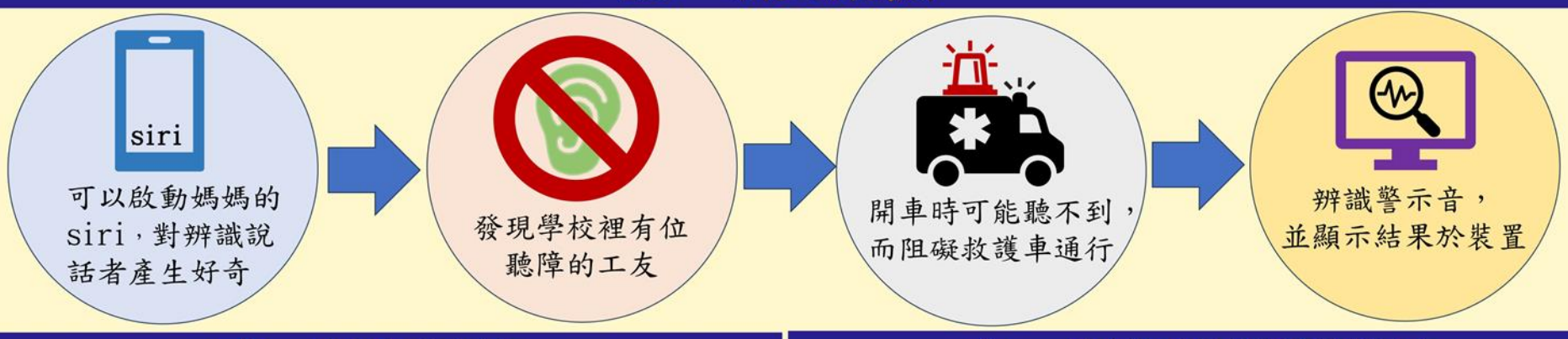
—以傅立葉轉換應用聲紋辨識及判讀警示音訊息

摘要

沒注意到警報聲往往使人錯過黃金逃命時間，對聽障者或聽力衰退的老人而言，更是不容忽視的問題。本研究嘗試編寫辨識說話者程式，先用人聲找出辨識聲音特徵的方法，以傅立葉轉換得到音檔頻率，再用兩種方法擷取特徵來辨識說話者，得知不同參數對準確率的影響，進而實際應用於辨識警示音。

結果顯示，取出越多頻率做為特徵對準確率沒有明顯影響。將音檔切為小段分析，對準確率影響較明顯，且每個小段音檔長0.02秒到0.05秒有最高的準確率。找出合適的變因後，在辨識警示音上準確率達100%，並以Arduino連接LED與LCD螢幕，讓聽障者看得見聲音。

壹、研究動機



貳、研究目的

1. 以 Python建構出錄音、分析、比對的程式。
2. 以「**取出的頻率數目**」、「**小段音檔的長度**」，探討**對辨識說話者準確率的影響**。
3. 應用辨識說話者程式**判讀警示音**
4. 將辨識警示音的結果以Arduino的**LED、LCD顯示**。

參、研究設備與器材

表1 研究設備表

設備	筆記型電腦、錄音語詞卡、Arduino UNO版、LED燈、LCD顯示裝置、杜邦線
軟體	anaconda、Arduino、Tinkercad

肆、研究過程與方法

一、名詞釋義：

傅立葉轉換：

將一個函數寫成傅立葉級數的過程。如果將聲音做傅立葉轉換就可得知聲音「**由哪些單純波組成**」「**單純波的頻率是多高、強度多大**」。

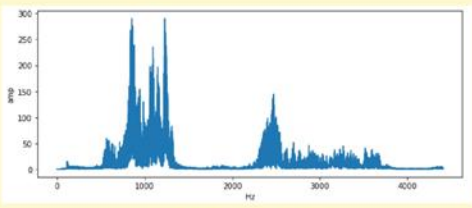


圖1 頻譜圖

二、研究架構圖：

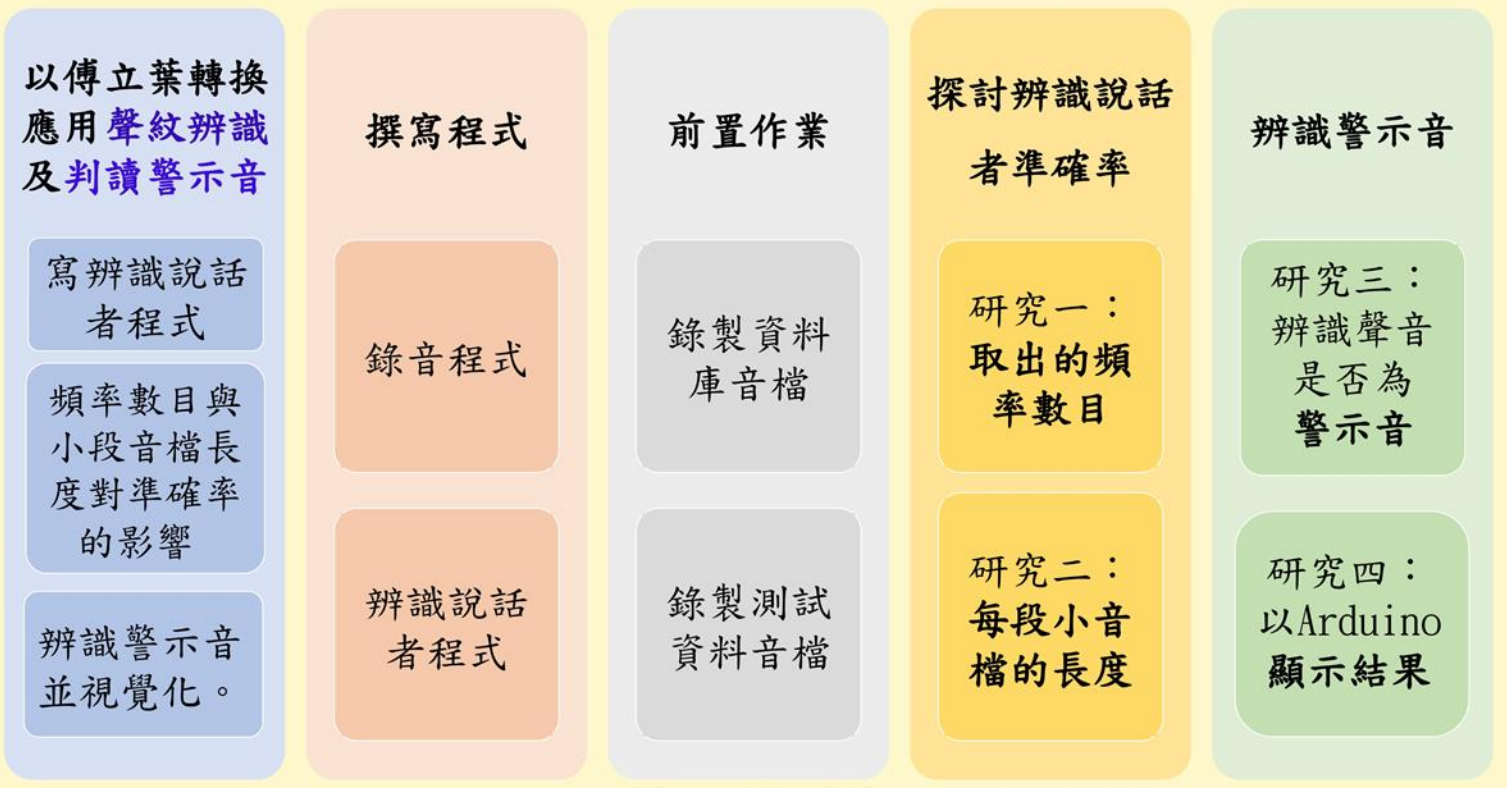


圖2 研究架構圖

三、錄製資料庫及測試資料音檔：

本研究要錄製**人聲**與**警示音**兩種音檔，兩種都要建立資料庫及測試資料的音檔，錄製流程如下：

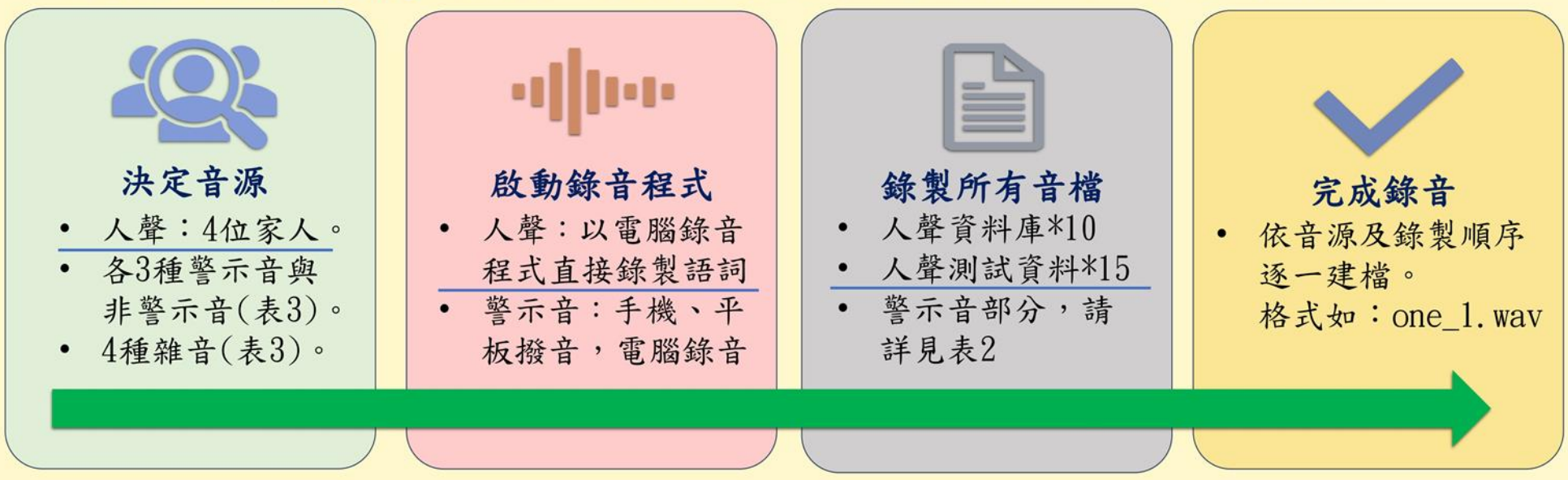


圖3 錄音流程圖

表2 警示音音檔錄製種類

聲音種類	資料庫	沒雜音的測試資料	有雜音的測試資料（雜音音量**小於**警示音）	有雜音的測試資料（雜音音量**大於**警示音）
錄製聲音	警示音 非警示音	警示音 非警示音	警示音	警示音

人聲、警示音的聲音總筆數：100筆 ＋ 247筆 ＝ 347筆以上

表3 詳細音種

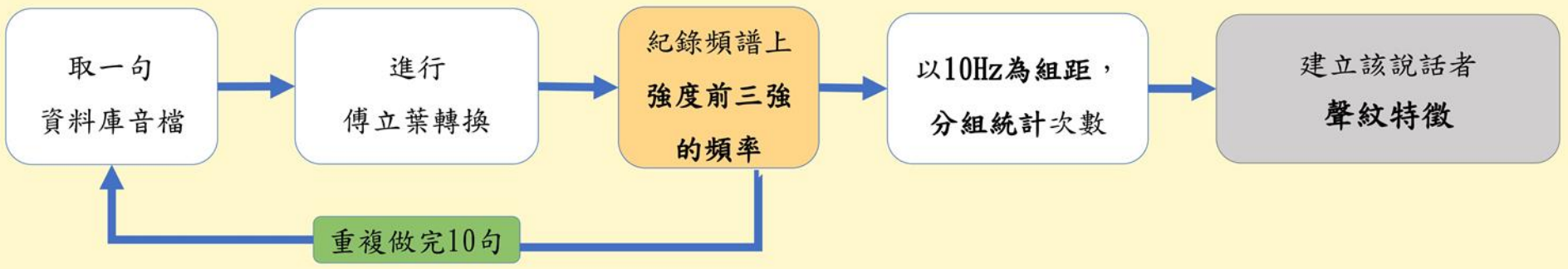
警示音
救護車、警車、火災警報
非警示音
喇叭、電風扇、引擎
雜音音源
街上的聲音、車站、賣場、音樂

四、用 Python寫出辨識說話者的程式，再進行研究：

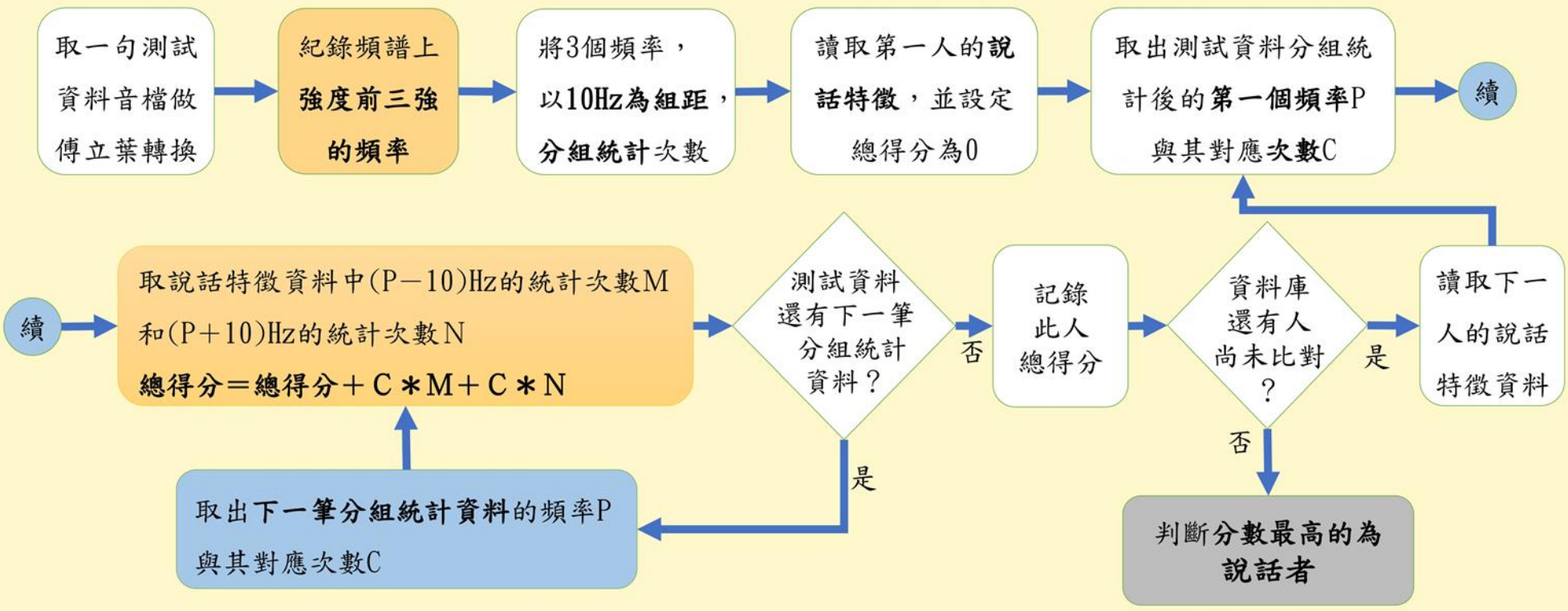
表4 說話特徵舉例表

頻率Hz	10	20	30	40
次數(次)	2	3	0	2

(一)如何建立「說話特徵資料庫」



(二)如何辨識一句測試資料的說話者



五、研究設計：

研究一：探討改變取出的頻率數量對準確率的影響

研究二：將音檔切為小段再分析特徵，探討改變小段的長度對準確率的影響

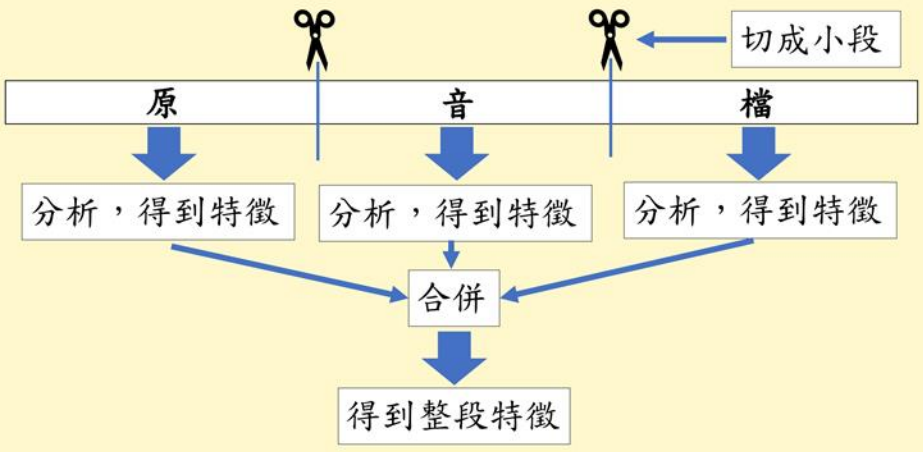
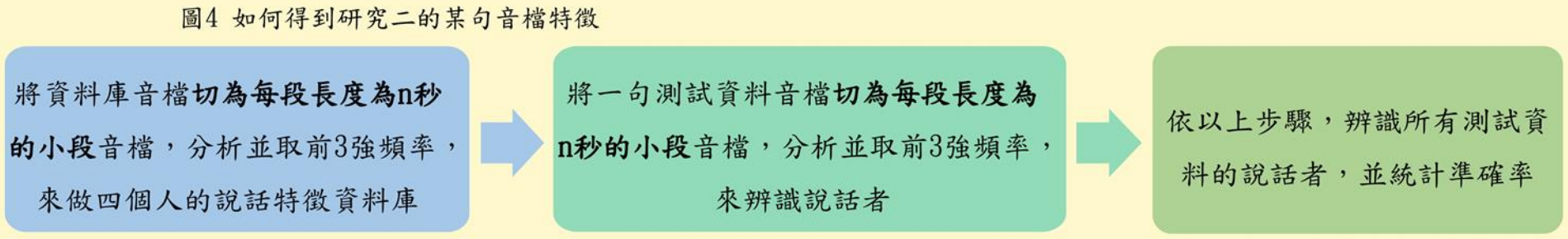


圖4 如何得到研究二的某句音檔特徵



研究三：以研究二的方法，辨識周遭的聲音是否為警示音

本研究以取出**前3強**頻率、小段音檔長度為**0.05秒**，來**分析資料庫及測試資料音檔**，因為分析的方式與研究二一樣，所以只解釋**辨識正確的條件**。

表5 建立音檔流程

救護車、警車、火災警報器(有/無 雜音皆一樣)	辨識成該警示音種類
喇叭、電風扇、引擎	辨識成三個任一皆可

研究四：將警示音辨識結果以 Arduino 顯示

圖5 建立音檔流程

伍、研究結果與討論

研究一：改變取出的頻率數目實驗分析結果

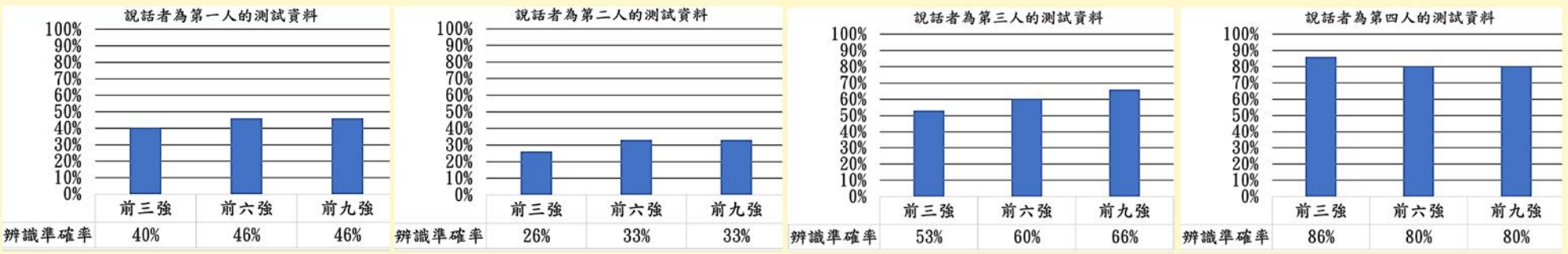


圖6 改變取出的頻率數目結果圖

表6 改變取出的頻率數目準確率統整表

取出的頻率數目	前三強	前六強	前九強
平均準確率	約51%	約55%	約56%

結果討論一：

1. 若取前三強準確率較低時，增加取出的頻率數目後，**準確率會上升**。
2. 若取前三強準確率較高時，增加取出的頻率數目後，**並無改善**。

研究二：改變切為小段後每段小音檔的長度結果

圖7 小段音檔長度準確率分析結果圖

表7 小段音檔長度準確率統整表

小段音檔長度	整段	0.5秒	0.1秒	0.05秒	0.02秒
平均準確率	約51%	約58%	約61%	約75%	約70%

結果討論二：

1. 若整段分析準確率較低時，將小段音檔的長度變短，再分析，**準確率會明顯上升**。
2. 若整段分析準確率較高時，將小段音檔的長度變短，再分析，**並無改善**。

想讓準確率變高的話，可增加取出的頻率數目，並將小段音檔設為0.05秒和0.02秒兩個長度，並**都比對出說話者後，判斷分數較高的結果為說話者**。如表7

表8 取分數高的那個人做為辨識結果

小段音檔長度	0.05秒	0.02秒
說話者	第一人	第二人
說話者的得分	700分	900分
判斷是哪一人？		V

研究三：辨識周遭的聲音，是否為警示音結果

表9 辨識警示音的結果表

雜音音量小於警示音的辨識準確率			
測試資料音檔	救護車	警車	火災警報器
辨識準確率	81%	90%	95%

雜音音量大於警示音的辨識準確率			
測試資料音檔	救護車	警車	火災警報器
辨識準確率	65%	70%	60%

結果討論三：

1. 當聲音為**背景雜音音量小的警示音時**，準確率**約在81%～95%**之間。
2. 當聲音為**背景雜音音量大於警示音時**，準確率**約在60%～70%**之間。

研究四：將警示音辨識結果以Arduino顯示

辨識結果為救護車：

辨識結果為引擎：

圖8 研究四的結果照片

結果討論四：

不論聲音是否為警示音，皆可**成功亮燈、顯示**。

陸、結論

1. 「取出的頻率數目」並不是影響準確率的主要變因，但**增加取出的頻率數目，可以讓準確率微微增加**。
2. 「切為小段後每個小音檔的長度」在切到**0.05秒、0.02秒的時候，是各組中準確率最高的**，所以可以依建議的方法，選擇要取哪個人做辨識結果，使得準確率更高。
3. 當**背景雜音音量小於警示音**時，各種警示音辨識準確率約為**81%～95%**之間。

   當**背景雜音音量大於警示音**時，因為做傅立葉轉換後頻譜上雜音頻率強度較強，所以會**取到雜音頻率作為特徵**，導致準確率下較到約在**60%～70%**之間。

柒、參考文獻資料

1. 洪錦魁（2019）。Python最強入門邁向頂尖高手之路王者歸來。深智數位。
2. 趙英傑（2020）。超圖解Arduino互動設計入門。旗標科技。
3. 日本Newton Press(2020)。三角函數sin、cos、tan。人人出版。
4. 作者本人。2022。新北市 110 學年度中小學科學展覽會。神奇的傅立葉你到底是誰。