國立中山大學機械與機電工程學系

隨機過程與機率

期末報告

利用 FastICA 分析圖片與音檔之應用

詹恆瑜

B083022001

民國 112 年 6 月 10 日

Abstract

在現實情況中在這個過程中，我們會遇到許許多多的問題，例如麥克風之間的時間延遲、迴聲、音量差異等等，我試著以透過 FactICA 的方式分離非高斯成份的混合圖片和混合聲音，試圖解決在各場合或各種因素導致所要元素與其他元素混合的情況，在聲音的實驗中我加入三種純人聲以及兩種人聲混合一種有背景音樂的樂聲來做測試，並試圖修改程式碼增加最大疊代次數使得效果更好，圖片則是使用三種不同的非高斯圖照片來做實驗，最後的成果會有黑白負片的效果。

Ⅰ. Introduction

獨立成分分析（Independent Component Analysis，ICA）是機器學習中是種時間都式的學習方法，可以將多變輛信號分離為可加性子分量，且他假設這些子分量都為非高斯訊號，是現在系號處理中處理盲源分離常見的方法，現在最常用的方法為 Infomax 及 FactICA 方法。

Infomax，它基於最大概然估計或最小化互信息的方法進行開發。另一種稱為 FastICA，它通過在固定點迭代過程中最大化用“負熵”近似衡量的非高斯性來進行開發。

在 ICA 的基本假設中被考量的來源是統計獨立的，且基本假設有三個，其中第一個假設是 ICA 的基礎。正如前面討論的那樣，統計獨立是能夠從觀察值 xi(t)估計獨立分量 sˆ(t)的關鍵特徵。

第二點，獨立分量具非高斯分量，因為高斯性和獨立性之間存在著緊密的聯繫。使用 ICA 框架無法分離高斯源，因為兩個或多個高斯隨機變量的和本體也是高斯的。也就是說，在 ICA 框架中，高斯源的和與單個高斯源無法區分開，因此禁止停止使用高斯源。

第三點，混合矩陣是可逆的，如果混合矩陣不可逆，那麼我們尋求估計的解混合矩陣根本不存在。

混合過程的一個基礎方面是傳感器必須空間上分離，以方便每個傳感器記錄不同的源信號混合。我們可以將混合過程序構建模擬為矩陣乘法，如下所示：

x(t) = As(t)

其中 A 是一個未知矩陣，稱為混合矩陣，x(t)和 s(t)分別是表示觀察信號和源信號的兩個向量。將這種信號處理技術描述為盲處理的原因是我們對混合矩陣，甚至對源信息號本身都沒有任何信息。目標是僅通過觀察量向 xi(t)恢復原始信號 si(t)。我們通過首先獲得解混合矩陣 W 的估計，其中 W = Aˆ(-1)，獲得得源信號的估計 s(t)：

s(t) = Wx(t)

ICA 的基礎模型是源信號通過線性基變和混合過程序生成觀測信號。假設有 N 個獨立的源信號 si(t)，通過未知的混合陣 A 與觀察測信號 xi(t)進入行線性組合。其中 ICA 所要求的獨立性就和隨機過程定理有密切的關係，是隨機過程中

重要的概念之一，利用此特性來估計混合矩陣與解混合矩陣來找出原始的獨立訊號。

Ⅱ. Methodology

ICA 的理論： ICA 是一種統計信息處理技術，用於將混合信息分解為獨立成分，這些分解在統計上是相互獨立的。它設置觀察到的混合合信號是由若干相互立的源信號通過線路性混合得到的。 ICA 的目標是通過找到一組濾波器，使通過濾波後的信號盡可能地可能相互獨立。 ICA 的算法（FastICA）： FastICA 是 ICA 的一種常用算法。它基於最高峰度（非高斯性）的原理，通過代優化來估計混合矩陣的逆矩陣，以分離混合信息中的獨立成分。其中我有優化過程式碼透過增加 max_iter 和 tol 參數來控制演算法的最大疊代次數和收斂容差，如此來提高演算法的收斂性，進而使訊號分析得更理想。

III. Data and Experiments

一、分析來自不同地方的聲音

我將我拍攝的三張不同非高斯照片放入寫好的程式碼中，其中把我的三張照片做一連串包括轉換為矩陣及灰階等等方式進而達到混合後，再把混合的圖做 FactICA 分析來看效果。

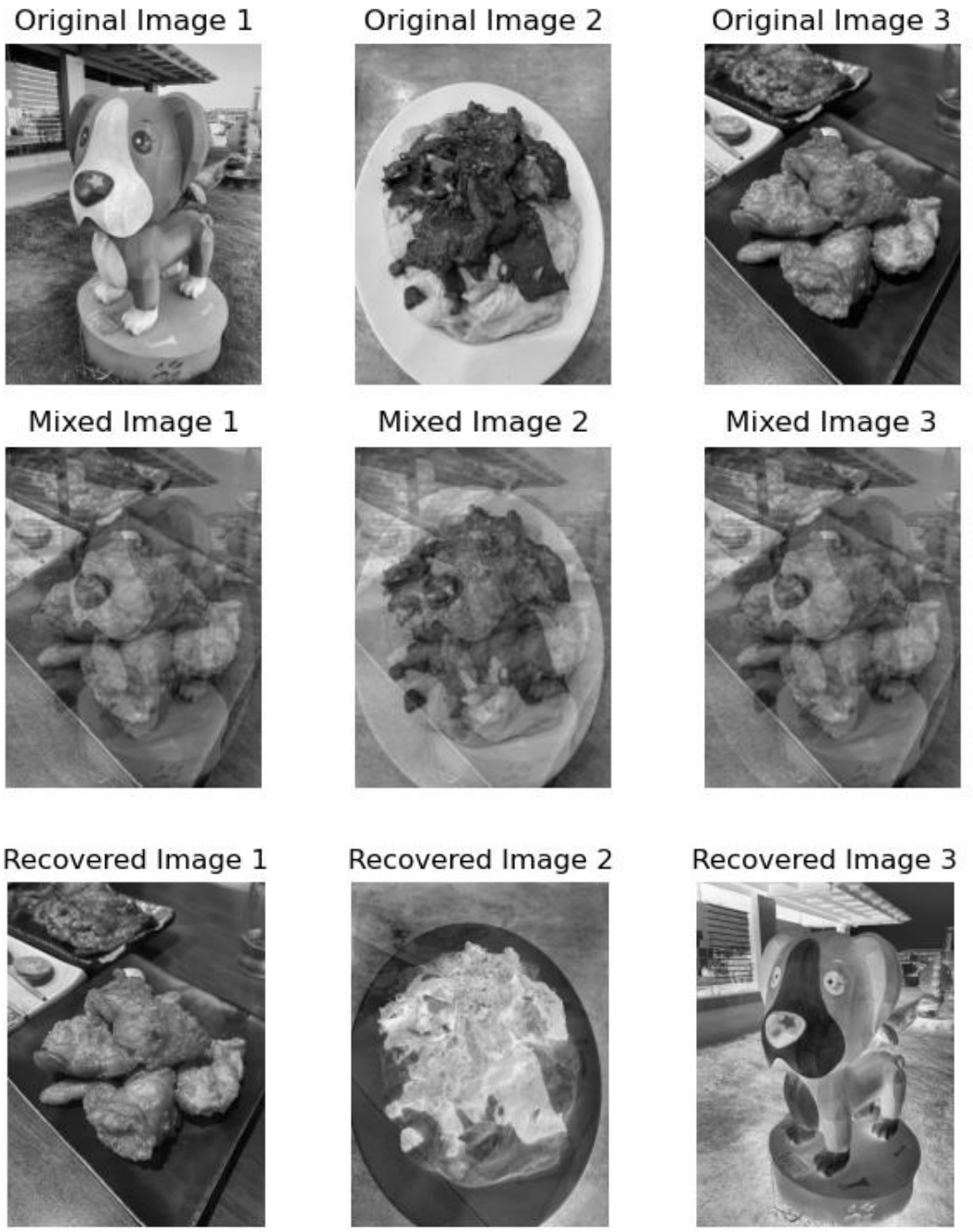
以下是我來分析我三張非高斯圖的圖形分析的 Python 程式碼:

```
from PIL import Image
import numpy as np
from sklearn.decomposition import FastICA
import matplotlib.pyplot as plt

image1 = Image.open("image1.jpg").convert("L")
image2 = Image.open("image2.jpg").convert("L")
image3 = Image.open("image3.jpg").convert("L")

image1_array = np.array(image1)
image2_array = np.array(image2)
image3_array = np.array(image3)

image1_normalized = image1_array / 255.0
image2_normalized = image2_array / 255.0
image3_normalized = image3_array / 255.0

image1_flattened = image1_normalized.reshape(-1)
image2_flattened = image2_normalized.reshape(-1)
```

```
image3_flattened = image3_normalized.reshape(-1)

mixing_matrix = np.array([[1, 0.5, 1.5], [1, 2, 1], [1.5, 1, 2]])
mixed_images = np.dot(mixing_matrix, np.array([image1_flattened,
image2_flattened, image3_flattened]))

mixed_image1 = mixed_images[0].reshape(image1_array.shape)
mixed_image2 = mixed_images[1].reshape(image2_array.shape)
mixed_image3 = mixed_images[2].reshape(image3_array.shape)

ica = FastICA(n_components=3)
recovered_images = ica.fit_transform(mixed_images.T).T

recovered_image1 = recovered_images[0].reshape(image1_array.shape)
recovered_image2 = recovered_images[1].reshape(image2_array.shape)
recovered_image3 = recovered_images[2].reshape(image3_array.shape)

plt.figure()
plt.subplot(2, 3, 1)
plt.imshow(image1_array, cmap="gray")
plt.title("Original Image 1")
plt.axis("off")

plt.subplot(2, 3, 2)
plt.imshow(image2_array, cmap="gray")
plt.title("Original Image 2")
plt.axis("off")

plt.subplot(2, 3, 3)
plt.imshow(image3_array, cmap="gray")
plt.title("Original Image 3")
plt.axis("off")

plt.subplot(2, 3, 4)
plt.imshow(mixed_image1, cmap="gray")
plt.title("Mixed Image 1")
plt.axis("off")
```

```
plt.subplot(2, 3, 5)
plt.imshow(mixed_image2, cmap="gray")
plt.title("Mixed Image 2")
plt.axis("off")

plt.subplot(2, 3, 6)
plt.imshow(mixed_image3, cmap="gray")
plt.title("Mixed Image 3")
plt.axis("off")

plt.tight_layout()
plt.show()

plt.figure()
plt.subplot(2, 3, 1)
plt.imshow(recovered_image1, cmap="gray")
plt.title("Recovered Image 1")
plt.axis("off")

plt.subplot(2, 3, 2)
plt.imshow(recovered_image2, cmap="gray")
plt.title("Recovered Image 2")
plt.axis("off")

plt.subplot(2, 3, 3)
plt.imshow(recovered_image3, cmap="gray")
plt.title("Recovered Image 3")
plt.axis("off")

plt.tight_layout()
plt.show()
```

實驗結果:



二、分析混合了三種不同聲音的分析程式碼為:

我將我錄製好的聲音 MP4 檔放入寫好的程式碼中，我這邊做了兩種不同的小實驗去測試演算法的能力，並做演算法的優化以達到更好的效果，我實驗一是放入三種不同的純人聲，三者都沒有背景音樂，實驗二則是放入兩種純人聲和一個有背景音樂的人聲，兩個實驗都是將錄製好的三個音檔放入後，經過一年串證規化等程序進行分析，最後呈現出結果。

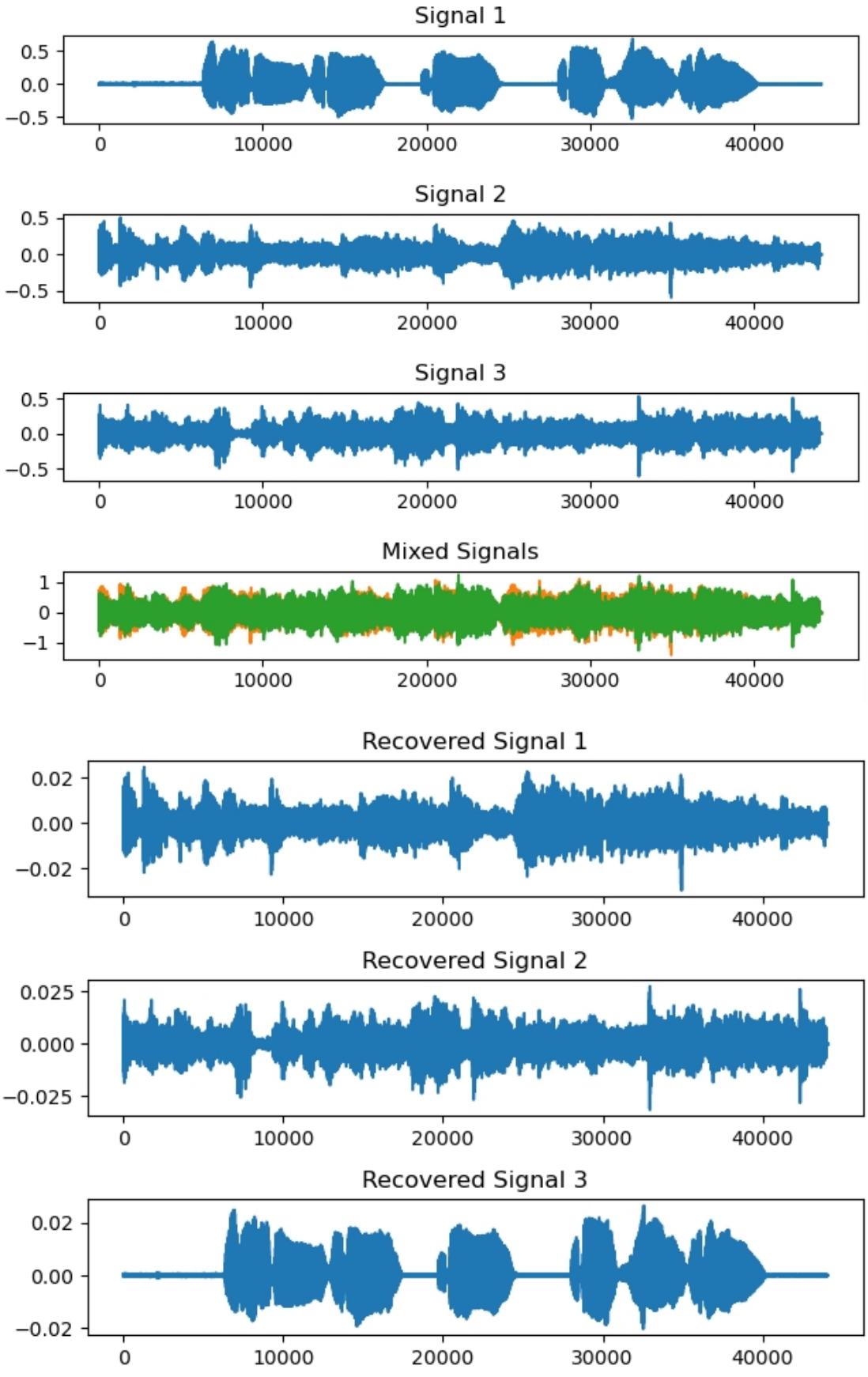
```
from moviepy.editor import AudioFileClip
from scipy import signal
```

```
import numpy as np
from sklearn.decomposition import FastICA
import matplotlib.pyplot as plt

def load_audio(file_path):
    clip = AudioFileClip(file_path)
    audio = clip.to_soundarray()[:, 0]
    clip.close()
    return audio

file_path1 = 'signal1.mp4'
file_path2 = 'signal2.mp4'
file_path3 = 'signal3.mp4'

signal1 = load_audio(file_path1)
signal2 = load_audio(file_path2)
signal3 = load_audio(file_path3)

target_sample_rate = 44100
signal1 = signal.resample(signal1, int(signal1.shape[0] *
target_sample_rate / signal1.shape[0]))
signal2 = signal.resample(signal2, int(signal2.shape[0] *
target_sample_rate / signal2.shape[0]))
signal3 = signal.resample(signal3, int(signal3.shape[0] *
target_sample_rate / signal3.shape[0]))

min_length = min(len(signal1), len(signal2), len(signal3))
signal1 = signal1[:min_length]
signal2 = signal2[:min_length]
signal3 = signal3[:min_length]

mixing_matrix = np.array([[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]])
mixed_signals = np.dot(np.array([signal1, signal2, signal3]).T,
mixing_matrix)

ica = FastICA(n_components=3, max_iter=1000, tol=1e-4)
recovered_signals = ica.fit_transform(mixed_signals)
```

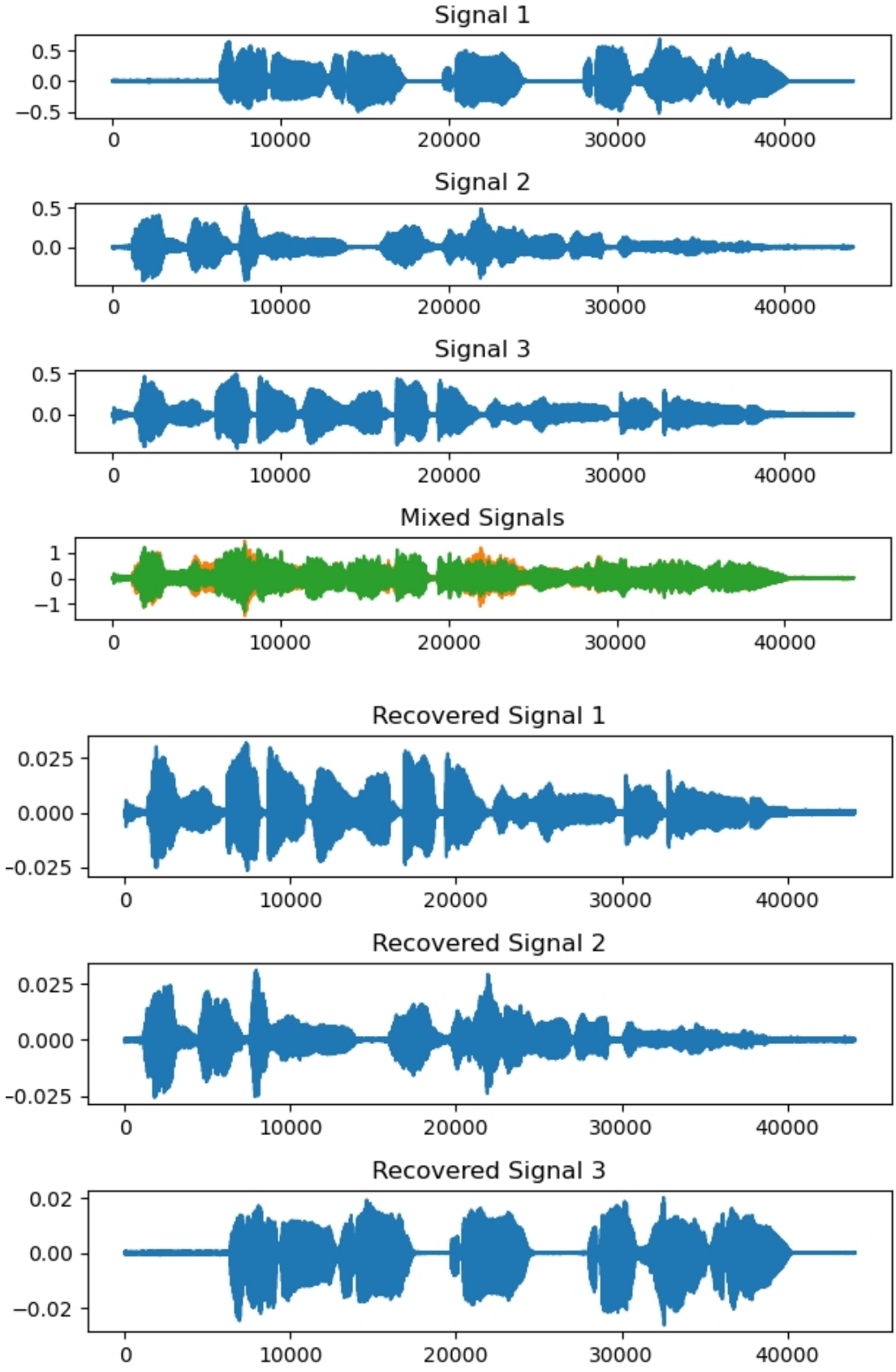
```
plt.figure()

plt.subplot(4, 1, 1)
plt.title("Signal 1")
plt.plot(signal1)

plt.subplot(4, 1, 2)
plt.title("Signal 2")
plt.plot(signal2)

plt.subplot(4, 1, 3)
plt.title("Signal 3")
plt.plot(signal3)

plt.subplot(4, 1, 4)
plt.title("Mixed Signals")
plt.plot(mixed_signals)

plt.tight_layout()
plt.show()

plt.figure()

plt.subplot(3, 1, 1)
plt.title("Recovered Signal 1")
plt.plot(recovered_signals[:, 0])

plt.subplot(3, 1, 2)
plt.title("Recovered Signal 2")
plt.plot(recovered_signals[:, 1])

plt.subplot(3, 1, 3)
plt.title("Recovered Signal 3")
plt.plot(recovered_signals[:, 2])

plt.tight_layout()
plt.show()
```

實驗結果:

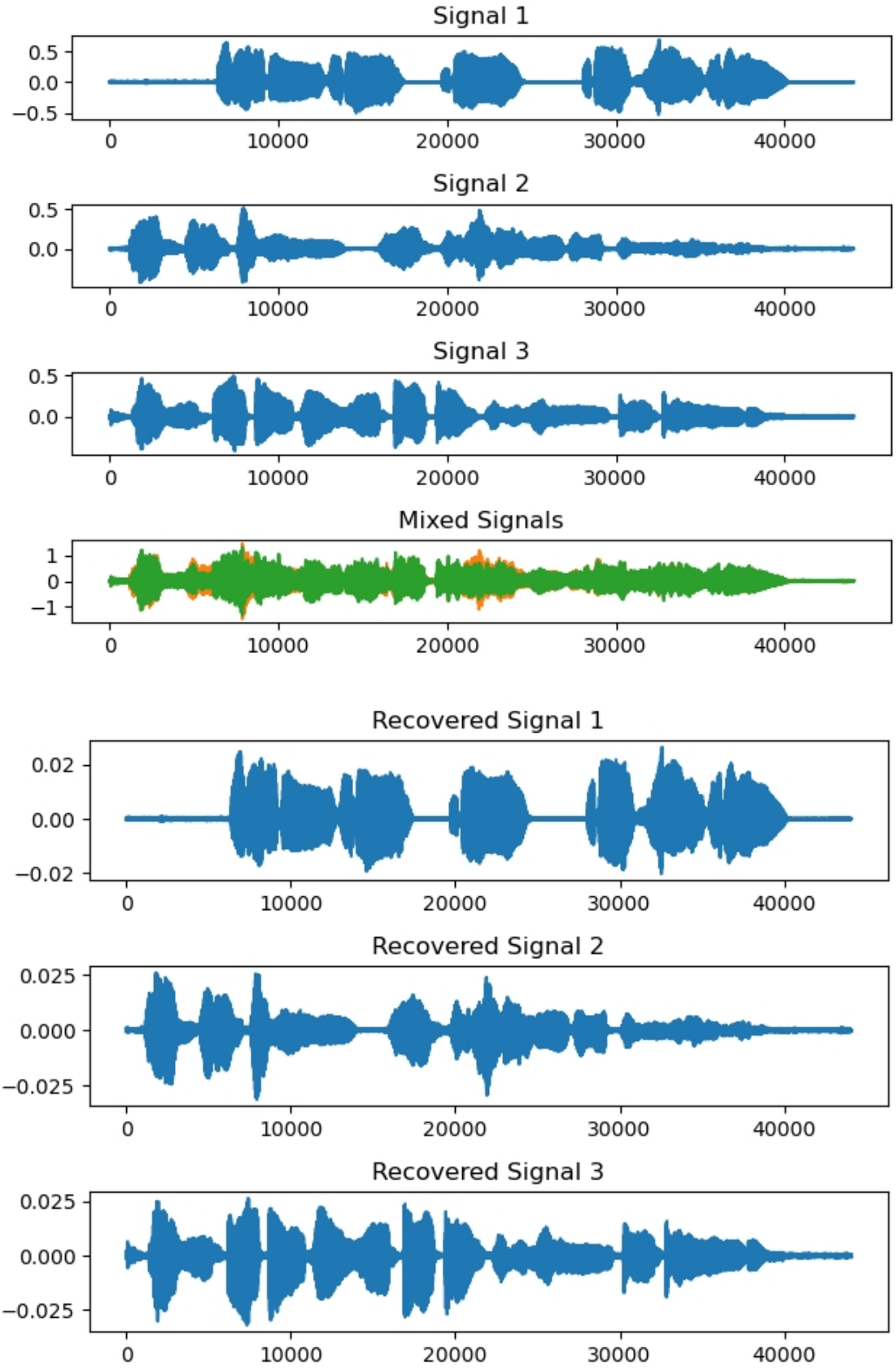
A. 兩種不同的人聲與一個有背景音樂的人聲混合



B. 三種不同的人聲混合並做 FastICA 分析



C. 三種不同的人聲混合並在 FastICA 中增加疊代次數和減少容差



IV. Observations and discussions:

從 FastICA 演算法來分析三張照片的實驗中，我將原圖三張混合成三張不同的

圖後，再以 FastICA 的方式做分析得到圖雖然形狀輪廓與原圖是一樣的，但是顏色方面會與原圖有差異，呈現像負片的那種效果，經查詢後 FastICA 算法無法確地獨立成份的正負號，因而把原始成份的相素質的正負號相反了，因為在 FastICA 演算法中的假設為使用非高斯的信號，再圖像處理中是以相素值的統計特性展現，而亮度值一般呈現高斯分布，只要獨立成份的正負號相反了就會導致此狀況。

在 FastICA 演算法來分析三種不同的人聲的實驗中，可以看出在放入三個純人聲的 MP4 檔分析的成果比放入兩個純人聲檔與一個有背景音樂的人聲混合的分析效果來得好一些，之後我為了想讓 FastICA 的效果更理想，我試著增加程式碼中疊代的次數及減少容差，可以看得出表現又更加好了，我想再不斷調整就可以把效果再提高一個層級。

V. Conclusion

在 FastICA 中必須使用非高斯的信號源來做分析，如果使用高斯信號來做分析的話，混合後的信號無法得到混合前德有效邊緣信息，只可以接受多個信號混合一個高斯分布信號，各個信號中是統計獨立的，經過線性疊加之後再透過解混合矩陣去分析出原信號，在分析出後的成分順序是無法確定的，透過 FastICA 可以把我們所需的信號從混合信號中分離出來，讓我們能對個別獨立的原信號進行分析及應用。

VI. Reference

- https://docs.google.com/document/d/1r4ELZKnNrJM9MtWQ7nx0kjlfD0WO7FQx/edit
- https://drive.google.com/drive/folders/1fmk3kF7AvG1bujW2Or3f7czGb-GRRLbE
- https://drive.google.com/drive/folders/1fmk3kF7AvG1bujW2Or3f7czGb-GRRLbE
- https://drive.google.com/drive/folders/1fmk3kF7AvG1bujW2Or3f7czGb-GRRLbE
- https://drive.google.com/drive/folders/1fmk3kF7AvG1bujW2Or3f7czGb-GRRLbE
- https://drive.google.com/drive/folders/1fmk3kF7AvG1bujW2Or3f7czGb-GRRLbE
- https://www.easyatm.com.tw/wiki/FastICA
- https://drive.google.com/drive/folders/1fmk3kF7AvG1bujW2Or3f7czGb-GRRLbE