

國立台灣海洋大學資訊工程學系專題報告

神鬼狩怪者 ZERO：基於 Android 行動裝置

與肌肉感測器之擴增實境人機互動系統



00457036 資工 4A 孫韻涵  
00457141 資工 4B 周希珉

報告編號: NTOUCSE 107 學年度-指導老師編號-競賽組第 05  
組 指導教授: 張欽圳老師

中華民國 107 年 12 月 14 日

## 摘要

本專題使用手持行動裝置與肌肉感測器開發一個擴增實境遊戲。本遊戲應用手持裝置的相機來觀察環境，以電腦視覺技術決定虛擬怪獸在環境中的位置，讓遊戲者必須透過改變手持裝置的相機視角或位置來找到虛擬怪獸。並根據肌肉感測器測量遊戲者肌肉活動狀況，讓遊戲者可以透過不斷改變肌肉施力大小與頻率來消滅虛擬怪獸。本遊戲不但可以讓遊戲者四處走動，也可以讓遊戲者達到活動筋骨、鍛鍊肌肉的目的。本遊戲另外設計不同關卡，包含施力大小與施力頻率，以增加遊戲豐富度。

Table 1: 分工表

姓名	工作內容
孫韻涵	感測器讀取、數值分析、能量條製圖、手機藍芽、系統整合、報告文件
周希珉	場景分析、背景偵測、怪獸縫合場景、系統整合、報告文件

# Contents

<b>1</b>	<b>介紹</b>	<b>1</b>
1.1	動機與目的 . . . . .	1
1.2	系統簡介 . . . . .	1
1.3	報告組織 . . . . .	2
<b>2</b>	<b>軟體與硬體架構</b>	<b>3</b>
2.1	軟硬體架構圖 . . . . .	3
2.2	硬體架構 . . . . .	4
2.2.1	系統外觀介紹 . . . . .	4
2.2.2	感測技術 . . . . .	5
2.3	軟體架構 . . . . .	7
2.3.1	藍芽功能 . . . . .	7
2.3.2	Arduino 端功能 . . . . .	8
2.3.3	遊戲端功能 . . . . .	9
<b>3</b>	<b>場景分析與肌肉感測訊號分析</b>	<b>12</b>
3.1	肌肉感測器訊號分析 . . . . .	12
3.2	場景分析 . . . . .	16
<b>4</b>	<b>實驗結果</b>	<b>21</b>
4.1	實驗環境 . . . . .	21
4.2	肌肉感測器參數測量 . . . . .	21
4.3	場景分析效能評估 . . . . .	23
4.4	專題結果展示 . . . . .	25
4.5	使用者體驗 . . . . .	27
<b>5</b>	<b>討論與結論</b>	<b>29</b>

# List of Figures

1.1 裝置使用之呈現 . . . . .	2
2.1 軟硬體架構圖 . . . . .	3
2.2 操作使用圖 . . . . .	4
2.3 遊戲裝置之外觀 . . . . .	5
2.4 Arduino 肌肉感測器使用之硬體裝置 . . . . .	5
2.5 感測器流程圖 . . . . .	6
2.6 藍芽開啟流程圖 . . . . .	6
2.7 藍芽連線程式碼片段 . . . . .	7
2.8 藍芽接收程式碼片段 . . . . .	8
2.9 Arduino 程式碼片段 . . . . .	8
2.10 肌肉感測器回傳數值 . . . . .	9
2.11 主遊戲操作流程 . . . . .	10
2.12 挑戰模式流程圖 . . . . .	11
3.1 陣列與訊號關係示意圖 . . . . .	12
3.2 設兩個分別為大跟小的陣列 . . . . .	13
3.3 開始執行的片段 . . . . .	13
3.4 感測訊號轉換 . . . . .	14
3.5 快速傅立葉程式碼片段 . . . . .	15
3.6 速度的分級 . . . . .	16
3.7 畫面加速程式碼片段 . . . . .	17
3.8 特徵點數量足夠與否 . . . . .	17
3.9 條件不好的 Homography . . . . .	18
3.10 條件好的 Homography . . . . .	19
3.11 Homography 運算公式 . . . . .	19
3.12 運算結果說明 . . . . .	20
4.1 遊戲場景 . . . . .	23
4.2 怪獸呈現 . . . . .	24
4.3 手持裝置的翻轉 . . . . .	25
4.4 遊戲流程畫面一 . . . . .	26

4.5 遊戲流程畫面二 . . . . .	26
4.6 遊戲流程畫面三 . . . . .	27

# List of Tables

1	分工表	I
4.1	感測器測量數據表	22
4.2	感測器手臂測量表	22
4.3	感測器姿勢測量表	22
4.4	玩家評分表	28

# Chapter 1

## 介紹

### 1.1 動機與目的

現代的人們常常久坐或長期維持同一個姿勢，導致肌肉痠痛或引發各種疾病。然而，人們逐漸發現運動的重要性，也開始養成每個禮拜都必須運動幾天的習慣。但是，卻不是每個人都有時間上健身房或是去公園附近運動。此外，還有一些人排斥乏味的運動器材，例如：跑步機、飛輪等，甚至是拒絕到戶外運動。為了解決種種因素，我們設計了一款能同時兼具娛樂及運動的遊戲，不僅能讓使用者隨時隨地想玩就玩，在遊戲的過程中還能同時發揮到讓肌肉放鬆的效果，並能從其中得到成就感。我們開發了一款手機遊戲，使用 Android 行動裝置、Arduino 及肌肉感測器帶來遊戲前所未有的趣味性。其中加入了擴增實境的遊戲畫面，讓使用者在遊戲中增加實際感；另外，我們還提供了一個練習模式，在一般情況下也能鍛鍊手臂肌肉。

### 1.2 系統簡介

本系統所提供之功能主要包含透過擴增實境使玩家與鏡頭前的實際場景融合，怪獸隨著場景視角改變，並利用手臂上的肌肉感測器測量施力的大小，隨著肌肉放鬆緊繃，增加遊戲的難易度，使遊戲更具趣味性。

首先，我們進入遊戲畫面尋找怪獸藏匿點，利用電腦視覺的技術判斷此場景的獨特性，假如場景夠複雜，怪獸出現。為了使手機移開原場景再移回來，怪獸出現在原位置，我們將原畫面與接下來之畫面做視角及距離的比較。當怪獸出現後，螢幕畫面會出現如何攻擊怪獸的提示訊息，玩家按照指令訊息操作肌肉感測器，例如：在時間內的肌肉用力或放鬆。我們除了訊號的大小以外還要得到他的快慢變化，因此利用訊號分析技術，如

快速傅立葉轉換，分析使用者之動作，得到肌肉的收縮節奏。如果操作和要求大致相符，遊戲判斷他是正確操作，便能成功攻擊怪獸。



Figure 1.1: 裝置使用之呈現

### 1.3 報告組織

本實驗報告第二章介紹系統的軟硬體架構，硬體包刮系統外觀與所使用之感測器及感測技術。軟體架構包括系統的流程圖及所有功能，並解釋各個功能的用途、流程與實作方法。第三章將著重介紹軟體演算法之過程。第四章為研究方法與實作期間所做的實驗數據；第五章展示專題之結論。

# Chapter 2

## 軟體與硬體架構

本章節介照系統的軟硬體架構及所使用的感測器之規格與感測技術。

### 2.1 軟硬體架構圖

如圖 2.1所示，系統架構分為兩大項，左半部為硬體，其中包含 Arduino 與肌肉感測器；右半部則為軟體，包含手持裝置與 OpenCV 模組。如圖 2.2所示，感測器端用於偵測肌肉施力大小之數值並回傳至系統；手機端有機器視覺模組用於影像處理，計算與分析感測器回傳之數值；藍芽模組共用於感測器與手機互相溝通。

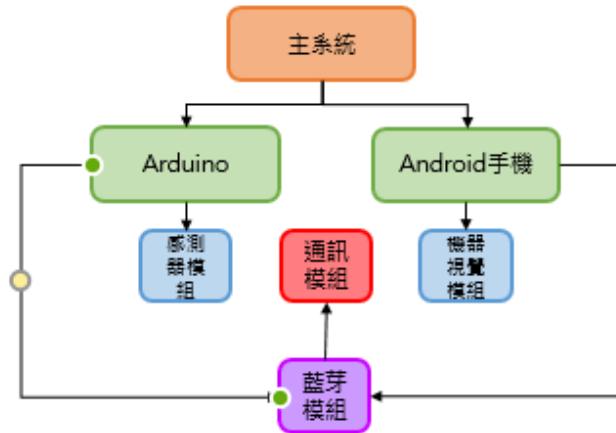


Figure 2.1: 軟硬體架構圖

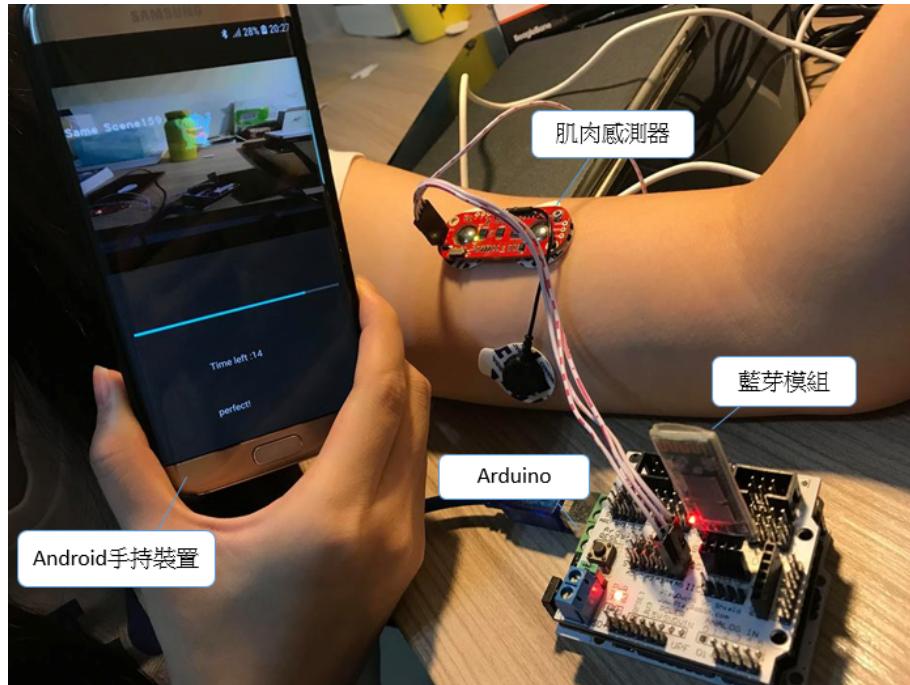


Figure 2.2: 操作使用圖

## 2.2 硬體架構

### 2.2.1 系統外觀介紹

如圖 2.3和圖 2.4所示，本系統外觀分別為藍芽裝置、電源、肌肉感測器、Android 手持裝置四項，介紹如下：

1. 電源：使用電流穩定既可重複使用的行動電源作為供電來源。
2. 藍芽裝置（圖 2.4a）：藍芽模組可以與手持裝置的藍芽互相傳送接收訊息。
3. 肌肉感測器（圖 2.4b）供電電壓： $+3.1V$  to  $+5V$   
RAW EMG 輸出：MyoWare 能擁有二次輸出 RAW EMG 波形的機會。  
LED 指示燈：增加兩個版載 LED 燈，當 MyoWare 的電源鍵是開的，肌肉用力時 LED 燈即會發亮。

4. Android 手持裝置 (圖 2.3b): 本專題使用到手機上的相機、藍芽功能。

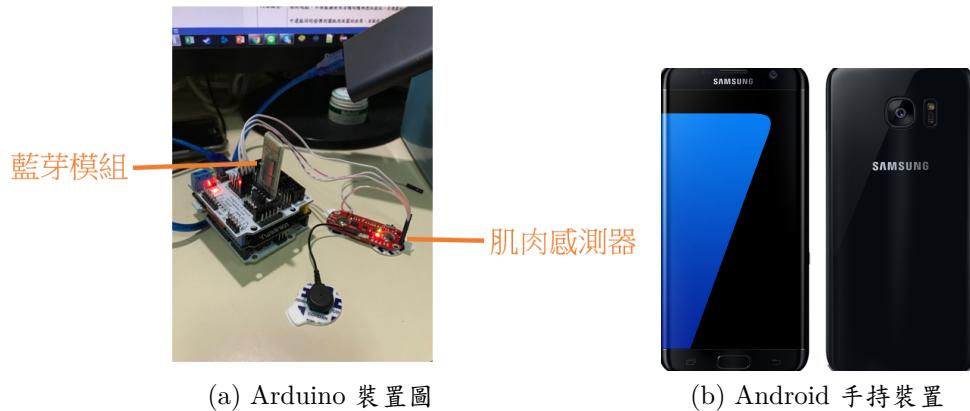


Figure 2.3: 遊戲裝置之外觀

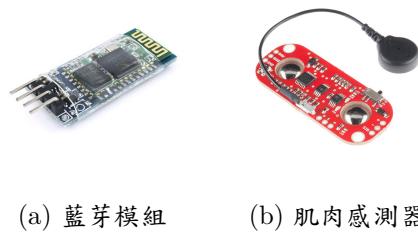


Figure 2.4: Arduino 肌肉感測器使用之硬體裝置

## 2.2.2 感測技術

製作本系統所需主要技術如下:

### 1. 肌肉感測器:

通過 MyoWare 可以測量肌肉的電訊號活動，在過去實驗一般稱之為肌電儀器 (EMG)。當大腦下指令告訴你的肌肉收縮，將發送一個傳遞路徑提醒肌肉開始徵招肌肉運動單元 (肌束纖維使肌肉產生力量)。當肌肉越用力，產生越多的肌肉運動單元來招募更大的肌肉力量。當招募越多的肌肉單元數目，將會產生更多的肌肉電位改變。MyoWare 將分析此電訊號並輸出訊號來表示肌肉如何用力。當你越用力 MyoWare 將輸出更大的電壓。雖然 EMG 已能偵測肌肉放電的訊號，但 MyoWare 則使它更具意義更為人性化。

圖 (2.5) 為感測器訊號傳遞的流程圖，我們以 0.15 秒讀取一次的頻率取得訊號，以 1.2 秒為一個間距分析其中的變化。之後在第三章會對感測器訊號分析的演算法有更詳細地介紹。

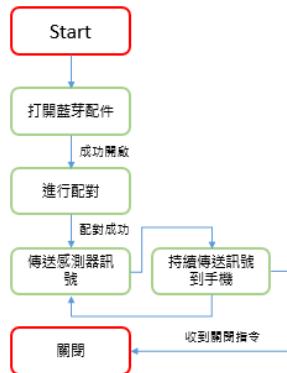


Figure 2.5: 感測器流程圖

2. Android 藍芽連接: 使用無線通訊的概念，藉由藍芽通訊，手持裝置得以與肌肉感測器相互傳遞訊息。  
操作流程:



Figure 2.6: 藍芽開啟流程圖

如圖 2.6 所示，藍芽開啟流程如下:  
流程一: 打開手持裝置藍芽。

流程二：點選「HC-06」進行配對。  
流程三：回到應用程式主畫面，點選藍芽連線。  
流程四：顯示連線成功。

3. Android 相機：我們使用 OpenCV Android SDK 提供的 JavaCameraView 來調用 android 相機。JavaCameraView 繼承 Android 內建 SurfaceView 但必須呼叫 onCameraFrame 來取得每帧相機影像來供後續 OpenCV 程式庫處裡。OpenCV(Open Source Computer Vision Library) 是一套跨平台的電腦視覺式庫，支援圖像處裡、電腦視覺和模式識別的程式開發。在本專題中我們使用 OpenCV 的特徵點偵測及影像縫合功能將怪物於相機畫面上，以達到擴增實境的效果。首先在程式中使用 AKAZE 方法對目標物影像進行特徵點採取。接著再用採集的特徵點與相機影像求取最佳配對的點對。之後在第三章會對如何分析場景進行更詳細地介紹。

## 2.3 軟體架構

軟體部分中包含了藍芽功能、Arduino 端功能以及遊戲端功能，接下來會依序介紹其用途及操作流程。

### 2.3.1 藍芽功能

用途：建立與肌肉感測器之間的通訊。圖 2.7為將藍芽連線至裝置的程式碼片段。圖 2.8為藍芽接收訊號，以 byte 形式儲存，並且將收到的數字存在 session 以便之後的計算

操作流程：

```
//開啟執行緒用於傳輸及接收資料  
mConnectedThread = new ConnectedThread(mBTSocket);  
mConnectedThread.start();  
//開啟新執行緒顯示連接裝置名稱  
mHandler.obtainMessage(CONNECTING_STATUS, 1, -1, name)  
    .sendToTarget();  
}  
}  
}.start();
```

Figure 2.7: 藍芽連線程式碼片段

```

mHandler = new Handler(){
    String readMessage = null;
    int data;
    public void handleMessage(android.os.Message msg){
        if(msg.what == MESSAGE_READ){ //收到MESSAGE_READ 開始接收資料
            try {
                readMessage = new String((byte[]) msg.obj, "UTF-8");
                //過濾字串中的所有非數字字元
                readMessage = readMessage.replaceAll("[^(\u00e0-\u00f9)]", "");
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
            Session session = Session.getSession();
            session.put("data", readMessage);
        }
    }
}

```

Figure 2.8: 藍芽接收程式碼片段

### 2.3.2 Arduino 端功能

將肌肉感測器輸出線路插在指定腳位，之後將貼片貼在手臂的指定位置上。圖 2.9為將感測器訊號由 Arduino 端傳至 Android 手機端。如圖 2.10所示，在手機端能清楚讀入感測器傳入的數值。

```

int analogPin = 5;
int val = 0;

void setup() {
    // put your setup code here, to run once:
    Serial.begin(9600);
}

void loop() {
    // put your main code here, to run repeatedly:
    val = analogRead(analogPin);
    Serial.println(val);
    delay(500);
}

```

Figure 2.9: Arduino 程式碼片段

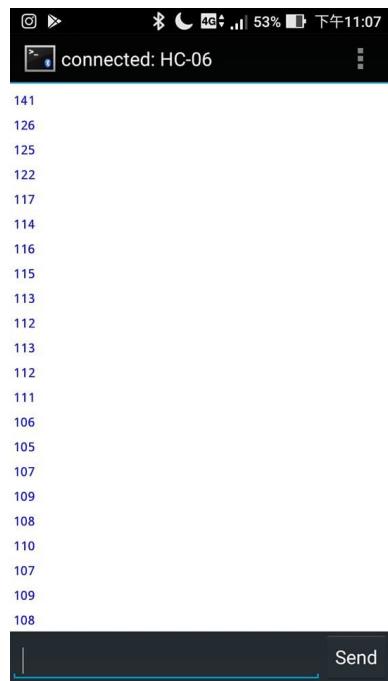


Figure 2.10: 肌肉感測器回傳數值

### 2.3.3 遊戲端功能

#### (1) 狩怪模式

操作流程：玩家開啟鏡頭，並開始尋找怪獸躲藏之處，找到怪獸後已指定的方式進行攻擊。

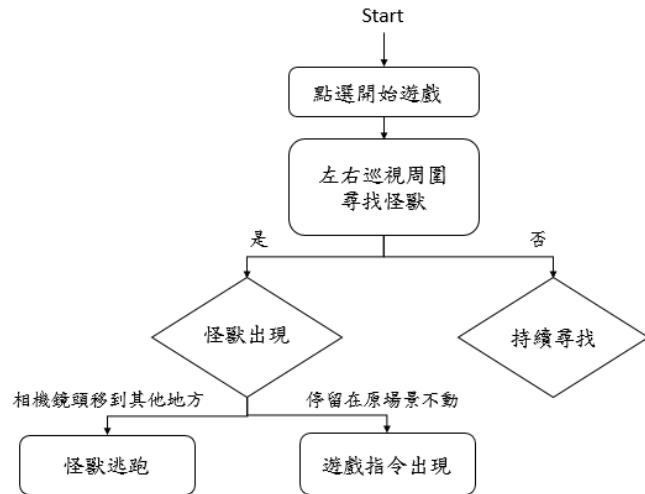


Figure 2.11: 主遊戲操作流程

如圖 2.11 所示，主遊戲流程如下：

流程一：進入遊戲畫面。

流程二：在所在的環境尋找怪獸藏在哪裡。

流程三：當怪獸出現便產生遊戲指令，透過肌肉感測器攻擊怪獸。

流程四：攻擊期間必須維持手的平衡，若離開場景，則怪獸逃跑。

流程五：如果在時間內完成指令成功；相反地，則挑戰失敗。

## (2) 小遊戲模式

操作流程：玩家有兩種模式可以選擇，一種是力量模式一種是速度模式，玩家要在限制的時間內完成指定的目標。其中包括：

1. 力量模式：訂一個力量標準，超過記為有效，每 0.15 秒計為一次，直到集滿指定的量或超過限制時間後結束遊戲。
2. 速度模式：將訊號過濾後以快速傅立葉轉換計算一個時間間格內發生的頻率，只要速度不是 0 就記為有效，直到集滿指定的量或超過限制時間後結束遊戲。

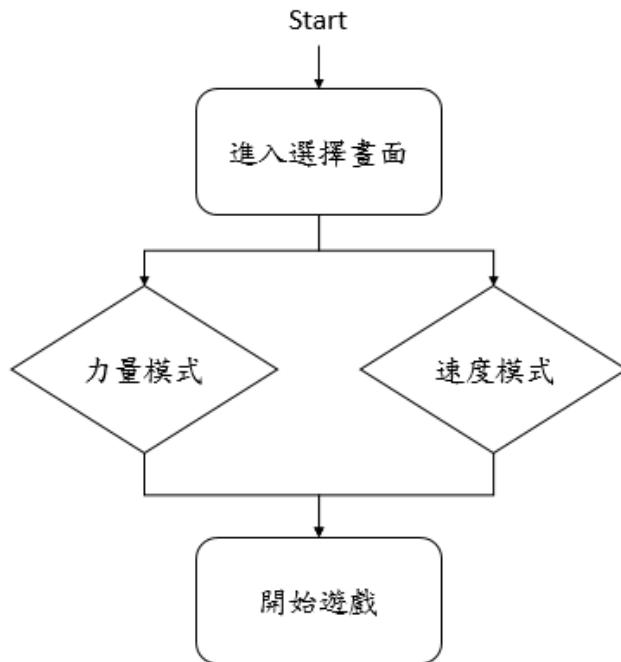


Figure 2.12: 挑戰模式流程圖

如圖 2.12所示，挑戰模式流程如下：

- 流程一：進入挑戰模式選擇關卡。
- 流程二：關卡分別為力量模式與速度模式。
- 流程三：點選後進入遊戲畫面。

# Chapter 3

## 場景分析與肌肉感測訊號分析

本章節著重介紹場景與肌肉感測訊號分析技術。將描述這些功能包含目的、演算法流程與程式碼片段說明。

### 3.1 肌肉感測器訊號分析

目的：

用於分析由 Arduino 傳至手持裝置的肌肉感測器訊號。過濾不可用的訊號後，利用快速傅立葉轉換得知訊號的大小與頻率。

演算法流程：

如圖 3.1所示，陣列一為隨時更新最新資訊的短陣列（紅色），陣列二為隨時更新最新訊號的長陣列（綠色）。下面會說明兩個陣列分別執行的不同計算工作。

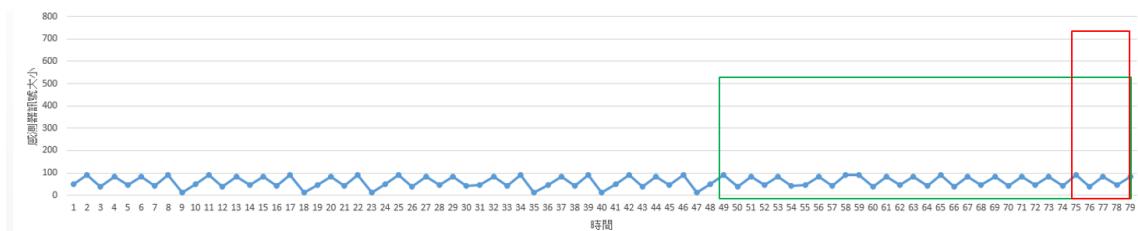


Figure 3.1: 陣列與訊號關係示意圖

如圖 3.2和圖 3.3所示，在大的陣列裝滿後，以小的陣列用於判斷訊號是否在變化，有的話就以大的陣列判斷變換的節奏。

```

/*陣列1*/
    logs[7] = o;
    int u;
    for (u = 0; u <= 6; u++) {
        fftcount++;
        logs[u] = logs[u + 1];
    }

/*陣列2*/
    longlogs[31] = o;
    int v;
    for(v=0;v<=30;v++) {
        longfftcount++;
        longlogs[v] = longlogs[v + 1];
    }

```

Figure 3.2: 設兩個分別為大跟小的陣列

```

/*開始*/
double move=1; /*判斷已沒有動的變數*/
if(longfftcount>=64) {/*第二陣列滿*/
    analysis a = new analysis();
    move = a.fftcalculate(logs)[1];

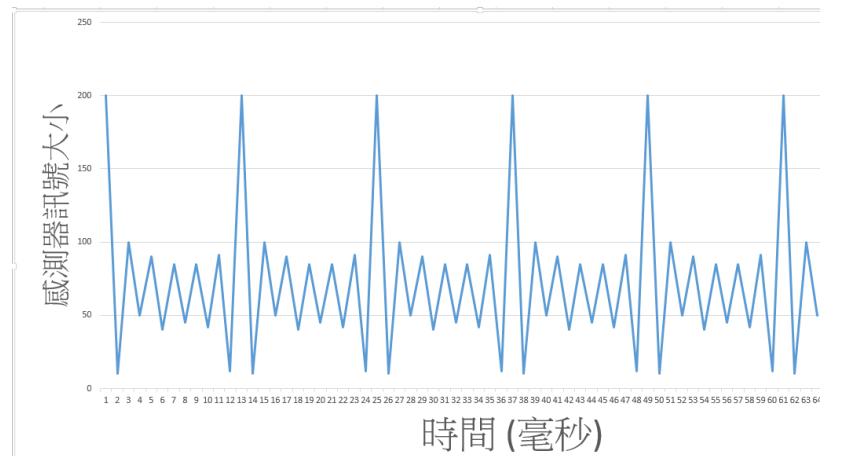
    //note.append(""+o);
    //note.append((int)a.fftcalculate

    if (move<10){/*不動的情況*/
        note.setText(" ");
        note.append("not move");
    }
}

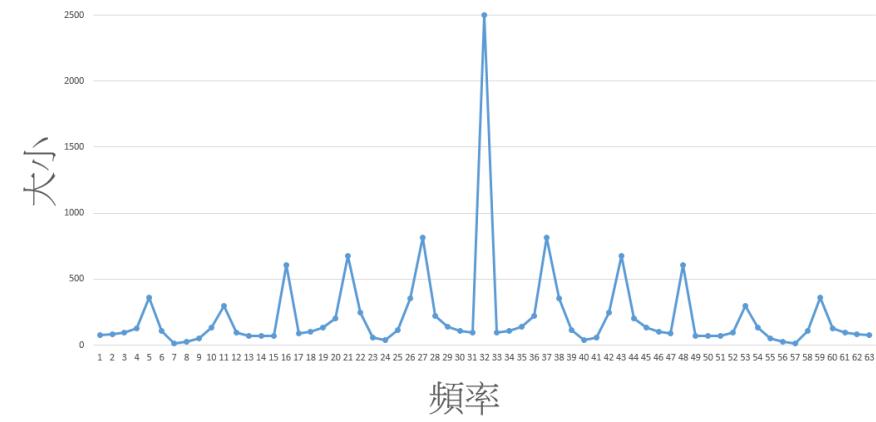
```

Figure 3.3: 開始執行的片段

如這兩張圖 3.4a 和 3.4b 所示，轉換後的結果較轉換前容易判斷頻率，在遊戲的演算法中可以較容易的定義肌肉放鬆節奏的快慢。



(a) 轉換前的訊號



(b) 轉換後的訊號

Figure 3.4: 感測訊號轉換

如圖 3.5 所示，傅立葉轉換主要的功能是用於信號在時域（或空域）和頻域之間的變換。我們用轉換後的結果來判斷訊使用者行為有沒有在動，然後再用另一個更長的來判斷使用者行為，這樣做的目的是減少延遲的時間。

```

static void fft(Complex[] buffer) {
    int bits = (int) (log(buffer.length) / log(2));
    for (int j = 1; j < buffer.length / 2; j++) {
        int swapPos = bitReverse(j, bits);
        Complex temp = buffer[j];
        buffer[j] = buffer[swapPos];
        buffer[swapPos] = temp;
    }

    for (int N = 2; N <= buffer.length; N <= 1) {
        for (int i = 0; i < buffer.length; i += N) {
            for (int k = 0; k < N / 2; k++) {

                int evenIndex = i + k;
                int oddIndex = i + k + (N / 2);
                Complex even = buffer[evenIndex];
                Complex odd = buffer[oddIndex];

                double term = (-2 * PI * k) / (double) N;
                Complex exp = (new Complex(cos(term), sin(term)).mult(odd));

                buffer[evenIndex] = even.add(exp);
                buffer[oddIndex] = even.sub(exp);
            }
        }
    }
}

```

Figure 3.5: 快速傅立葉程式碼片段

由下圖 3.6得知，將速度分為三個階段，越快的會加分的越快。

```

        else /*有動的情況*/
            note.append("move" + " ");

        if(( a.fftcalculate(longlogs) [1]>400)){
            note.setText(" ");
            note.append("fast" + " ");
            count++;
        }

        if(( a.fftcalculate(longlogs) [2]>1000)){
            note.setText(" ");
            note.append("very fast" + " ");
            count++;
        }

        if(( a.fftcalculate(longlogs) [3]>1000)){
            note.setText(" ");
            note.append("very fast" + " ");
            count++;
        }

        quickbar.setProgress(count);
    }
}

```

Figure 3.6: 速度的分級

## 3.2 場景分析

目的:

1. 目的一為尋找可藏怪獸之場景。利用 Canny 演算法進行邊緣偵測計算場景密度，接著以 AKAZE 偵測周圍特徵點是否夠多，當特徵點夠多代表場景夠複雜，得以將怪物縫合在此相機畫面上。
2. 目的二則是怪獸的擴增實境合成。以兩畫面之特徵點的對應關係，判斷點對點之間是否相符進而分析手持裝置是否移動到其他畫面，當下個畫面與原先怪物出現畫面的 Match Points 不夠多時，怪物則會消失不見。

演算法流程：

(1) 尋找場景

步驟 1：首先取一張影像，使用 GaussianBlur 將讀取到的影像雜訊去除。

步驟 2：再來使用 Canny 偵測邊緣，當邊緣強度超過某個門檻比例，進入特徵點的計算。

步驟 3：接著連續讀入兩張相機影像，判斷此場景是否有夠多「獨特的點」。所謂「獨特的點」，即為利用 AKAZE 尋找特徵點時，此特徵點與其他特徵點很不像。

步驟 4：如果「獨特的點」夠多，便能將怪獸圖像縫合至場景中的特徵點。

如圖 3.7，為了使畫面處理速度加速，我們只取整個螢幕畫面的中間區域做特徵點的偵測與計算。原先加速的方法是將畫面的長寬各除以 2，但此方法會縮減特徵點的數量，因此我們改用只針對中間畫面進行測量，縮小範圍及不更動特徵點個數可使兩個畫面的比對更加精準。

```
//使畫面加速
//Imgproc.resize(temp1, temp1, new Size(temp1.width() / speedup, temp1.height() / speedup));//長寬縮小speedUp倍 加速
//Imgproc.resize(temp2, temp2, new Size(temp2.width() / speedup, temp2.height() / speedup));//長寬縮小speedUp倍 加速

//只取畫面中間
Rect roi1 = new Rect((int)(temp1.width()*0.25), (int)(temp1.height()*0.25), temp1.width()/2, temp1.height()/2);
Mat subTemp1 = new Mat(temp1, roi1);
Rect roi2 = new Rect((int)(temp2.width()*0.25), (int)(temp2.height()*0.25), temp2.width()/2, temp2.height()/2);
Mat subTemp2 = new Mat(temp2, roi2);
```

Figure 3.7: 畫面加速程式碼片段

圖 3.8a 與圖 3.8b 畫出特徵點的數量與實際位置。



(a) 特徵點數量 <10

(b) 特徵點數量 >10

Figure 3.8: 特徵點數量足夠與否

## (2) 擴增實境合成

步驟 1：將怪獸圖像的四個角利用 Homography 的方式縫合至場景中特徵點的四個點。由於原先只取畫面的中間區域，因此我們必須將特徵點的座標與一個特定矩陣相乘，使座標與原先螢幕位置相符。

步驟 2：將轉換前的特徵點位置記錄起來，以便與下個畫面做比較。

步驟 3：當找到地方藏怪獸後，則開始判斷接下來相機讀到的場景是否與原先藏怪獸的場景相符。兩者進行 KnnMatch 的計算，如果 Match Point 的數量大於 12，則為相似的場景。

步驟 4：接著轉換原先存起來的特徵點至與原先螢幕位置相符的座標，再計算原先存怪獸的場景與此場景的 Homography，綜合以上兩點將怪獸貼上。

步驟 5：如果場景與原先場景兩者比對的特徵點太少，且 Homography 的運算不合常理，代表場景不同，則怪獸逃跑不縫合。

當計算 Homography 的行列式值，判斷其值是否大於 1。若行列式值小於 0 時，怪獸會呈現鏡像效果縫合在場景中，形成扭曲形狀；行列式值過小趨近於 0 時，會是退化矩陣，如圖 3.9a 與圖 3.9b 所示，兩者的行列式值分別為 -0.06 與 -0.08，因此須避免此情況發生。

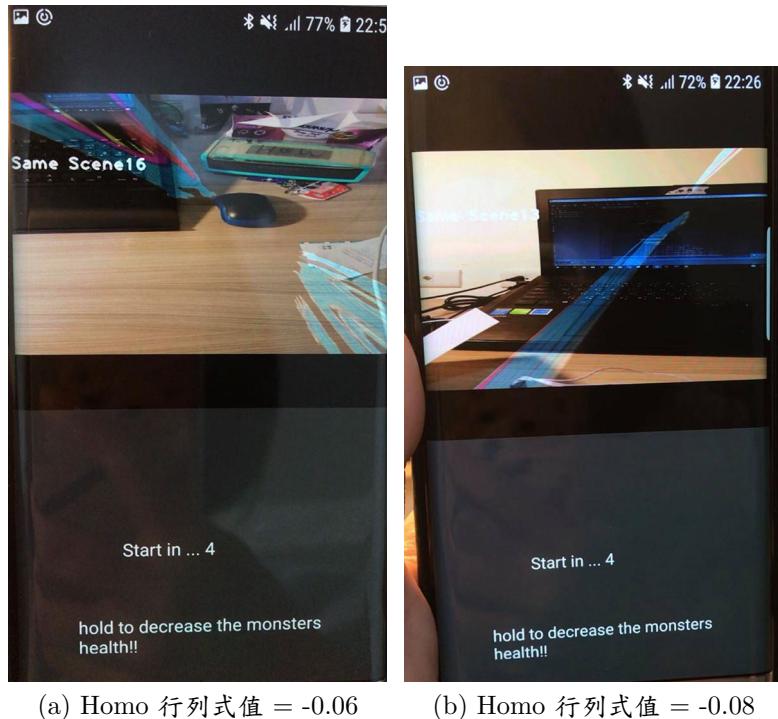
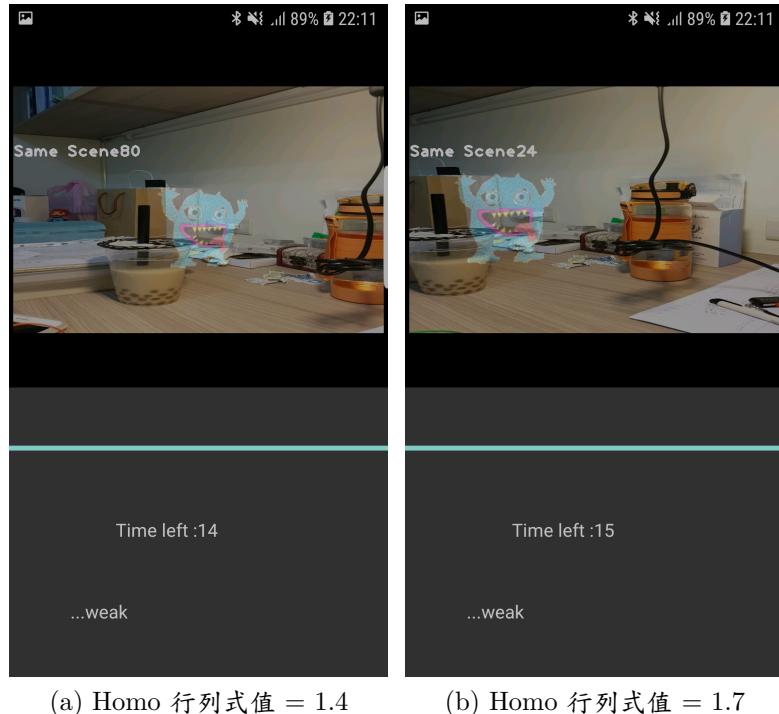


Figure 3.9: 條件不好的 Homography

圖 3.10a 和圖 3.10b 為擴增實境合成成功的樣貌。



(a) Homo 行列式值 = 1.4      (b) Homo 行列式值 = 1.7

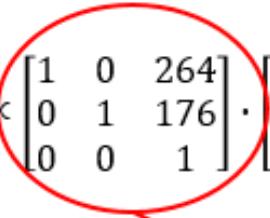
Figure 3.10: 條件好的 Homography

圖 3.11為怪獸四個角之座標分別與 Homography 矩陣相乘，得出怪獸縫合場景位置之座標。

$$\text{經 Homography 轉換後之座標} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \propto H \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{原場景之座標}$$

Figure 3.11: Homography 運算公式

如圖 3.12所示，在擴增實境合成步驟 1 所提及，由於原先只取畫面的中間區域，因此必須將特徵點的座標與一個特定矩陣相乘。手持裝置鏡頭畫面長寬為 704\*1056，我們得知只取螢幕四邊往內長寬分別縮減 1/4 之畫面，因此獲得小畫面座標 (0, 0) 在原畫面座標 (176, 264) 這個位置。

$$\begin{bmatrix} 264 \\ 176 \\ 1 \end{bmatrix} \leftarrow \begin{bmatrix} 1 & 0 & 264 \\ 0 & 1 & 176 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$


特定矩阵

Figure 3.12: 運算結果說明

# Chapter 4

## 實驗結果

本章節將實際展示本專題成果，將依序展示各項功能在實驗環境下的結果，並講述實作的過程及在實作途中所遭遇到的問題與我們對於問題的解決方式，以下有感測器參數測量、場景分析效能評估、使用者評估。

### 4.1 實驗環境

下面列出本系統的硬體設備與軟體開發環境：

手持裝置: Android 8.0.0, RAM:4GB

相機視覺處理技術: OpenCV 3.1.0

Arduino 程式開發環境: Arduino 1.6.12

Android 程式開發環境: Android Studio 2.3

### 4.2 肌肉感測器參數測量

我們在開始製作系統之前，先進行了肌肉感測器的實際測量與統計，分別記錄了手臂肌肉的各種姿勢。以下為測量結果的數據：

Table 4.1: 感測器測量數據表

測量數據：

	孫	周
前手臂放鬆時	45 以下	50-60
用力握拳時	>70	60-70
整隻完全出力	>200	>200
二頭肌放鬆舉著	70-100	70-110
用力時	>200	>200

Table 4.2: 感測器手臂測量表

手臂下垂：

沒出力	70↓
預備力	70-200
集氣	200↑

手臂舉起：

舉著沒出力	110↓
正要出力	110-200
舉著用力	200↑

Table 4.3: 感測器姿勢測量表

各種姿勢：

甩蝴蝶袖	維持 200-400 數秒
突然擊拳	瞬間跳到 >500
拿重物	100-200

根據感測器參數測量結果，表格右方數字為使用者貼上肌肉感測器測量後之直接輸出數值，表格左方內容為指定動作。利用以上所彙整出之表格，我們可以看出肌肉用力時，感測器輸出之數值的平均大小大約為 120，放鬆時則為 20 至 50 之間。

### 4.3 場景分析效能評估

如圖所示：

場景一 (圖 4.1a)：場景較為單純，經計算後場景輪廓密度小，不將怪獸縫合至影像中。

場景二 (圖 4.1b)：場景輪廓密度足夠，但特徵點少，因此怪獸縫合得較扭曲。

場景三 (圖 4.1c)：場景輪廓密度足夠，特徵點也足夠，怪獸縫合完整且穩定。

綜合上述三張圖片，我們傾向將怪獸藏在環境較為複雜的場景，此結果也與原本設定相符。

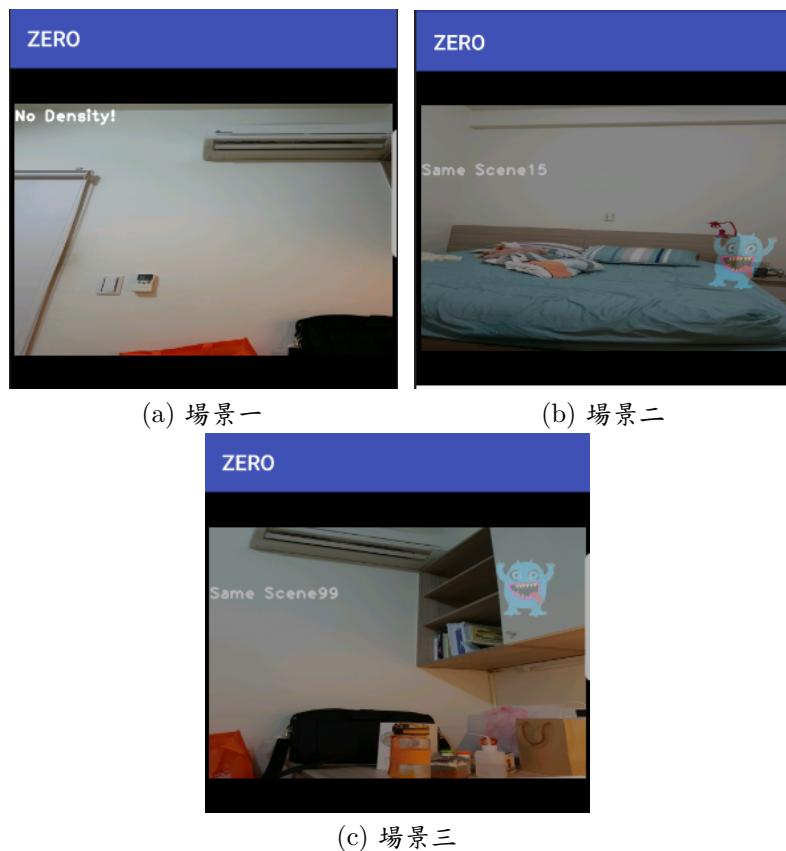


Figure 4.1: 遊戲場景

如圖 4.2，怪獸會依照距離遠近的不同，縫合在場景中跟著場景放大、縮小和傾斜。



Figure 4.2: 怪獸呈現

圖 4.3a 為直立式地拿著持裝置；由於我們將相機鏡頭鎖住，圖 4.3b 為當手持裝置倒著拿著時，怪獸會跟著鏡頭前的場景維持正的。

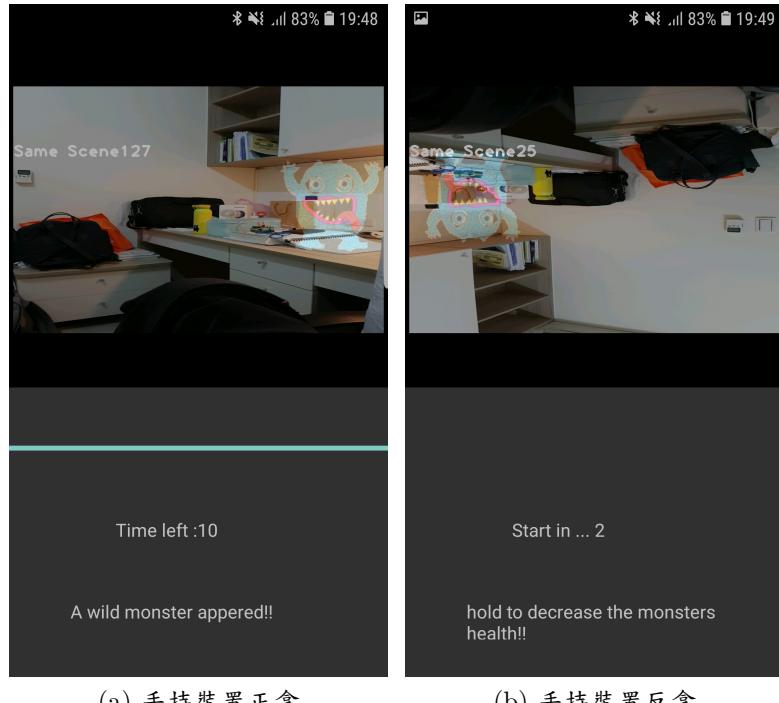
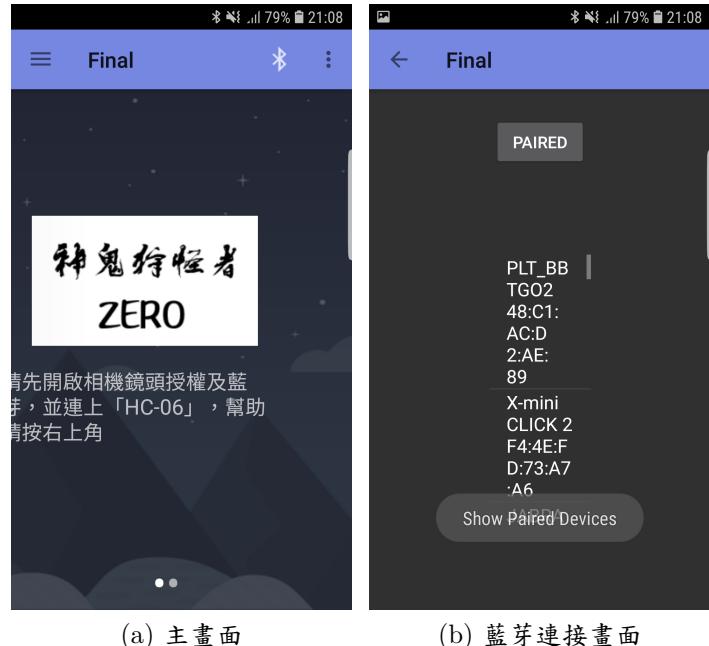


Figure 4.3: 手持裝置的翻轉

## 4.4 專題結果展示

以下展示本專題之完整流程：

圖 4.4a 和圖 4.4b 分別展示遊戲主畫面以及藍芽搜尋和連接畫面。

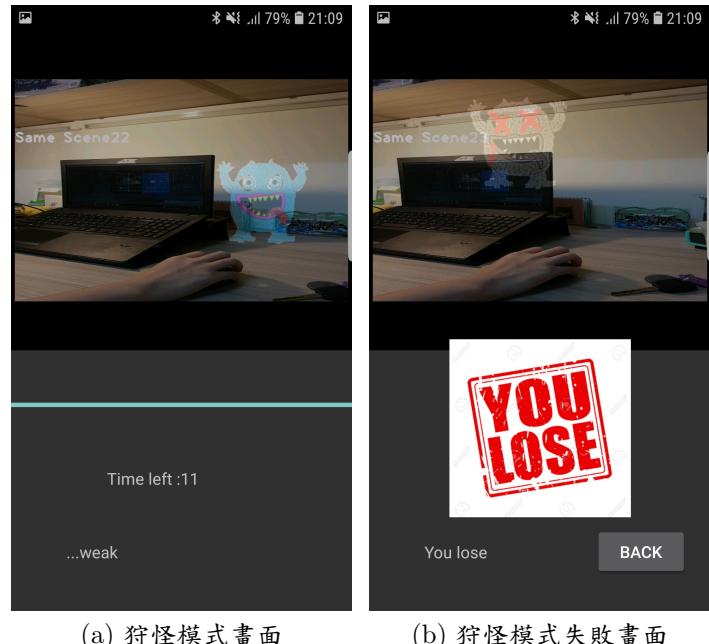


(a) 主畫面

(b) 藍芽連接畫面

Figure 4.4: 遊戲流程畫面一

圖 4.5a 和圖 4.5b 分別呈現狩怪模式進行中與失敗畫面。



(a) 狩怪模式畫面

(b) 狩怪模式失敗畫面

Figure 4.5: 遊戲流程畫面二

圖 4.6a 至圖 4.6e 為小遊戲模式畫面。



Figure 4.6: 遊戲流程畫面三

## 4.5 使用者體驗

以下為尋找不同玩家體驗本遊戲，並給予評分及評論。

Table 4.4: 玩家評分表

姓名	有趣度 (1-10)	順暢度 (1-10)	整體評價 (1-10)	留言區
玩家 1	8	6	7	我覺得小遊戲很難，手很痠。
玩家 2	9	7	8	希望能有更多不同關卡

# Chapter 5

## 討論與結論

在本次專題中，我們設計的遊戲的優勢在於這是一個非常創新的玩法，且能在遊戲中兼具運動效果。此外，本系統包含了許多技術，包括利用 OpenCV 模組進行場景分析以及使用快速傅立葉轉換的方法分析肌肉感測器的訊號。軟體層面透過 OpenCV 影像處理技術，實作怪獸融合至場景效果，目前會遇到螢幕晃動因素干擾，如手部晃動、特徵點過少產生縫合效果不佳等問題，因此需在光源良好的環境下執行此系統。在肌肉感測器訊號分析上，雖然在顯示怪獸的畫面上會有些延遲，又或者在決定快速傅立葉的時間框時，當時間框太長會延遲，時間框太短會影響判斷準度，我們已盡量將整個系統優化到最順暢的程度。另外在感測的過程中偶爾會接收到雜訊，它通常是一個非常大的值（超過人體能達到的數字）。在理想固定的情況下（例如：放鬆不施力），他的數值會是保持固定的，然而，在現實的情況下感測的結果會有微小的波動，所以會造成計算與判斷上的誤差。因此我們想了一些方法解決這個問題，我們將轉換後頻率的極高頻過濾，雖然還是會有少部分漏掉，但其誤差已經幾乎不影響遊戲體驗。

本系統所使用肌肉感測器可以貼在任何明顯肌群部位，可以更符合使用者需求。比如從手臂更換到小腿，就可以有更多不同玩法。在未來，我們希望能夠為系統增設關卡與難度，且加強功能至兩台行動裝置的連線與對決。另外，新增自行畫怪獸的功能，將畫出的怪獸成為攻擊的對象也是一種有趣的互動模式。

# 附錄

## 附錄 1.1 過濾感測器雜訊程式片段：

```
/*過濾*/
if(i>100000) i=i/1000;

else if(i<100){
    o=i;
}

else if((i>10000)&&(i<100000)){
    if(i%1000==v%1000)
        o=i/1000;
    else if(i%100==v%100)
        o=i/100;
}
else if((i>1000)&&(i<10000)){
    if(i%100==v%100)
        o=i/100;

    else if((i%10==v%10)&&(i%100!=v%100))
        o=i/10;
}
else if(i<1000){
    if(i%10==v%10)
        o=i/10;
    else
        o=i;
}

v=i;
/*過濾end*/
```

## 附錄 1.2 快速傅立葉轉換式片段:

```
static void fft(Complex[] buffer) {
    int bits = (int) (log(buffer.length) / log(2));
    for (int j = 1; j < buffer.length / 2; j++) {
        int swapPos = bitReverse(j, bits);
        Complex temp = buffer[j];
        buffer[j] = buffer[swapPos];
        buffer[swapPos] = temp;
    }

    for (int N = 2; N <= buffer.length; N <= 1) {
        for (int i = 0; i < buffer.length; i += N) {
            for (int k = 0; k < N / 2; k++) {

                int evenIndex = i + k;
                int oddIndex = i + k + (N / 2);
                Complex even = buffer[evenIndex];
                Complex odd = buffer[oddIndex];

                double term = (-2 * PI * k) / (double) N;
                Complex exp = (new Complex(cos(term), sin(term)).mult(odd));

                buffer[evenIndex] = even.add(exp);
                buffer[oddIndex] = even.sub(exp);
            }
        }
    }
}
```

## 附錄 1.3 場景分析密度程式片段:

```
/*----計算密度----*/
//去除雜訊
Imgproc.GaussianBlur(temp1, blurred, new Size(5, 5), 1, 1);
//用canny偵測邊緣
Imgproc.Canny(blurred, canny, 30, 150, 5, true);
//用threshold創造一個binary image
Imgproc.threshold(canny, blackThres, 0, 255, Imgproc.THRESH_BINARY + Imgproc.THRESH_OTSU);
Imgproc.threshold(canny, whiteThres, 0, 255, Imgproc.THRESH_BINARY_INV);
//讀取pixels
bp = Core.countNonZero(blackThres);
wp = Core.countNonZero(whiteThres);
//計算密度(numPixelsOn/totalPixels)
density = bp / (bp + wp);
/*----計算密度----*/
```

## 附錄 1.4 場景分析特徵點程式片段：

```
//準備兩個list來存goodmatch
List<KeyPoint> gdKeyPoint = new ArrayList<KeyPoint>();
List<KeyPoint> gdCompareKeyPoint = new ArrayList<KeyPoint>();
//配對兩張圖的descriptors
List<MatOfDMatch> matches = new ArrayList<MatOfDMatch>();
descriptorMatcher.knnMatch(storeValue.descriptors1, descriptors3, matches, 2);
LinkedList<DMatch> good_matches = new LinkedList<DMatch>();
//----依matches.size()來篩選goodmatch
for (int i = 0; i < matches.size(); i++) {
    MatOfDMatch matofDMatch = matches.get(i);
    DMatch dmatches[] = matofDMatch.toArray();
    DMatch dml = dmatches[0];
    DMatch dm2 = dmatches[1];
    if (dml.distance < 0.5 * dm2.distance) {
        good_matches.addLast(matofDMatch.toArray()[0]);
    }
}
for (int j = 0; j < good_matches.size(); j++) {
    gdKeyPoint.add(orgKeyPoint.get(good_matches.get(j).queryIdx));
    gdCompareKeyPoint.add(orgCompareKeyPoint.get(good_matches.get(j).trainIdx));
}
```

## 附錄 1.5 場景座標轉換程式片段：

```
if (good_matches.size() > 10) {
    message = "Same Scene" + good_matches.size();
    Imgproc.putText(drawInImg, message, new Point(0, 200), FONT_HERSHEY_PLAIN, 3, new Scalar(255, 255, 255));
    List<Point> sceneTemp1 = new LinkedList<Point>();
    MatOfPoint2f scenel = new MatOfPoint2f();
    for (KeyPoint k : gdKeyPoint) {
        scenel.add(k.pt);
    }
    scenel.fromList(sceneTemp1);
    List<Point> sceneTemp2 = new LinkedList<Point>();
    MatOfPoint2f scene2 = new MatOfPoint2f();
    for (KeyPoint k : gdCompareKeyPoint) {
        sceneTemp2.add(k.pt);
    }
    scene2.fromList(sceneTemp2);
    Mat H = Calib3d.findHomography(scenel, scene2);

    Mat scenel_corners = new Mat(4, 1, CvType.CV_32FC2);
    Mat scene2_corners = new Mat(4, 1, CvType.CV_32FC2);

    //原本怪獸貼的區塊
    Point point0 = storeValue.spKeyPoint1.getFirst();
    Point point1 = new Point(point0.x + width, point0.y);
    Point point2 = new Point(point0.x + width, point0.y + height);
    Point point3 = new Point(point0.x, point0.y + height);

    scenel_corners.put(0, 0, new double[]{point0.x, point0.y});
    scenel_corners.put(1, 0, new double[]{point1.x, point1.y});
    scenel_corners.put(2, 0, new double[]{point2.x, point2.y});
    scenel_corners.put(3, 0, new double[]{point3.x, point3.y});

    Core.perspectiveTransform(scenel_corners, scene2_corners, H);
```

## 附錄 1.6 怪獸縫合程式片段:

```
Point a_src = new Point(0, 0);
Point b_src = new Point(width - 1, 0);
Point c_src = new Point(width - 1, height - 1);
Point d_src = new Point(0, height - 1);
Point[] corner_dst = scene.toArray();
Point a_dst = new Point(corner_dst[0].x, corner_dst[0].y);
Point b_dst = new Point(corner_dst[1].x, corner_dst[1].y);
Point c_dst = new Point(corner_dst[2].x, corner_dst[2].y);
Point d_dst = new Point(corner_dst[3].x, corner_dst[3].y);
Point[] pts_src = new Point[4];
pts_src[0] = a_src;
pts_src[1] = b_src;
pts_src[2] = c_src;
pts_src[3] = d_src;
LinkedList<Point> pts_dst = new LinkedList<Point>();
pts_dst.add(a_dst);
pts_dst.add(b_dst);
pts_dst.add(c_dst);
pts_dst.add(d_dst);

MatOfPoint2f src = new MatOfPoint2f();
src.fromArray(pts_src);
MatOfPoint2f dst = new MatOfPoint2f();
dst.fromList(pts_dst);

Mat H = Calib3d.findHomography(src, dst);
Imgproc.warpPerspective(readMat, resultMat, H, resultMat.size());
```

## 附錄 1.7 感測器訊號讀入程式片段:

```
public static Session getSession(){
    if(session == null){
        session = new Session();
        return session;
    }
    else{
        return session;
    }
}
```

# Bibliography

- [1] [讀取系統圖像資源]  
<http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2012/1201/655.html>
- [2] [介面按扭]  
<http://hukai.me/android-training-course-in-chinese/basics/actionbar/adding-buttons.html>
- [3] [首頁介面效果]  
<https://dotblogs.com.tw/starhao/2016/10/08/012050>
- [4] [場景分析特徵點偵測]  
[https://opencv-python-tutorials.readthedocs.io/en/latest/py\\_tutorials/py\\_feature2d/py\\_table\\_of\\_contents\\_feature2d/py\\_table\\_of\\_contents\\_feature2d.html#py-table-of-content-feature2d](https://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html#py-table-of-content-feature2d)
- [5] [圖像合成]  
<http://peaceandhilightandpython.hatenablog.com/entry/2016/01/16/002028>
- [6] [快速傅立葉轉換]  
[https://rosettacode.org/wiki/Fast\\_Fourier\\_transform#C.2B.2B](https://rosettacode.org/wiki/Fast_Fourier_transform#C.2B.2B)
- [7] [藍芽連線]  
<https://home.gamer.com.tw/creationDetail.php?sn=3671289>
- [8] [Session]  
<https://blog.csdn.net/ZDW86/article/details/35795125>