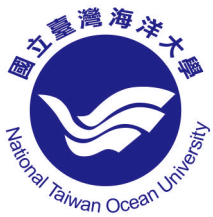
國立臺灣海洋大學資訊工程學系專題報告

AndMuscle

基於 Android 行動裝置與肌肉感測器之網路連線即時對戰擴增實境遊戲



00557128 3B 林令婕

00557149 3B 王佳君

指導教授：張欽圳老師

中華民國 108 年 10 月 03 日

Abstract

本專題設計為一款雙人網路連線對戰的擴增實境遊戲，於 Android 行動裝置上開發，結合影像處理及肌肉感測器技術，以之為主架構來設計整個遊戲的開發。在這款遊戲中，利用肌肉感測器測量肌肉活動狀況，及不斷改變肌肉施力大小與頻率，來達成開發者要求。建立網路通道，即時傳輸感測器數據與雙方畫面，以現實中對方遊戲者臉部影像為基礎，改變其皮膚色調，且在指定位置拼貼上逗趣的圖示。期許玩家在煩悶的生活中，透過此遊戲藉由肌肉感測器活動筋骨之餘，看見對方即時畫面如視訊功能般，在遊玩過程中增進彼此間的溫度、紓解身心靈的疲憊感。

Contents

1 介紹

1.1 動機與目的

在現今 VR、AR 如此熱門的時代，不難察覺人們對於遊戲體驗不再滿足於透過螢幕點擊及搖桿來與遊戲對話。

除了舊有侷限於遊戲機台、幾乎未和環境結合的定點遊戲，近期多推出創新的互動類型遊戲。例如熱門的 Switch LABO 紙箱系列，利用簡易的物件模擬出遊戲環境，使玩家有身歷其境的體驗。因此我們對於外接式硬體以達成高互動性之遊戲，深感興趣。故選擇使用肌肉感測器作為該專題之主軸。

利用手臂肌肉出力，得以模擬真實的出拳；取得真實生活的影像為素材，呈現在遊戲畫面後，再加以修改；且同時加入網路即時連線之功能，提供人與人之間互動及對戰的平台。使遊戲方式能不僅限於單機或因距離受限，基於相似於視訊功能之遊戲模式，玩家在遊戲期間可同時聯繫感情。

總合上述分析，我們開發「行動裝置與肌肉感測器之網路連線即時對戰擴增實境遊戲」－一款比一般手機遊戲更真實的遊戲程式。

1.2 系統介紹

分為兩種模式：「StartGame」-網路連線版本、「ChallengeMode」-單機模式版本。

1.2.1 網路連線版本

本系統主要利用：肌肉感測器量測手臂肌肉放鬆或緊繃，讀取感測器數據，判斷施力的大小；藉由 TCP 架構建構出網路通道，建立伺服器端及客戶端之連線，用來傳輸肌肉感測器數據及即時影像；透過擴增實境技術，使兩位玩家根據所得到之參數，改變對方的面容影像；此外，定義多種數據變化形式，增加玩法之豐富度及難易度，使整體遊戲更具趣味性。

在進入遊戲畫面後，使用手持裝置的相機來拍下遊戲者面部，並以電腦視覺技術偵測出臉部輪廓及五官位置，此即可獲得遊戲的初始影像素材。遊戲開始後，將使用肌肉感測器，測量遊戲者手臂肌肉活動狀況之數據，利用訊號分析技術 (如：快速傅立葉轉換)，可知玩家的肌肉施力大小與收縮節奏。而為使雙方遊戲者能獲得彼此即時的面部影像及感測器之數據，兩方遊戲者皆與伺服器連接後，將遊戲過程中的畫面與參數傳送給伺服器端，再由伺服器端轉送給對

方，以達成即時傳遞之目的。而後透過分析出的肌肉感測器數據，比對定義的遊戲招式，使得雙方遊戲者之臉部畫面會有不同程度的改變。且為了使手機移動或改變角度後，臉部被更改的情形仍相同，我們會對臉部定座標，則透過相對位置，即可計算應被更改之處。最後，依據臉部畫面被更改之嚴重程度，判斷獲勝者為何方，即遊戲結束。

1.2.2 單機模式版本

本系統為練習模式，分為：「BusterActivity」-力量模式、「QuickActivity」-速度模式。

肌肉感測器操作與網路連線版本相同，玩法則更改為：玩家須在限定時間內，完成指定目標，即算獲勝。

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。

1.3 報告組織

本實驗報告第二章介紹系統之軟硬體架構：硬體包括系統外觀介紹及所使用之感測器與技術，軟體架構包括系統流程圖及系統中各功能的用途、流程與實作方法。第三章將著重介紹軟體演算法之過程，包括肌肉感測器之數據分析及影像變化之處理。第四章為實驗結果，包括系統實測之數據效能分析及整體系統遊戲之體驗。第五章為此專題之結論。

2 軟體與硬體架構

此章節介紹，軟體系統及硬體系統之架構與技術。

2.1 軟硬體架構圖

下圖 1 所示，系統架構分為兩大項：左半部為軟體，包含 Android 手持裝置之網路連線系統與 OpenCV 模組；右半部則為硬體，包含 Arduino 與肌肉感測器。軟硬體通訊，則由藍芽負責連接。

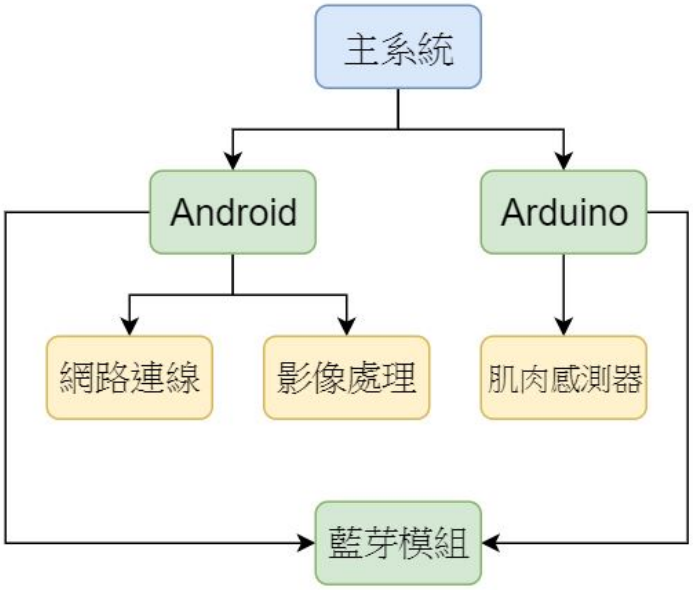


Figure 1: 軟硬體架構圖

2.2 硬體架構

2.2.1 系統外觀介紹

本系統外觀分別為 Android 手持裝置、肌肉感測器、藍芽裝置、電源四項，介紹如下：

1. Android 手持裝置（圖 2(a)）：本專題使用到手機上的相機、藍芽功能。

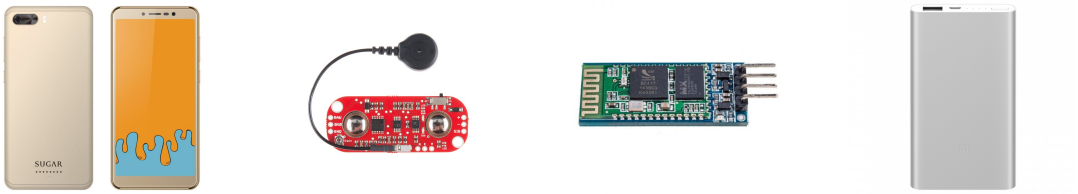
2. 肌肉感測器（圖 2(b)）：

   - 供電電壓：+3.1V~+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。

3. 藍芽裝置（圖 2(c)）：藍芽模組可以與手持裝置的藍芽互相傳送接收訊息。

4. 電源（圖 2(d)）：使用電流穩定既可重複使用的行動電源作為供電來源。



(a) Android 手持裝置　(b) 肌肉感測器　(c) 藍芽裝置　(d) 電源

Figure 2: 系統外觀介紹

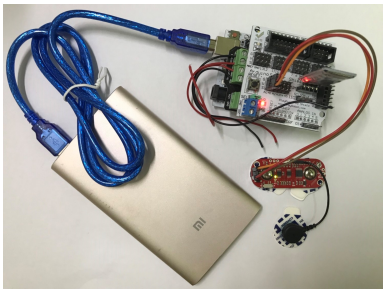


Figure 3: 硬體系統整合

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。

- 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。
- 圖 4 為感測器流程圖

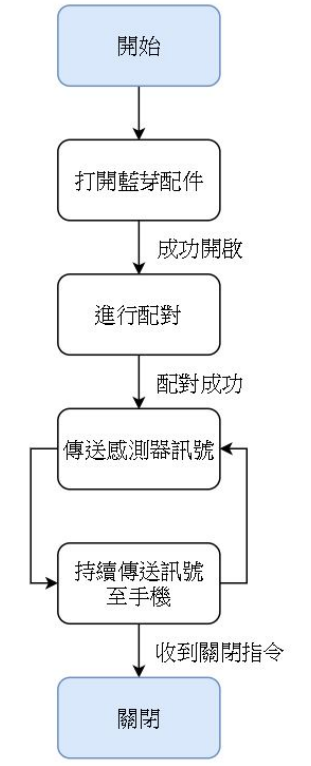


Figure 4: 感測器流程圖

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。
- 藍芽開啟之操作流程:

  Step1. 打開手持裝置藍芽。

  Step2. 點選「HC-06」進行配對。

Step3. 回到應用程式主畫面，點選藍芽連線。

Step4. 顯示連線成功。

- 圖 5 為藍芽開啟流程圖

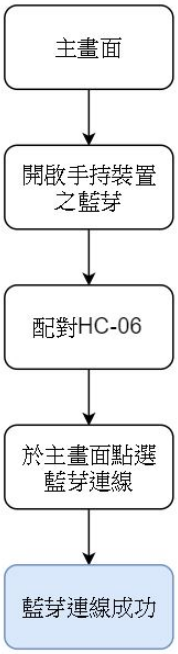


Figure 5: 藍芽開啟流程圖

3. Android 相機

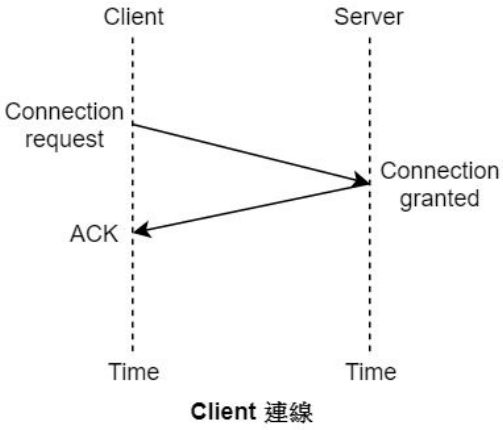
- 我們使用 OpenCV Android SDK 提供的 JavaCameraView 來調用 android 相機。JavaCameraView 繼承 Android 內建 SurfaceView 但必須呼叫 onCameraFrame 來取得每帧相機影像來供後續 OpenCV 程式庫處裡。OpenCV(Open Source Computer Vision Library) 是一套跨平台的電腦視覺式庫，支援圖像處裡、電腦視覺和模式識別的程式開發。在本專題中我們使用 OpenCV 的臉部偵測及影像縫合功能將圖示縫合於臉部畫面上，以達到擴增實境的效果。

2.3 軟體架構

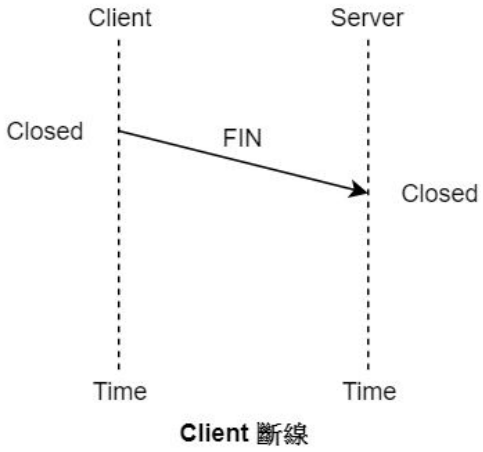
軟體部分中包含了網路連線功能、藍芽功能、Arduino 端功能以及遊戲端功能，接下來會依序介紹其用途及操作流程。

2.3.1 網路連線功能

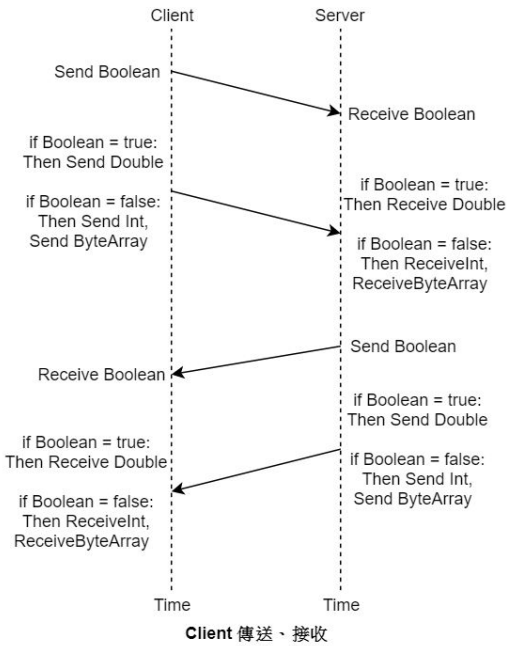
- 基於 TCP 網路架構，建構出 Server 端及 Player（Client）端，使影像與數據以此通道傳輸。
- 使用不同 Thread 分別執行 inputstream 與 outputstream，於網路通道上進行輸入及輸出，且以 lock、unlock、await、signal 處理同步問題。
- 圖 6 為網路協定圖



(a) Player 連線協定圖



(b) Player 結束連線協定圖



(c) Player 傳送、接收影像協定圖

Figure 6: 網路協定圖

- 以下為 Server 端之程式碼片段
  - 圖 7 為建立連線

```
public Server() {
    try {
        serverSocket = new ServerSocket(port);
        System.out.println("Server started on port " + serverSocket.getLocalPort() + "...");
        System.out.println("Waiting for client...");
        thread = new Thread(this);
        thread.start();
    } catch (IOException e) {
        System.out.println("Can not bind to port : " + e);
    }
}
```

Figure 7: 建立連線

  - 圖 8 為接收數據

```
is = socket.getInputStream();
os = socket.getOutputStream();
dis = new DataInputStream(is);
dos = new DataOutputStream(os);

file = new File(path);
rand = new RandomAccessFile(file, "rw");

while(true) {
    type  = dis.readBoolean();
    System.out.println("Receive Boolean: " + type);

    if(type == true) { // muscle data
        muscleData = dis.readDouble();
    }
    else { // byte data
        byteLenData = dis.readInt();
        System.out.println("Receive image file length: "+ byteLenData);
        try {
            sleep(100);
        } catch (InterruptedException e) {
            System.out.println("Error : " + e.getMessage());
        }

        buffer = new byte[byteLenData];
        int count = 0;
        while(count < byteLenData) {
            count += is.read(buffer, count, byteLenData - count);
        }
```

Figure 8: 接收數據

  - 圖 9 為傳送數據

```
if(type == true) { // muscle data
    dos.writeBoolean(true);
    dos.writeDouble(muscleData);
}
else { // byte data
    dos.writeBoolean(false);
    dos.writeInt(buffer.length);
    System.out.println("Send image file length: "+ buffer.length);

    try {
        sleep(100);
    } catch (InterruptedException e) {
        System.out.println("Error : " + e.getMessage());
    }

    os.write(buffer);
    os.flush();
    System.out.println("Send image to Client...");
    try {
        sleep(100);
    } catch (InterruptedException e) {
        System.out.println("Error : " + e.getMessage());
    }
    System.out.println("Send image FINISH.");
}
```

Figure 9: 傳送數據

- 以下為 Client 端之程式碼片段
  - 圖 10 為建立連線

```
public Client() {
    try {
        socket = new Socket(serverName, serverPort);
        System.out.println("Client started on port " + socket.getLocalPort() + "...");
        System.out.println("Connected to server " + socket.getRemoteSocketAddress());

        os = socket.getOutputStream();
        dos = new DataOutputStream(os);

        client = new ChatClientThread(this, socket);
        thread = new Thread(this);
        thread.start();
    } catch (IOException e) {
        System.out.println("Error : " + e.getMessage());
    }
}
```

Figure 10: 建立連線

  - 圖 11 為傳送數據
  - 圖 12 為接收數據

```
private void setByteFile(){
    System.out.println("Client Thread Lock!");
    lock.lock();
    try{
        // 鎖空byte，有東西才解
        System.out.println("Client Thread Await!");
        condition.await();

        writeBuffer = new byte[CameraSurfaceView.getByteCount()];
        writeBuffer = CameraSurfaceView.getByteFile();
    }catch (InterruptedException e){
        e.printStackTrace();
    }finally {
        lock.unlock();
        System.out.println("Client Thread UnLock!");
    }

    new Thread(new Runnable()
    {
        @Override
        public void run()
        {
            updateTask();
        }

    }).start();
}
```

```
try {
    setByteFile();

    if(lock.tryLock()){
        try{
            byteFile = writeBuffer;
            byteCount = byteFile.length;
        }finally {
            lock.unlock();
        }
    }

    try {
        thread.sleep(100);
    } catch (InterruptedException e) {
        System.out.println("Error : " + e.getMessage());
    }

    // false 表示傳送影像array
    dos.writeBoolean(false);
    System.out.println("Send Boolean: FALSE");

    dos.writeInt(byteCount);
    System.out.println("Send image file length: " + byteCount);
    try {
        thread.sleep(100);
    } catch (InterruptedException e) {
        System.out.println("Error : " + e.getMessage());
    }

    // 拍下影像downsize!!不需要這麼高
    System.out.println("Start Send image file");

    os.write(byteFile);
    os.flush();
    System.out.println("Send image to Server..." + byteFile.length);
```

Figure 11: 傳送數據

```
boolean type = dis.readBoolean();
if(type == true){ // true 表示收到muscleData
    muscleData = dis.readDouble();
}
else{ // false 表示收到影像array
    byteLenData = dis.readInt();
    System.out.println("Receive image file length: " + byteLenData);
    try {
        sleep(100);
    } catch (InterruptedException e) {
        System.out.println("Error : " + e.getMessage());
    }

    buffer = new byte[byteLenData];
    int count = 0;
    lock.lock();
    while(count < byteLenData){
        count += is.read(buffer, count, byteLenData - count);
    }
    lock.unlock();
    // rand.write(buffer); //Writes bytes to output stream
    System.out.println("Receive image from Server..." + count);

    try {
        sleep(100);
    } catch (InterruptedException e) {
        System.out.println("Error : " + e.getMessage());
    }

    // rand.close();
    System.out.println("Receive image FINISH.");
}

if(lock.tryLock()){
    try{
        readBuffer = buffer;
    }finally {
        lock.unlock();
    }
}
```

Figure 12: 接收數據

2.3.2 藍芽功能

- 用途為建立與肌肉感測器之間的通訊。
- 圖 13 為將藍芽連線至裝置之程式碼片段。

```
                //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 13: 藍芽連線之程式碼片段

- 圖 14 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。

```
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("[^(a-zA-Z0-9\\u4e00-\\u9fa5)]", "");


            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
            Session session = Session.getSession();
            session.put("data", readMessage);
        }
```

Figure 14: 藍芽接收之程式碼片段

2.3.3 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 15 為將感測器訊號由 Arduino 端傳至 Android 手機端之程式碼片段。

```
int analogPin = A0;
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 15: Arduino 之程式碼片段

3 肌肉感測訊號分析與影像處理

本章節著重介紹肌肉感測訊號分析與影像處理技術。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 16 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。

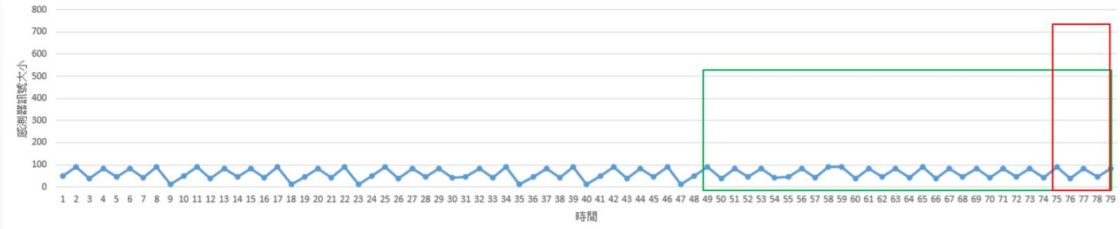


Figure 16: 陣列與訊號關係示意圖

  - 如圖 17 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];
    }
```

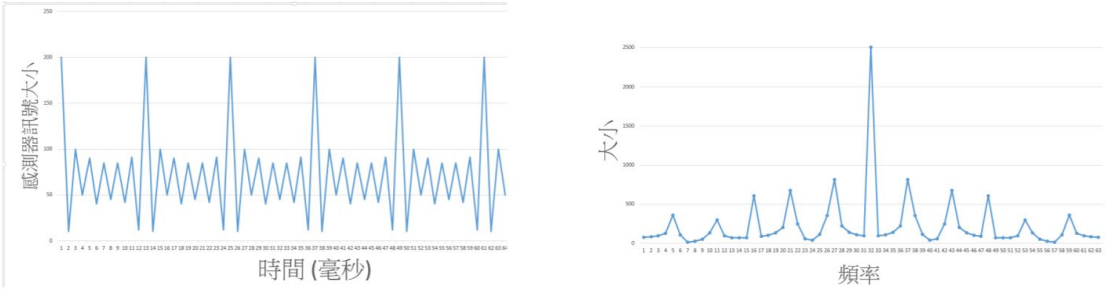
(a) 設兩個分別為大跟小的陣列

```
/*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");
    }
```

(b) 開始執行

Figure 17: 部分程式碼片段

– 如圖 18 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) 轉換前之訊號　　(b) 轉換後之訊號

Figure 18: 訊號頻率

– 如圖 19 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 19: 快速傅立葉之程式碼片段

– 如圖 20 所示，將速度分為三個階段，越快的會加分的越快。

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

            //longlogs
            if(( a.fftcalculate(longlogs)[1]>300)&&( a.fftcalculate(longlogs)[1]<700)){
                //note.setText(" ");
                //note.setText("fast" + " ");

                count++;
            }

            if(( a.fftcalculate(longlogs)[2]>300)&&( a.fftcalculate(longlogs)[2]<700)){
                //note.setText(" ");
                //note.setText("very fast" + " ");

                count++;
            }

             if(( a.fftcalculate(longlogs)[3]>300)&&( a.fftcalculate(longlogs)[3]<700)){

                //note.setText(" ");
                //note.setText("very fast" + " ");

                count++;
            }
        }
        quickbar.setProgress(count);
    }
```

Figure 20: 速度分級之程式碼片段

3.2 影像處理

3.2.1 使用工具

- 使用 Google Firebase ML Kit。ML Kit 為 Google 在 2018 年新開發出專為行動裝置所設計的 SDK。主要將 Google Cloud Vision API、Mobile Vision、Tensorflow Lite 三部分組合而成。本次專題中主要使用到 ML Kit 提供的影像相關之 Vision API 的 Face Detection 部分。

- Face Detection API 之特點

  1 能快速辨識並且標記出臉部的特徵點：能辨識出各臉部中五官之位置及人臉或五官之輪廓。

  2 辨識臉部表情：能辨識臉部是否在微笑、是否閉上眼睛。

  3 人臉追蹤：辨識影片曾出現過的人臉，並給予 ID 以便讀取關於某面孔之資料。

  4 即時處理：即使在沒有網路服務下仍能快速處理辨識。在動態畫面的處理上效能更佳。

3.2.2 特徵點提取及臉部偵測技術

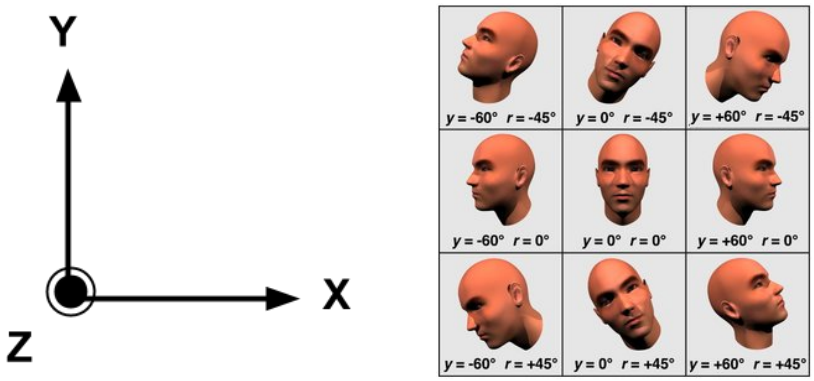
- 臉部偵測概述

  1 Face Orientation

  - 臉部偵測的第一步為偵測臉部之位置，並找出其轉向的角度。(圖 21 所示)
  - API 中所檢測到的為 Euler Y 以及 Euler Z。預設回傳 Euler Z，如果需取得 Euler Y 則需將檢測器之設定做修改。

  2 Landmarks

  - 此為面部中的特徵點，例如左右眼、鼻頭皆為此。
  - 在此 API 中，偵測時並沒有先以特徵點來做為面部偵測的基礎，而是先將臉部整體偵測出來後從中提取出所需要的 Landmarks。
  - 其中因不同的 Euler Z 所能偵測的也會有所不同。詳細如下圖 22 所示。



(a) The coordinate system (b) Pose angle examples

Figure 21: Pose angle estimation

Euler Y angle	Detectable landmarks
< -36 degrees	left eye, left mouth, left ear, nose base, left cheek
-36 degrees to -12 degrees	left mouth, nose base, bottom mouth, right eye, left eye, left cheek, left ear tip
-12 degrees to 12 degrees	right eye, left eye, nose base, left cheek, right cheek, left mouth, right mouth, bottom mouth
12 degrees to 36 degrees	right mouth, nose base, bottom mouth, left eye, right eye, right cheek, right ear tip
> 36 degrees	right eye, right mouth, right ear, nose base, right cheek

Figure 22: Landmark 與 Euler Z 之關係圖

3 Contours

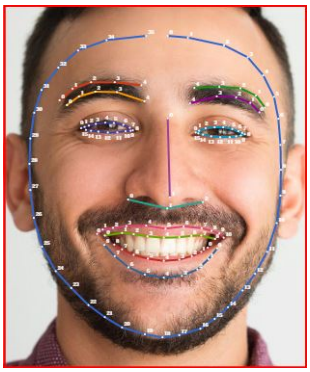
將代表面部特徵之點組合而形成的輪廓，而所偵測到的特徵輪廓都由特定數量的點來表示，而當面部中總共 133 個 Contours 皆取得時，每個特徵輪廓皆會有特定的編號。(圖 23 所示)

Face oval	36 points	Upper lip (top)	11 points
Left eyebrow (top)	5 points	Upper lip (bottom)	9 points
Left eyebrow (bottom)	5 points	Lower lip (top)	9 points
Right eyebrow (top)	5 points	Lower lip (bottom)	9 points
Right eyebrow (bottom)	5 points	Nose bridge	2 points
Left eye	16 points	Nose bottom	3 points
Right eye	16 points		
Left cheek (center)	1 point		
Right cheek (center)	1 points		

(a) 特徵輪廓數量

Indexes of feature contours	
0-35	Face oval
36-40	Left eyebrow (top)
41-45	Left eyebrow (bottom)
46-50	Right eyebrow (top)
51-55	Right eyebrow (bottom)
56-71	Left eye
72-87	Right eye
88-96	Upper lip (bottom)
97-105	Lower lip (top)
106-116	Upper lip (top)
117-125	Lower lip (bottom)
126, 127	Nose bridge
128-130	Nose bottom (note that the center point is at index 128)
131	Left cheek (center)
132	Right cheek (center)

(b) Landmark 的編號



(c) 所偵測之結果

Figure 23: 特徵點相關資訊

- 特徵點之校對

  在實際運行中發現該實驗之方法為將臉部區域偵測出來後，將其集中訓練的平均人臉及 Landmarks 匹配至區塊上。再以該影像中的 pixel 進行特徵點的校準，反覆循環直至特徵點誤差趨於平衡。(圖 24)

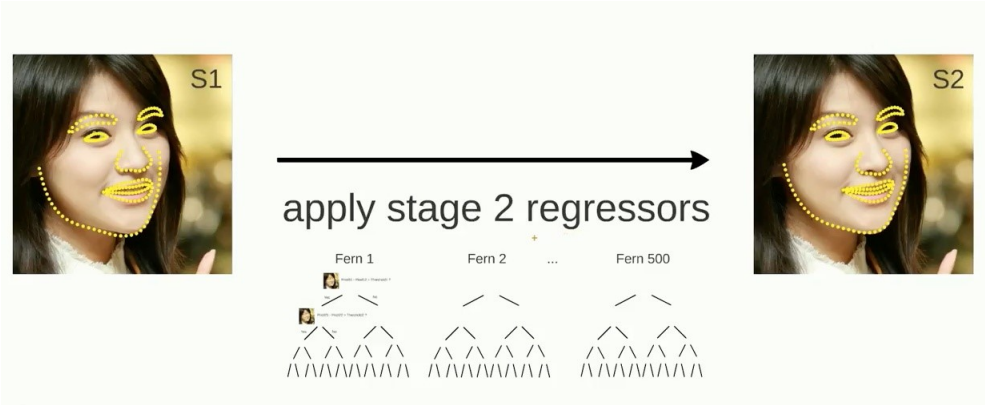


Figure 24: Landmarks 之校對及訓練

3.2.3 ML Kit 運行步驟

- 前置步驟及基本需求

  1 將所需之 firebase 檔案新增近專案中

  2 為了使面部偵測更加準確，在圖像中至少需要 100x100 像素以偵測到臉部區域，如要更精確地獲得輪廓資訊則需要 200x200 像素。

- 運行步驟

  1 建立檢測器 FirebaseVisionFaceDetectorOptions

  - 在將人臉檢測應用於圖像之前，可更改人臉檢測器的默認設置。例如是否識別 Landmarks 亦或是在模式中對於速度及準確度兩者的優先。
  - 以圖 25 解釋，在此檢測器中因為即時影像傳送，故以速度作為優先選擇。

```
FirebaseVisionFaceDetectorOptions options =
        new FirebaseVisionFaceDetectorOptions.Builder()
                .setClassificationMode(FirebaseVisionFaceDetectorOptions.FAST)
                .setContourMode(FirebaseVisionFaceDetectorOptions.ALL_CONTOURS)
                .setLandmarkMode(FirebaseVisionFaceDetectorOptions.ALL_LANDMARKS)
                .build();
```

Figure 25: 程式碼截圖

2 將資料轉成檢測器所需之圖像物件 FirebaseVisionImage

– FirebaseVisionImage extends Object

– 能將不同的資料型態轉變為可用於偵測器的物件，但不同的資料性態在轉換時會擁有不同的需求，例如 ByteBuffery 在轉換時需要先將物件轉換成 FirebaseVisionImageMetadata 物件來進行描述。(圖 26)

```
FirebaseVisionImage firebaseVisionImage = FirebaseVisionImage.fromBitmap(bitmap);
```

Figure 26: 程式碼截圖

3 執行檢測器 FirebaseVisionFaceDetector(圖 27)

```
detector = FirebaseVision.getInstance().getVisionFaceDetector(options);
detector.detectInImage(firebaseVisionImage)
        .addOnSuccessListener((OnSuccessListener) (firebaseVisionFaces) → {
                getFaceResults(firebaseVisionFaces);
        })
        .addOnFailureListener((e) → {
                Log.e(TAG,  msg: "fail detectInImage: " + e);
        });
```

Figure 27: 程式碼截圖

4 取得偵測完之物件 FirebaseVisionFace(圖 28) 在 FirebaseVisionFace 便

```
private void getFaceResults(List<FirebaseVisionFace> firebaseVisionFace) {
    int counter = 0;
    for(FirebaseVisionFace face : firebaseVisionFace){
        System.out.println("------------Face Detect Result-------------");
        System.out.println(counter);
        Rect rect = face.getBoundingBox();

        FaceGraphic faceGraphic = new FaceGraphic(graphicOverlay, face, CameraSource.CAMERA_FACING_FRONT);

        graphicOverlay.add(faceGraphic);

        counter = counter + 1;
    }

}
```

Figure 28: 程式碼截圖

存取了在臉部改動所需要的臉部輪廓座標以及面部特徵輪廓座標資料。

3.2.4 面部更改

- 取得特徵資料之座標

  運用 FirebaseVisionFaceLandmark 物件紀錄所取得的臉部不同五官資料以取得右眼中心資料為例，並取得 x,y 座標。(圖 29)

```
FirebaseVisionFaceLandmark landmark = face.getLandmark(FirebaseVisionFaceLandmark.RIGHT_EYE);
if (landmark != null) {
    FirebaseVisionPoint point = landmark.getPosition();
}
```

Figure 29: 程式碼截圖

- 圖片合成

  將欲合成之圖片轉換成 bitmap 檔案後創立與原圖大小相同之新 Bitmap 及其 Canvas。在其中進行圖片合成。(圖 30)

```
FirebaseVisionPoint point;
if (background == null) {
    return null;
}

int bgWidth = background.getWidth();
int bgHeight = background.getHeight();
//create the new blank bitmap
Bitmap newbmp = Bitmap.createBitmap(bgWidth, bgHeight, Bitmap.Config.ARGB_8888);
Canvas cv = new Canvas(newbmp);
//draw bg into
cv.drawBitmap(background,  left: 0,  top: 0,  paint: null);//在 0，0座標開始畫入bg
//draw fg into
FirebaseVisionFaceLandmark landmark = face.getLandmark(FirebaseVisionFaceLandmark.RIGHT_EYE);
if (landmark != null) {
    point = landmark.getPosition();
    cv.drawBitmap(foreground, point.getX(), point.getY(),  paint: null);//從任意位置畫
}
//save all clip
cv.save(Canvas.ALL_SAVE_FLAG);//保存
//store
cv.restore();
return newbmp;
```

Figure 30: 程式碼截圖

4 實驗結果

本章節將實際展示此專題成果，並依序分析各項功能在實驗環境下的結果，包含：網路傳送延遲時間之計算評估、影像處理精確度及延遲之評估、肌肉感測器延遲評估。此外，講述實作的過程，以及在實作途中所遭遇到的問題與解決方式，並附上遊戲者體驗之評價。

4.1 實驗環境

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：Ardunio 1.6.12
- Android 程式開發環境：Android Studio 2.3

4.2 網路傳送延遲時間之計算評估

網路傳送延遲時間之計算評估

4.3 影像處理精確度及延遲之評估

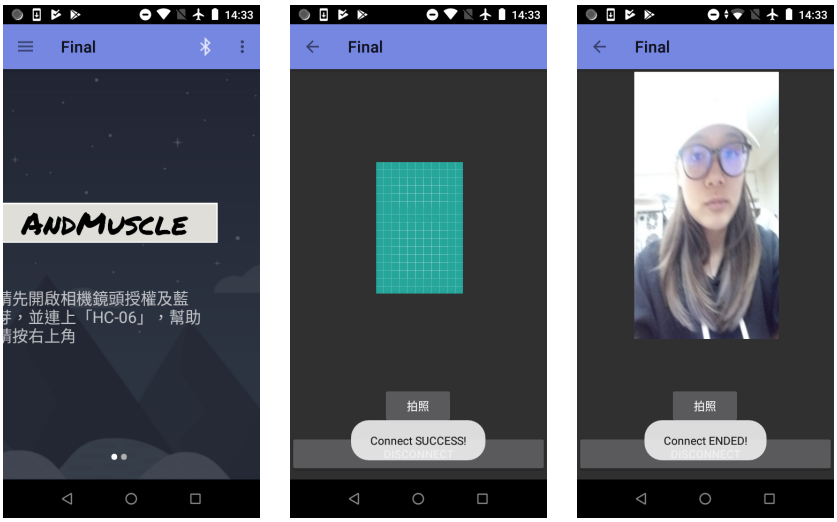
影像處理精確度及延遲之評估

4.4 肌肉感測器延遲評估

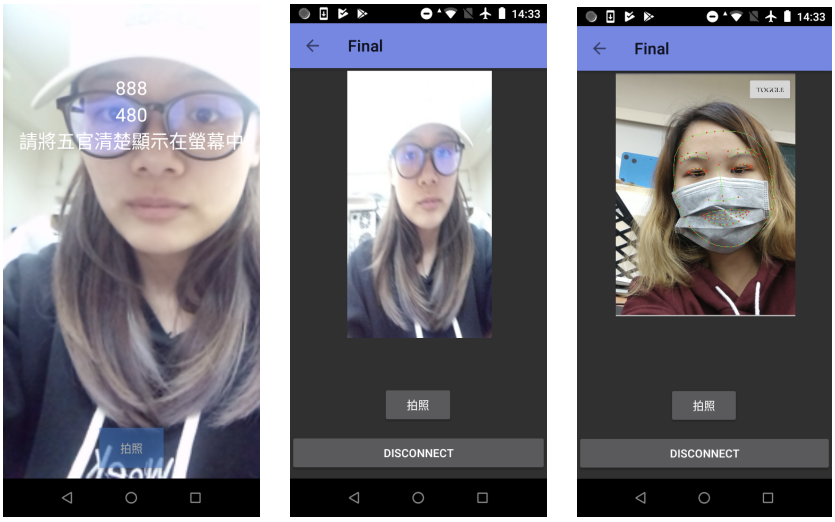
肌肉感測器延遲評估

4.5 專題結果展示

- 遊戲主頁面及開始、結束畫面



- 遊戲中畫面



4.6 遊戲者體驗

遊戲者體驗之評價

5 結論

在本專題中，我們所開發的遊戲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及聯絡情誼之意義。此外，本系統包含了多項技術，包括：使用快速傅立葉轉換分析肌肉感測器的訊號，利用 OpenCV 技術進行影像處理，透過 TCP 架構之網路連線即時傳送。

本系統所使用肌肉感測器可以貼在任何明顯肌群部位，可以更符合遊戲體驗者需求，比如從手臂更換到小腿，即可有更多不同玩法。未來，希望能增設遊戲招式之難度，且持續優化影像處理後之畫面，以及網路傳輸影像、數據之效率，提升整體遊戲品質。

6 工作分配

姓名	內容	說明
王佳君	網路連線	建構 TCP 網路架構，建立伺服器端及客戶端連線
	肌肉感測器	讀取 Arduino 數據，分析波形、判斷數據
	系統整合	整合成完整系統
	實驗測試	網路傳送與肌肉感測器延遲時間之評估
	報告文件	使用 Latex 撰寫專題報告及海報
林令婕	影像處理	利用 OpenCV 技術，變更影像色調及拼貼圖案等
	系統整合	整合成完整系統
	實驗測試	影像處理精確度及延遲之評估
	報告文件	撰寫專題報告、海報內容

References

[1] 首頁介面效果

https://dotblogs.com.tw/starhao/2016/10/08/012050

[2] 快速傅立葉轉換

https://www.youtube.com/watch?v=k48iUhNkqh4

https://reurl.cc/M7047K

[3] 藍芽連線

https://home.gamer.com.tw/creationDetail.php?sn=3671289

[4] Session

https://blog.csdn.net/ZDW86/article/details/35795125

[5] TCP 連線

https://github.com/marselsampe/java-socket-tcp-sample