National Taiwan Ocean University

Department of Computer Science and Engineering

The Monster Hunter ZERO: the AR interacting system run on the Android system and muscle sensor

00457036 CSE4A Sun, Yun-Han
00457141 CSE4B Chou, Hsi-Min

Professor: Chang, Chin-Chun

2019 / 12 / 14

Abstract

Our project, an AR game system, is developed from a mobile device and a muscle sensor. The game system uses a camera hidden in the mobile device to observe the environment and combines the technique of computer vision to decide where the virtual monsters should be. The player must change his position and the angle of the camera in order to find the virtual monsters. Moreover, by measuring the player's muscle activity based on the muscle sensor, players can perish the virtual monsters by altering the strength and frequency of the muscle. In conclusion, our game system can not only make players move around, but also make them achieve the target of loosening up and training muscles. In addition, we design different game levels by changing the strength and the frequency of muscle force in order to make the game more abundant.

Table 1: Division table of work tasks

Name	Job Content
Sun, Yun-Han	muscle sensor access, data analysis, energy bar design, bluetooth function, system combination, report document
Chou, Hsi-Min	scene analysis, background detection, virtual monsters stitch on scenes, system combination, report documen

Contents

List of Figures

List of Tables

Chapter 1

Introduction

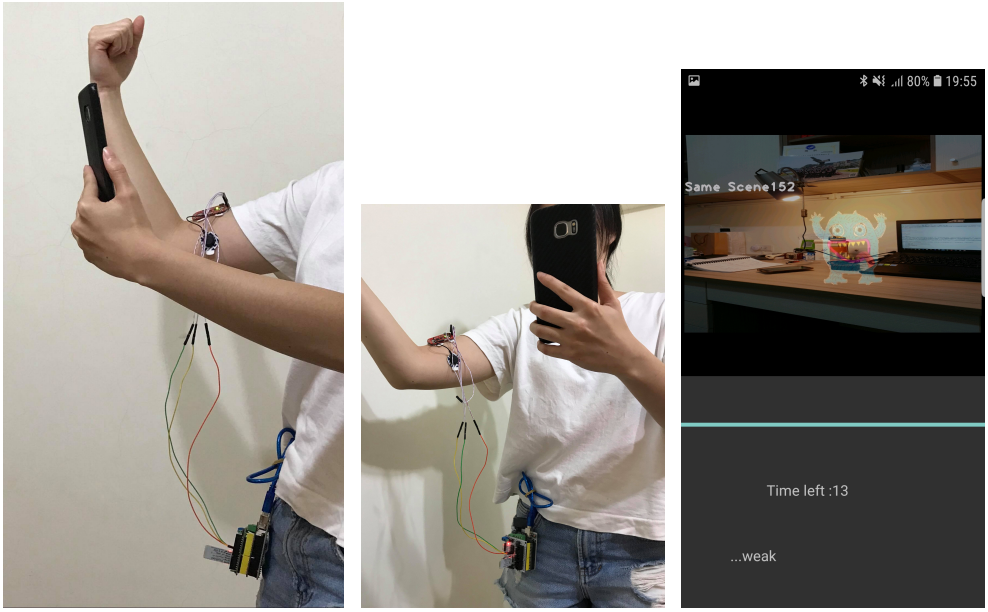
1.1 Motivation and Purpose

Recently, people often sit and hold the same posture or position for an extended period of time, leading to muscle sourness or disease-causing. However, people detect the importance of exercise gradually, and start building habits of weekly workout. Unfortunately, not everyone has time to go to the gym or to the park for exercise and sports. Furthermore, some people resist boring workout machines, for example, treadmills, spinning bikes, and so on, even participation in outdoor activities. In order to resolve those problems, we design a game system which combines entertainment and exercise for the benefits of users to play where and when they want. During the process of the game, the system can either reach the effect of loosening up muscles or gaining the accomplishment. Our game application is developed from Android mobile device, Arduino and muscle sensor, which can bring the unprecedented levels of interest. We implement technique of augmented reality on the game screen, leading players to experience a real-world environment during the game. Besides, we provide training mode so that players can engage in physical exercise in ordinary circumstance.

1.2 Brief Introduction of System

The main function provided by our system is that the player can combine with the real scene from the camera through augmented reality. In addition, the virtual monster chamges by different angles of the camera, by measuring the strength of the muscle based on the muscle sensor. The levels of difficulty in game increased by the relaxation and tautness of muscle, making the game more interesting.

First, we log into the game screen in order to find the virtual monster's hiding place, and use the technique of computer vision to decide whether the scene is distinct. If the scene is complicated, the virtual monster appears. In order to make the virtual monster stay in its original position when the mobile device moves away and moves back, we compare the viewpoint and the distance of the original scene and the next scene. When the monster appears, the hint information about how to attack the virtual monster shows up, players can follow the instructions to manipulate the muscle sensor, for example, loosening up the muscle or raising strength in a limited time. Not only the signal of the strength is needed, but also the variation of the frequency, so we analyze player's movement in order to obtain the shrink rhythm of muscle based on the technique of signal analysis like Fast Fourier Transform. If the manipulation is correspond to the demands, the game system views the player's motion as the correct operation, and the player can perish the virtual monster successfully.



(a) Game demonstration 1 (b) Game demonstration 2 (c) Game screen

Figure 1.1: Display of devices

1.3 Organization of Report

The introduction of software and hardware framework in our system is presented in Chapter 2 of the project report. For hardware framework, we introduce the apperance of system, the sensors we use and the sensor

technology we apply. For software framework, which includes flowcharts and every functions, we explain each function's application, sequence of processes and implementation. The process of algorithms in the software system is introdued in Chapter 3. Chapter 4 presents the methods of research and the experimental data during the implementation. Finally, the conclusion of our project is displayed in Chapter 5.

Chapter 2

Framework of Software and Hardware

The research content includes the introduction of software and hardware framework in our system, the specifications of the sensors and the sensor technology we use.

2.1 Infrastruction of Software and Hardware

As Figure 2.1, the framework of the system is seperated into two subjects. Hardware is on the leftside, which includes Arduino and the muscle sensor; software is on the rightside, which includes the mobile device and OpenCV module. As Figure 2.2, in the sensor part, we detect the data of strength from muscle force and send it back to the system; in the mobile device part, we calculate and analyze the return data from the sensor, also apply machine vision module for image processing. The sensor connects to the moblie device via the bluetooth module.

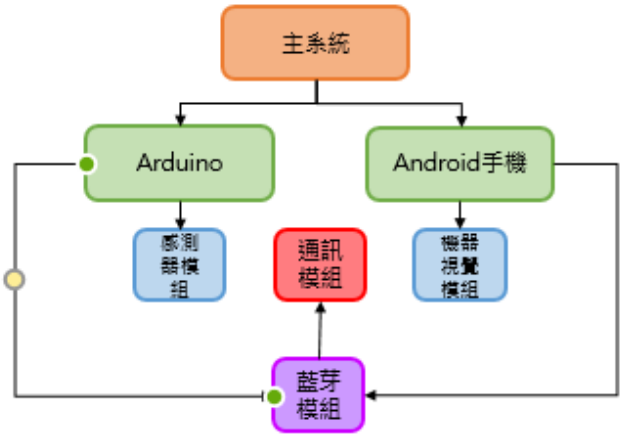


Figure 2.1: Infrastruction of software and hardware

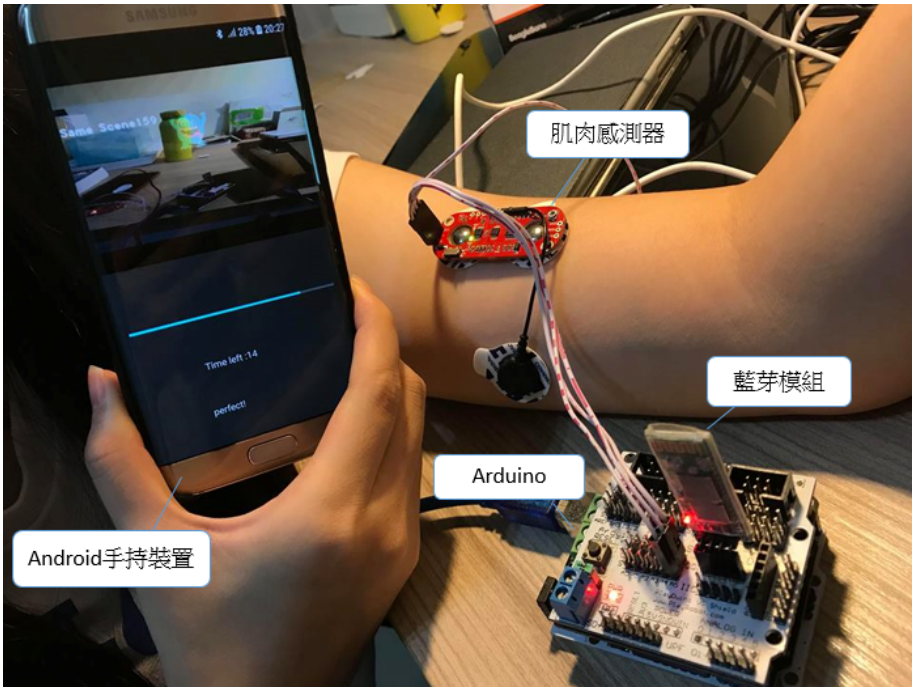


Figure 2.2: Operating instruction

2.2 Framework of Hardware

2.2.1 Introduction of System's Appearance

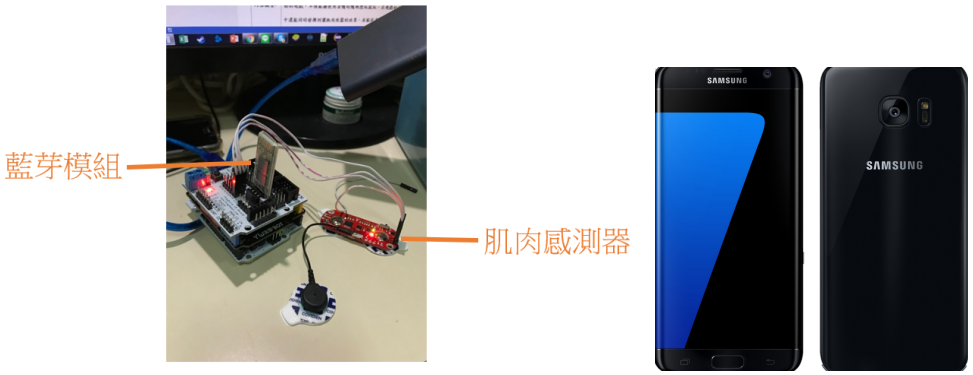
Shown as Figure2.3 and Figure 2.4, the appearance of our system consists of a bluetooth device, a power supply, a muscle sensor and an Android mobile device, as following:

1. Power Supply: Using a power bank because of its stable electric current and the feature of reusable in order to provide power source.

2. Bluetooth Device(Figure 2.4a): The bluetooth module can deliver and receive messages via the bluetooth on the moblie device.

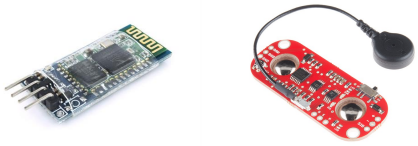
3. Muscle Sensor(Figure 2.4b) Single Supply: +3.1V to +5V、Polarity reversal protection.
   RAW EMG Output: A popular request from grad students, the MyoWare now has a secondary output of the RAW EMG waveform.
   LED Indicatorsz: Adding two on-board LEDs one to know when the MyoWare's power is on and the other will brighten when the muscle flexes.

4. Android Device(Figure 2.3b): We apply the camera and the bluetooth on the mobile device in our project.



(a) Figure of Arduino device (b) Android device

Figure 2.3: Appearance of the game device



(a) Bluetooth module (b) Muscle sensor

Figure 2.4: Hardware device of operating Arduino and MyoWare muscle sensor

2.2.2 Sensor Technology

The main technology for designing our system can see as the following:

1. Muscle Sensor:
   Measuring muscle activity by detecting its electric potential, referred to as electromyography (EMG), has traditionally been used for medical research. When the brain gives an instruction about shrinking muscle, it delivers a route reminding the muscle to start gathering the muscle units. When using more energy, it produces more muscle units to gather more strength. When gathering more muscle units, it produces more changes of electrical activity of muscle. MyoWare analyzes the electrical signals and outputs them to indicate how we use our muscle. The more force you make, the more voltage MyoWare outputs. Although EMG can already detect the electrical signal of muscle, MyoWare can make it become more meaningful and more humanity.

   The flowchart of the transmit signal from the sensor is shown as Figure 2.5, we can get signal once in 0.15 seconds, and set 1.2 seconds as an interval to analyze the variation. We will explain the algorithm of signal analysis more specifically later in Chapter 3.

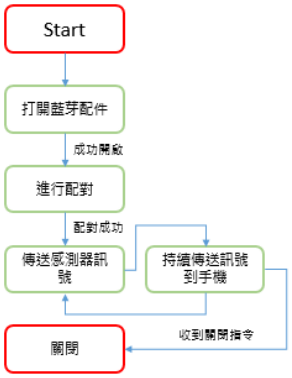


Figure 2.5: Flowchart of sensor

2. Connection of Bluetooth in Android: Applying the concept of wireless comnunication to pass messages between the mobile device and the muscle sensor.
   Process of operation:

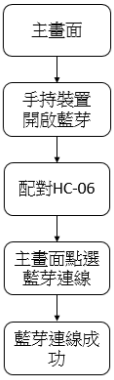


Figure 2.6: Flowchart of opening bluetooth

Shown as Figure 2.6, see the process of opening the bluetooth as below:
Process 1: Open the bluetooth in the mobile device.
Process 2: Click ”HC-06” for pairing.
Process 3: Return to the main screen of the application, and click the

bluetooth connection.
Process 4: Message of "Connect Successfully" shows up.

3. Android Camera: We apply JavaCameraView provided from OpenCV Android SDK to call the camera hidden in the mobile device. JavaCameraView inherits SurfaceView built-in Android, but it neesd to call onCameraFrame in order to obtain every image. So OpenCV, a program library, is able to process images later on. OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Our project, we apply the detection of feature points and image stitching with OpenCV, making the virtual monsters appear on the camera and leading to the result of augmented reality. Firstly, we search for feature points from our target image based on AKAZE in the program. Then we pair the feature points we collected before with the best match points from the camera. We will discuss how we analyze the scene in details in Chapter 3.

2.3 Framework of Software

The software part includes the bluetooth, Arduino and the game funtion. The applications and the processes of operation are presented in order as the following.

2.3.1 Bluetooth Function

Application: Establishing the communication with the muscle sensor. A part of source code about the bluetooth connect to the mobile device is shown as Figure 2.7. Figure 2.8 is about the bluetooth receives signals stored by the byte format, and keeps the receiving numbers in the session in order to calculate it easily afterwards.
Process of Operation:

```
                //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: Source code of connecting bluetooth

```
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 2.8: Source code of receiving bluetooth

2.3.2 Arduino Function

First, we plug the output circuit from the muscle sensor into the specific pin location. Then we stick the adhesive tape on a particular position of our arm. The source code of signals we transmit from Arduino to Android device is shown as Figure 2.9. As Figure 2.10, the mobile device can read the data imported from the sensor clearly.

```
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: Source code of Arduino

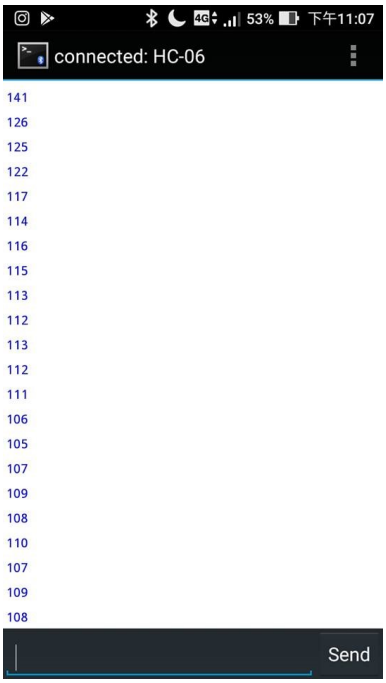


Figure 2.10: Return data from muscle sensor

2.3.3 Game Function

(1)Hunting Monster Mode
Process of operation: First, the player opens the camera, then starts to find

the virtual monster's hiding place. After the virtual monster is found, follow the instructions to attack it.

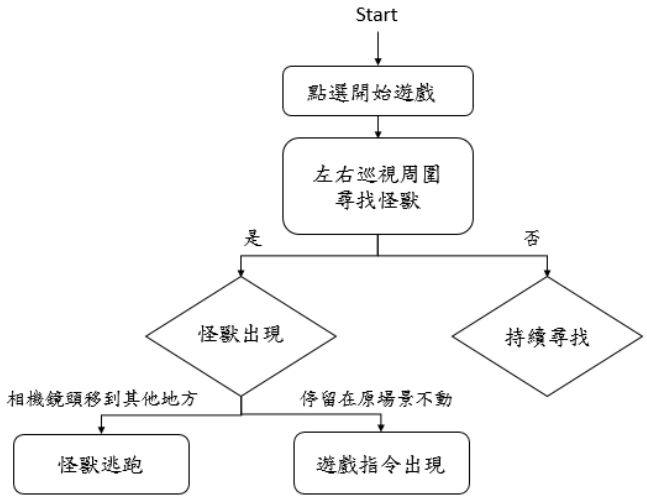


Figure 2.11: Process of operating main game

As Figure 2.11, the main game process is shown as the following:
Process 1: Enter to the game screen.
Process 2: Find where the virtual monster hides in the circumstance.
Process 3: When the virtual monster appears, the game instruction shows up. Attack the virtual monster with the muscle sensor.
Process 4: In the period of attacking, player must sustains the balance of his hands. If player leaves the scene, the virtual monster escapes.
Process 5: If player finishes the instruction in the limtied time, the player successes; in contrast, the player fails.

(2)Training Mode
Process of operation: Players have two choices for different modes, one is the power mode, and the other is the speed mode. Players are obliged to finish the assigned targets in the limited time. Including:
1. Power Mode: We set a force standard, the data views as valid if it goes beyond the standard. We collect the data once in 0.15 seconds, the game ends when we collect enough energy or when the limited time is up.
2. Speed Mode: Filtering the data and computing the frequency of one time interval based on Fast Fourier Transform. Viewing the data as valid if the speed is not zero, the game ends when we collect enough energy or when the

limited time is up.

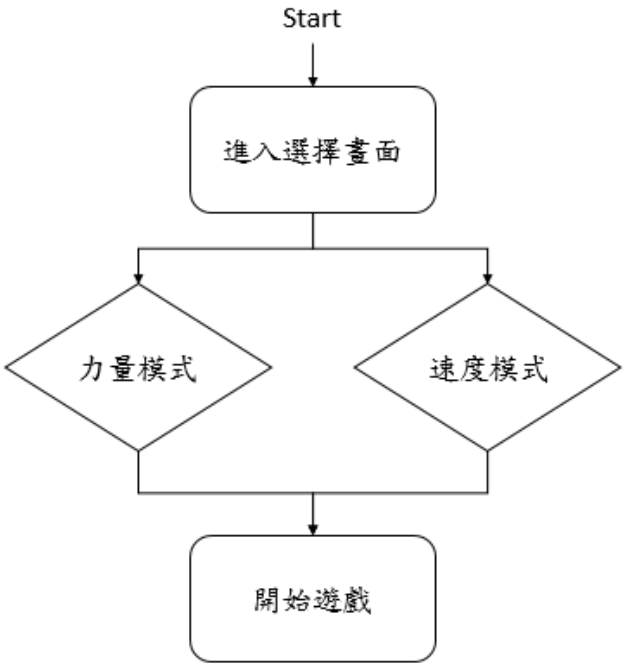


Figure 2.12: Flowchart of challenge mode

As Figure 2.12, the process of challenge mode is shown as the following:
Process 1: Enter to the challenge mode and choose the game level.
Process 2: The game level is distributed to power mode and speed mode.
Process 3: Click it and enter to the game screen.

Chapter 3

Scene Analysis and Signal Analysis of Muscle Sensor

In this chapter, we introduce scene analysis and signal analysis of the muscle sensor. We describes the purpose of the functions, the process of the algorithms and the illustration of the source code passages.

3.1 Analysis of MyoWare Muscle Sensor

Purpose:
It is used for analyzing the signal transmited from the mobile device to the muscle sensor. We apply Fast Fourier Transform to attain the force and the frequency of the signal after we filter the useless signal.
The Process of algorithm:
As Figure 3.1, Array 1 is the short array that updates the latest information (red). Array 2 is the long array that updates the latest signal (green). We will explain the different computing works between two arrays later.

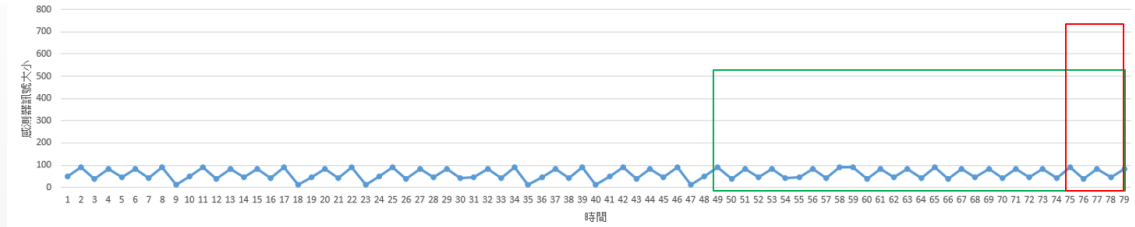


Figure 3.1: Connection between matrix and signal

As Figure 3.2 and Figure 3.3, the small array decides whether the signal is change after the big array is filled up. If the signal changes, we decide the rhythm of the variation based on the big array.

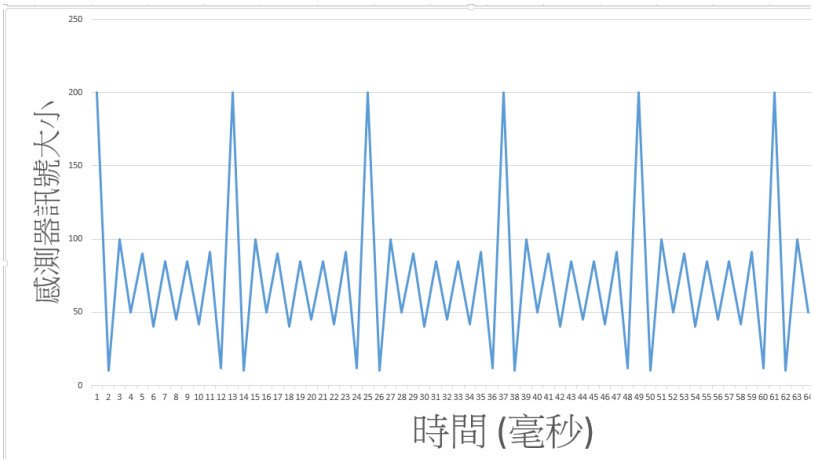
```
/*陣列1*/

    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: Set a big matrix and a small matrix

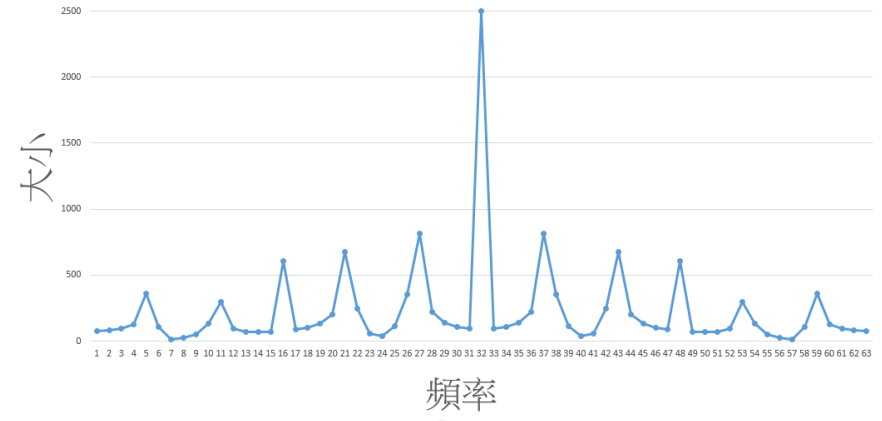
```
/*開始*/
double move=1;/*判斷已沒有動的變數*/
if(longfftcount>=64) {/*第二陣列滿*/

    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: Source ccde of starting implementing analysis

We can see from Figure 3.4a and Figure 3.4b, the signal after transforming is much more easy to determine the frequency than before. So it is more easy to define the speed of muscle relaxation in the game algorithm.



(a) Signal before transformation



(b) Signal after transformation

Figure 3.4: Signal transformation

Figure 3.5 illustrates the main function of Fast Fourier Transform. It is the transformation between time area (space area) and frequncy area from the signal. We decide whether the user is moving based on the result after the transformation, and decide the user's motion based on the longer one. The purpose by doing so is to reduce the delay time.

```
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: Source code of Fast Fourier Transform

As we can see from Figure 3.6, we seperate the speed into three stages, the faster you achieve the more points you earn.

```
    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: Speed levels

3.2 Scene Analysis

Purpose:

1. Purpose 1: Searching for the scene where the virtual monster can hide. Computing the density of the scene with edge detection based on Canny Algorithm. After that, we detect whether the feature points are enough by AKAZE. The more feauture points show, representing the scene is more complicated, so the virtual monster is able to stitch on the camera screen.

2. Purpose 2: Combination of the virtual monster and augmented reality. Comparing the feature points of two different frames in order to analyze whether the mobile device moves to another scene. If the Match Points of the next frame are not enough with the original frame that the virtual monster appears, the virtual monster vanishes.

Process of algorithm:
(1)Find the scene

Step 1: First, we get an image and reduce the image noise by applying a Gaussian Blur.

Step 2: Applying Canny Algorithm for edge detection, if the strength is excess of the degree of threshold, entering to the computation of feature points.

Step 3: Input two images continuously and decide if the ”special points” are enough in this scene. So-called ”special points”, is the meaning of when computing with KnnMatch, the distance of finding the feature points between two images with AKAZE is very clsoe.

Step 4: If the ”special points” are enough, the virtual monster is able to stitch to the feature points in the scene.

As Figure 3.7, we only detect the feature points and compute it with the middle area of the whole screen, in order to speed up the image processing. At first, we divide with the length and the width of the screen by 2 to accelerate the speed, but this method reduces the number of the feature points. So we change the method to measure the screen aiming at the middle part, narrowing the range and not modifying the number of the feature points, leading the comparison between two frames more precisely.

```
//使畫面加速
//Imgproc.resize(templ, templ, new Size(templ.width() / speedup, templ.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)(templ.width()*0.25), (int)(templ.height()*0.25), templ.width()/2, templ.height()/2);
Mat subTempl = new Mat(templ, 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: Source code of accelerated screen

Figure 3.8a and Figure 3.8b show the number of the feature points and the actual position.

(a) Feature points <10 (b) Feature points >10

Figure 3.8: Whether feature points are enough

(2)Combination with augmented reality

Step 1: If the "special points" are enough, we stitch the four corners of the virtual monster image to the four feature points by applying Homography. We need to multiply the feature points of the coordinates by a special matrix, so that the coordinates are correspond to the original position.

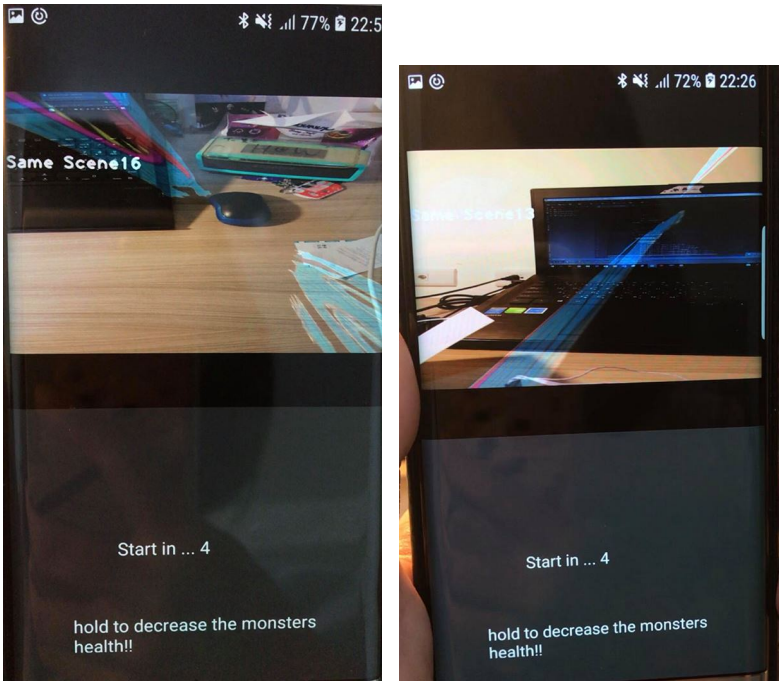
Step 2: Write down the feature points before the transformation, in order to compare to the next screen.

Step 3: When the hiding place for the virtual monster is found, we start to decide whether the next scene is correspond with the scene that the virtual monster hide originally. Compute both images with KnnMatch, if the Match Point is bigger than 12, we view the scene as the same one.

Step 4: Later on we convert the feature points we store before to the coordinates which are correspond with the original screen position. Then calculating the homography of the scene that stores the virtual monster with the present scene. Above all, we stick the virtual monster on it.

Step 5: If the scene is different with the original one, the virtual monster escapes.

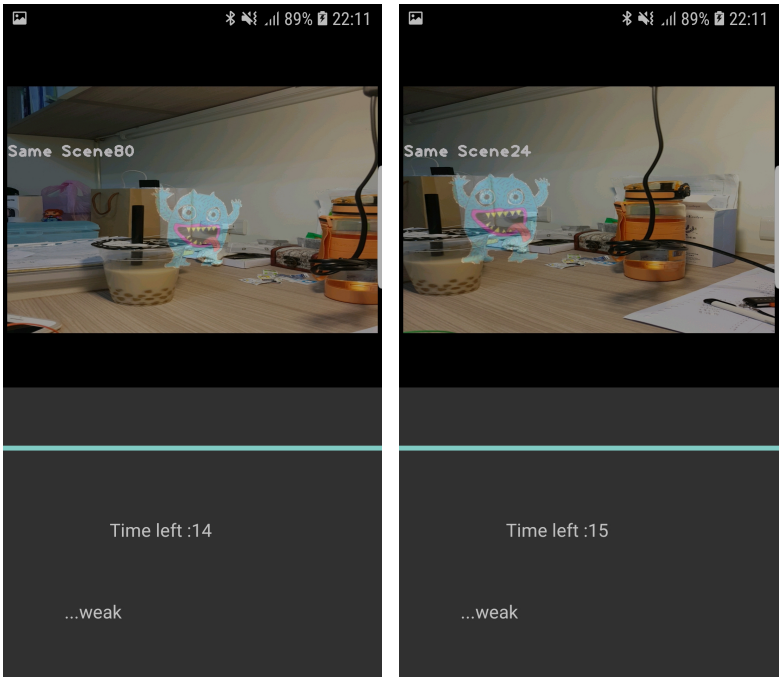
Compute the determinant of the Homography and decide whether the value is bigger than 1. When the determinant is smaller than 0, the virtual monster stitches on the scene with the mirror image, causing a distort shape; if the determinant approaches 0, it is a degenerate matrix, shown as Figure 33.9a and Figure 3.9b, each determinant is -0.06 and -0.08, so we need to avoid this situation.



(a) Homo determinant = -0.06 (b) Homo determinant = -0.08

Figure 3.9: Bad condition of homography

Figure 3.10a and Figure 3.10b show the successful appearance about the combination with augmented reality.



(a) Homo determinant = 1.4 (b) Homo determinant = 1.7

Figure 3.10: Good condition of homography

Figure 3.11 presents the coordinates of the virtual monster that stitches on the scene position by multiplying the four corners to the matrix of homography.

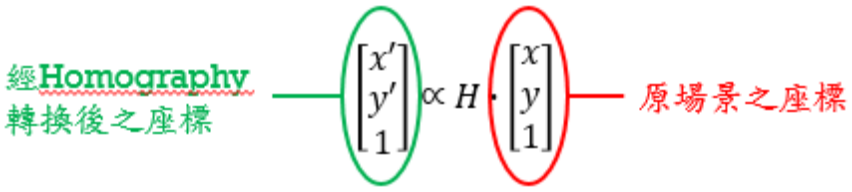


Figure 3.11: Computing fornula of homograph

As Figure 3.12, we need to multiply the coordinates of feature points to a special matrix, because we only get the middle area of the screen that we mentioned in process 1 of implementation with augmented reality. The length and the width of camera hidden in the mobile device is 704*1056, we obtain the coordinate of the small screen (0, 0) which is in the coordinate of

the original screen (176, 264) since we know that we only get the screen that reduced 1/4 of the length and the width.

Figure 3.12: Results of computation

Chapter 4

Experiment Result

In this chapter we exhibit the result of our project. We show the consequence of each functions in order under every circumstances, and explain the process of the implementation. In addition, we discuss the problems we encounter and the solution we resolve the issues through the experiment. There are parameter measurement of the sensor, efficiency of the scene analysis and user experience evaluation.

4.1 Experimental Condition

The list below shows the hardware facilities and the environment of software development in our system:
Mobile device: Android 8.0.0, RAM:4GB
Technique of image processing on camera: OpenCV 3.1.0
Environment of Arduino program development: Ardunio 1.6.12
Environment of Android program development: Android Studio 2.3

4.2 Measurement of Parameters from MyoWare Muscle Sensor

We measure and integrate statics from the muscle sensor, including recording every posture of arm muscle before we start to design the system. The following is the result of the data:

Table 4.1: Table of sensor measurement

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: Table of sensor on arms

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: Table of posture with sensor

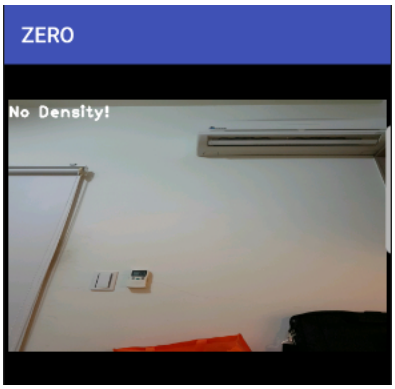
各種姿勢：

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

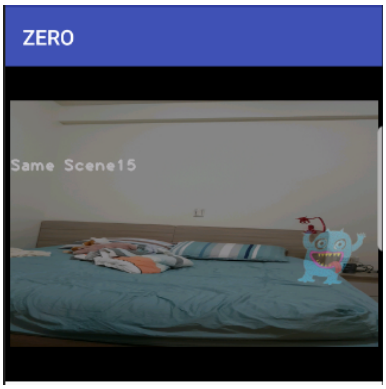
According to the result of the measurement, the numbers in the right-side column are the direct output data from the muscle sensor stitch on the user; the left-side column are the assigned movement. Based on the list above, we can observe that the average data of the force is approximately 120 when we put strength, and the data is between 20 and 50 when relaxing.

4.3 Efficiency of Scene Analysis

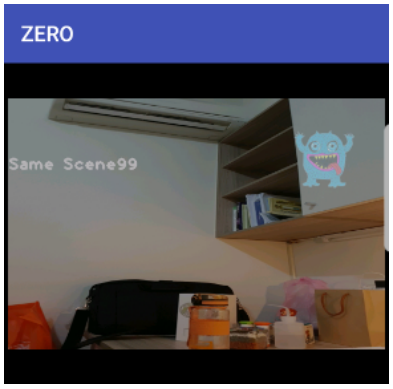
As the following:
Scene 1 (Figure 4.1a): The scene is simple. After computing the density of the scene's edge is small, the virtual monster is not able to stitch on the image.
Scene 2 (Figure 4.1b): The density of the scene edge is big enough, but the feature points are few, so the virtual monster seems distort in the image.
Scene 3 (Figure 4.1c): The density of the scene edge is big enough, and the feature points are enough, so the virtual monster stitches thoroughly and stably.
By integrating the three images mentioned above, we tend to stitch the virtual monster on the complex scene, so the result is correspond to the original setting.



(a) Scene 1



(b) Scene 2



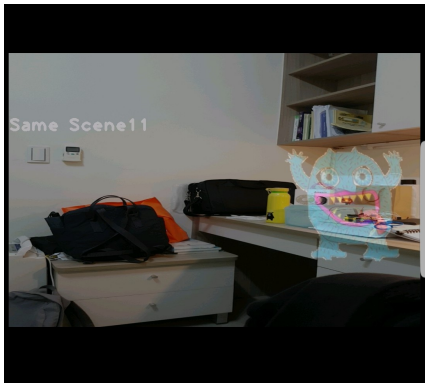
(c) Scene 3

Figure 4.1: Scene of game

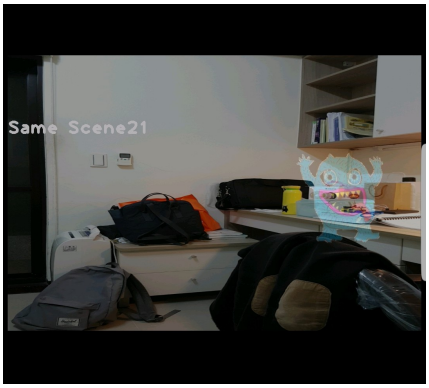
As Figure 4.2, the virtual monster displays dissimilarly through the dif-

ference of the distance. It follows the scene to enlarge, shrink and incline.

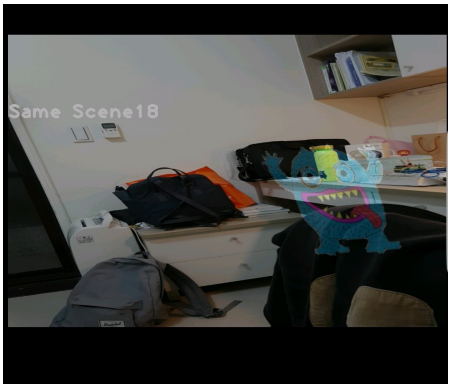
(a) Normal display of virtual monster



(b) Enlarge display of virtual monster



(c) Shrink display of virtual monster

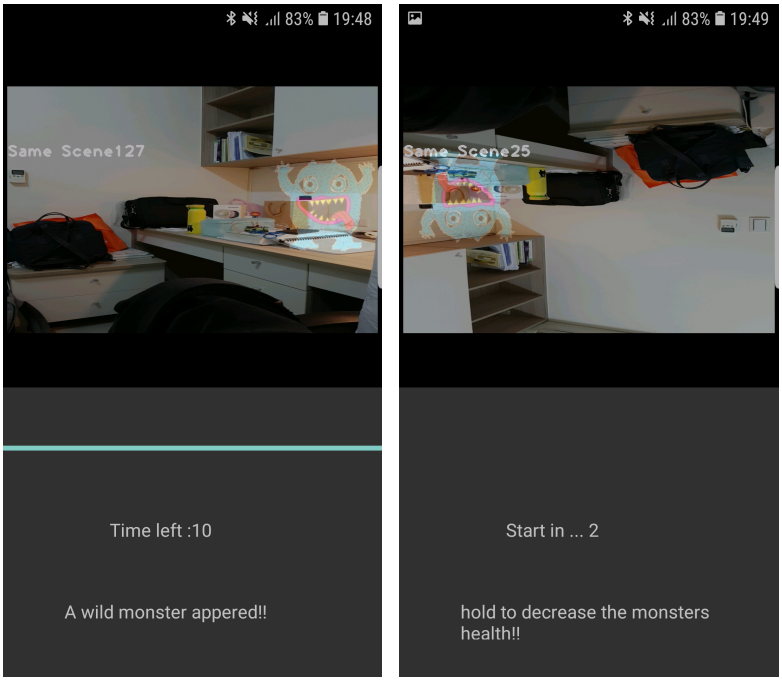


(d) Incline display of virtual monster

Figure 4.2: Display of virtual monster

Figure 4.3a shows the screen when we hold the mobile phone straight; Figure 4.3b presents the result that the screen keeps straight like the front of the camera when holding the mobile device upside down because we lock the camera hidden in the mobile device

.

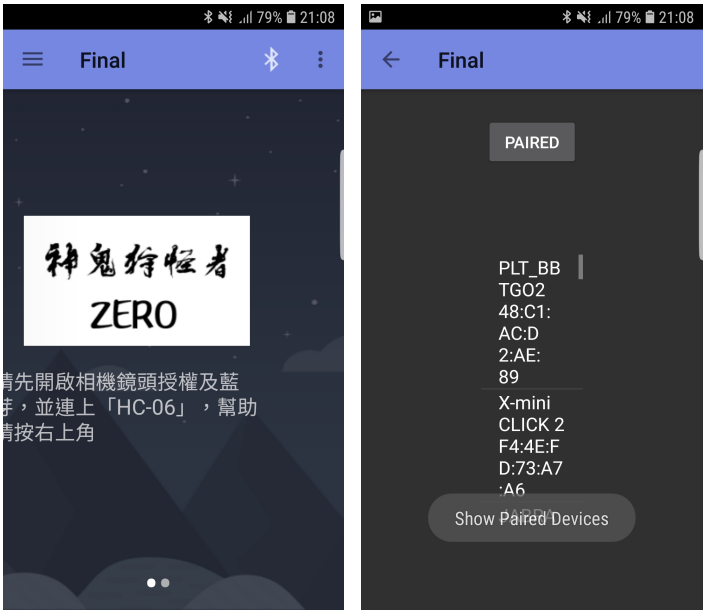


(a) Hold the phone straight (b) Hold the phone upside down

Figure 4.3: Turn the mobile device

4.4 Exhibition of the Project Resultation

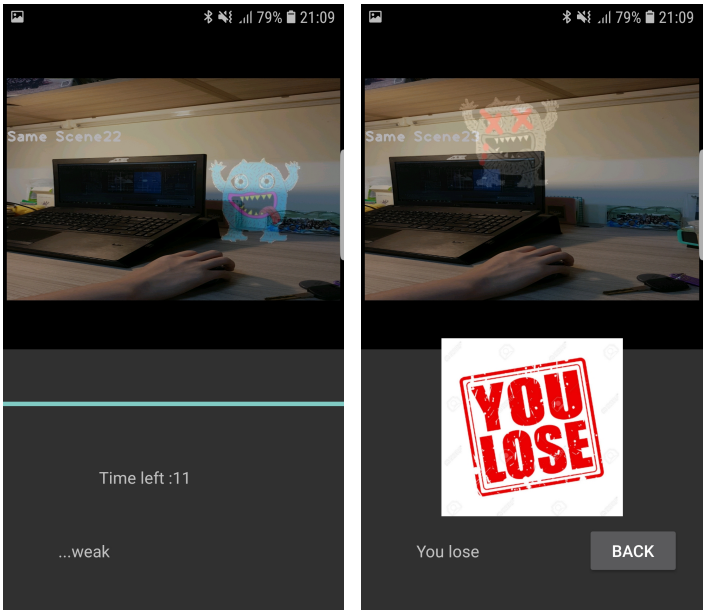
The following shows the whole process of our project:



(a) Main screen (b) Connecting bluetooth

Figure 4.4: Screen 1 of game process

Figure 4.5a and Figure 4.5b each reveals the on-going game screen and the screen when losing.



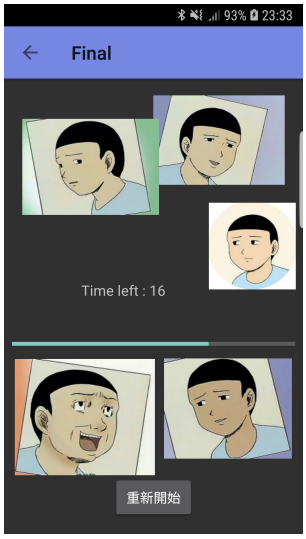
(a) Snapchat of hunting virtual monsters mode

(b) Snapchat of losing games

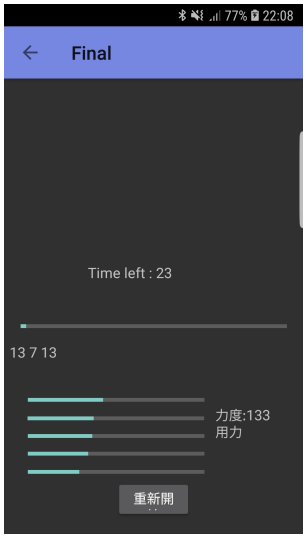
Figure 4.5: Screen 2 of game process

Figure 4.6a to Figure 4.6e are the screens of challenge mode.

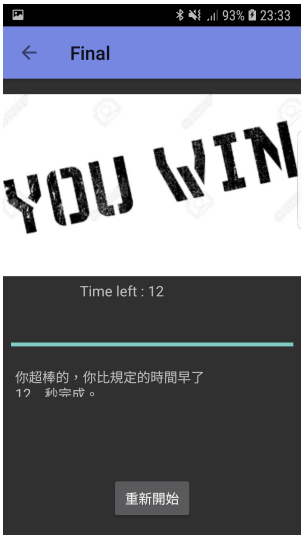
(a) Main screen of small game



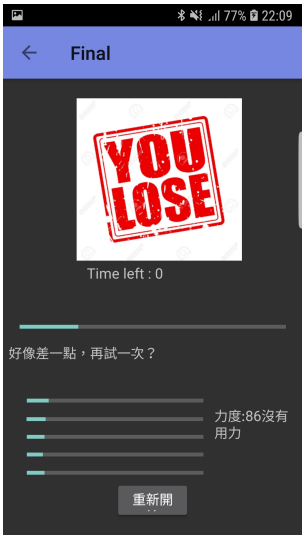
(b) Force game



(c) Rhythm game



(d) Winning game



(e) Losing game

Figure 4.6: Screen 3 of game process

4.5 User Experience Evaluation

About the table below, we find some players to experience our game. Moreover, let them rate and comment the game.

Table 4.4: Score from Players

Name	Interest(1-10)	Fluency(1-10)	Overall(1-10)	Comment
Player 1	8	6	7	我覺得小遊戲很難，手很瘦。
Player 2	9	7	8	希望能有更多不同關卡

Chapter 5

Discussion and Conclusion

In our project, the dominance of our game is its significantly innovative way of playing game. Moreover, it combines exercising. Besides, we use many techniques, including applying OpenCV module to analyze scene and analyzing the muscle sensor based on Fast Fourier Transform. In software level, we combine the virtual monster with the scene by the image processing technique of OpenCV. The problem we encounter currently is the screen shake, like hand shaking, also the bad result of stitching the virtual monster because of the lack feature points. Hence, the system must process under the enough light source environment. For the analysis of the muscle sensor, although demonstrating the virtual monster on the screen delays sometimes. Furthermore, when the frame of the time decision for the muscle sensor analysis is too long, it delays; when it is too short, it affects the accuracy, we make the smooth degree of the system to superior. In addition, we sometimes receive the noise during sensoring, it often is a huge data (exceeding the number that the human can achieve). The data keeps regular in an ideal state (for example, relax and do not put force). However, there are extremely small wave in the reality, so leading to the difference of computation and decision. Thus, we figure out some methods to resolve the issues, we filter the unusual high frequency after converting. Even though we miss the few part of frequency, the difference does not disturb the experience of the game.

MyoWare muscle sensor we use in out system can stick on every obvious muscle position, satisfying the requirement of users. For instance, obtaining different ways of playing by changing the position from arm to leg. Our future goals are raising the levels of the game's difficulty, and enhancing the function by connecting two mobile devices and stand off between two opponents. Futhermore, providing the function players can draw the virtual

monsters by themselves, and making the virtual monsters become the target of attacking is also an interesting interactive mode.

Appendix

Appendix 1.1 Source code of filter noise from sensor:

```
/*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*/
```

Appendix 1.2 Source code of Fast Fourier Transform:

```
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);
            }
        }
    }
}
```

Appendix 1.3 Source code of analyzing scene's density:

```
/*----計算密度----*/
//去除雜訊
Imgproc.GaussianBlur(templ, 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----*/
```

Appendix 1.4 Source code of analyzing scene's feature points:

```
//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 dm1 = dmatches[0];
    DMatch dm2 = dmatches[1];
    if (dm1.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));
}
```

Appendix 1.5 Source code of converting scene's coordinates:

```
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> sceneTempl = new LinkedList<Point>();
    MatOfPoint2f scenel = new MatOfPoint2f();
    for (KeyPoint k : gdKeyPoint) {
        sceneTempl.add(k.pt);
    }
    scenel.fromList(sceneTempl);
    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.spKeyPointl.getFirst();
    Point pointl = 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[]{pointl.x, pointl.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);
```

Appendix 1.6 Source code of stitching virtual monsters:

```
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());
```

Appendix 1.7 Source code of reading sensor's data:

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

Bibliography

[1] [Resource of reading system's images]
http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2012/1201/655.html

[2] [Button interface]
http://hukai.me/android-training-course-in-chinese/basics/actionbar/adding-buttons.html

[3] [Effect of homepage interface]
https://dotblogs.com.tw/starhao/2016/10/08/012050

[4] [Detection of feature points in scene analysis]
https://opencv-python-tutroals.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] [Images combination]
http://peaceandhilightandpython.hatenablog.com/entry/2016/01/16/002028

[6] [Fast Fourier Transform]
https://rosettacode.org/wiki/Fast_Fourier_transform#C.2B.2B

[7] [Bluetooth connection]
https://home.gamer.com.tw/creationDetail.php?sn=3671289

[8] [Session]
https://blog.csdn.net/ZDW86/article/details/35795125