SENIOR PROJECT CN-2-2016

Emotivity
An EEG Application Using Emotiv

Submitted to

School of Information, Computer and Communication Technology
Sirindhorn International Institute of Technology
Thammasat University

May 2017

by

Poonnawich Siriwongse Na Ayudhaya 5622770329

Advisor: Assoc. Prof. Dr. Cholwich Natte

Abstract

Emotivity is an electroencephalogram application using electroencephalographic headset known as 'Emotiv EPOC' in which facial expression could be measured, allowing one's emotion to be determined using those expressions in real-time.

Acknowledgements

I would like to give my acknowledge to the Emotiv team for creating the headset , and to the community surrounding the said creation for providing great examples on not only how to properly use complicating commands from the library, but also on how to properly take care of the device to extend its longevity.
I would also like to thank Assoc. Prof. Dr. Cholwich Natte for providing me with the headset, allowing me to complete the project.

Contents

Abbreviations

The technical terms listed below will be expressed by their abbreviations in this report. Other abbreviations will be defined at their first occurrence.

EEG Electroencephalogram

List of Figures

Chapter 1

Project Concept

1.1 Summary

The project will simply use Emotiv EPOC, a neuroheadset capable of recording electromagnetic waves surrounding the user's head, to measure those waves to be used for an application development in which the user's emotion could be determined and outputted, leading to many possibilities in programming with the most prominent one being the automation of emoticon selection in social network applications.

1.2 Motivation

Having social network as a daily mean of communication fades out the raw expression that is in face-to-face conversation. Real-time facial conversation streaming and the usage of emoticon already have established pillars that are to mimic the real life proximity of socialising. Emotivity aims to be the central pillar that combines the flow of real-time facial conversation and the compacted overhead of emoticon.

1.3 Users and Benefits

This program is aimed towards people who are interested in social network and its application development. They can be divided into two groups:

- Programmers that are interested in Emotiv and social network application could potentially integrate an automated emoticon selection feature as either a plugin or a built-in as well giving themselves insight on their current emotion.
- Non-programmers may give themselves insight on their current emotion without having prior knowledge of EEG or programming.

1.4 Typical Usage

Programmers will be able utilise the output from Emotivity to develop their own application or plugin surrounding social network applications.

Both programmers and non-programmers will be able to give themselves basic insight on their current emotion.

1.5 Main Challenges

One of the potentially hardest challenges would be how to accurately capture the emotion on multitude of people. Some of us may have a slightly different way of expressing ourselves.

Another challenge would be the speed in which the program can compute the output. To be able to be effective on social network applications, it would have to be stable and reliable.

Chapter 2

Requirements Specification

2.1 System Description

Emotivity is an application that classifies your emotion and outputs that emotion. It allows programmers to utilise the output to further their own applications. It also allows non-programmers or programmers alike to be able to examine that emotion using the output display and other types of visualisation.

2.1.1 Perspective

The relationships between Emotivity, Emotiv EPOC, EmoEngine and users are shown below in Figure 2.1

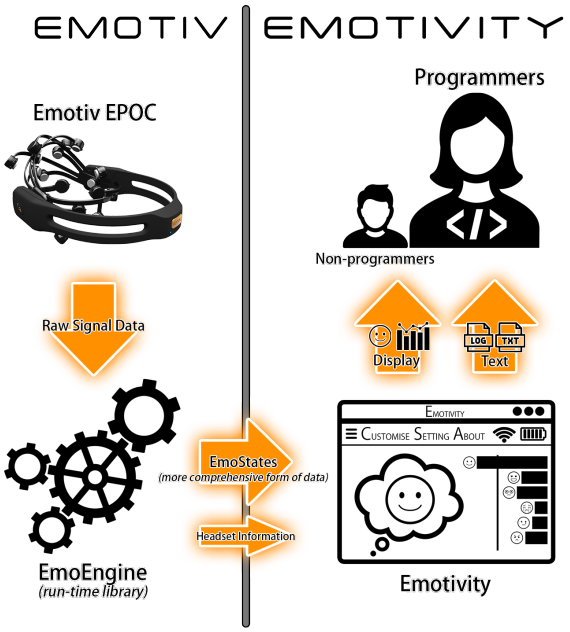


Figure 2.1: System Perspective

Since Emotiv EPOC outputs only as a raw signal data, we need the EmoEngine to process that data in order to make the data comprehensive to us.

Emotivity, then, uses the processed data called 'EmoStates' as input. After being processed by Emotivity, it is then output to the user. The output from Emotivity can be categorised into two major entities; display and text.

Programmers may utilise both types of the output, while non-programmers may only be able to use the display output. This is due to the fact that non-programmers may not have the necessary knowledge or experience to utilise the latter.

2.1.2 Functions

The interactions between the functions and the classification engine are show below in Figure 2.2.

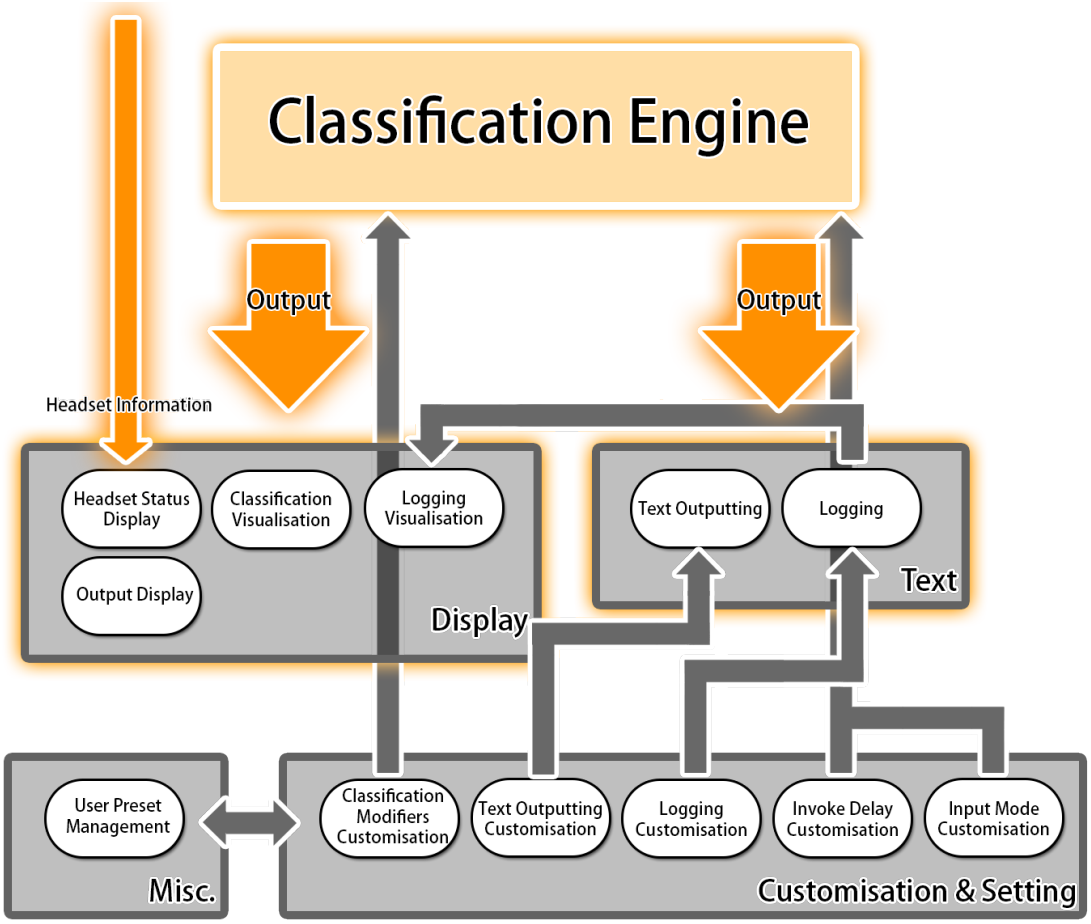


Figure 2.2: System Functions

The functions are summarised as:

- Output Display: allows the users to see the visualised form of the output.
- Classification Visualisation: allows the users to see the visualised form of classification as bar charts.
- Text Outputting: allows the programmers to obtain the output in form of text.
- Logging: allows the programmers to record instants of values and output over a period time.
- Logging Visualisation: allows the programmers to see the logs from within the program.
- Headset Status Display: allows the user to check the status of the headset.
- Classification Modifiers Customisation: allows the users to customise the value of the modifiers for each and every emotion profile.
- Text Outputting Customisation: allows the programmers to toggle the text outputting function on or off and customise the location of the output text.
- Logging Customisation: allows the programmers to toggle the logging function on or off and customise the location of the logs.
- Invoke Delay Customisation: allows the programmers to customise the computation delay.
- Input Mode Customisation - allows the programmers to select between device input or simulator input.
- User Preset Management - allows the users to save states, separate them as desired and delete it when it is needed to be done.

2.2 Requirements

The requirements of the system are listed in the following subsections.

2.2.1 General

G1. It should be very responsive; that is very little delay when interacted with.

G2. It should be able to handle the case where the headset is not plugged in, and efficiently notify the user.

2.2.2 Output Display

OD1. It should be universal to all users regardless of their language.

OD2. There should be very little latency in the updating of the current state.

OD3. It must output the correct emotion state according to the classification.

OD4. It should be able to handle the case where the headset is not plugged in, and efficiently notify the user. /item[OD5.] It should be able to display the output even when the program is trayed.

2.2.3 Classification Visualisation

CV1. It must correctly display the status of each and every emotion in the classification process.

CV2. There should be very little latency in the updating of the current state.

CV3. It should be universal to all users regardless of their language.

CV4. It should be able to handle the case where the headset is not plugged in, and efficiently notify the user.

2.2.4 Text Outputting

TO1. It must output data in a format or manner that can be used by a multitude of different programming languages.

TO2. There should be very little latency in the writing of the output.

TO3. It must output the correct emotion according to the classification.

TO4. It must allocate the output in the location that is set by the customisation.

2.2.5 Logging

L1. There should be very little latency in the logging of the output.

L2. It must correctly log the output in correspondence with time.

L3. It must be able to function as set by the customisation.

2.2.6 Logging Visualisation

LV1. There should be very little latency in the visualisation of the logging.

LV2. It must correctly display the logging the output in correspondence with time.

LV3. It must be synchronised with the text form of output logging.

LV4. It must be able to be toggled on or off by the user.

2.2.7 Headset Status Display

HSD1. It must correctly represent the status of the headset based on the headset information data.

HSD2. It should be able to handle the case where the headset is not plugged in, and efficiently notify the user.

2.2.8 Classification Modifiers Customisation

CMC1. It should be universal to all users regardless of their language.

CMC2. The values that are changed should be correctly and immediately passed to the classification engine when they are saved.

CMC3. The state of every value should be able to be set to the default.

CMC4. It should be able to restrict invalid input format.

CMC5. It must be able to correctly show loaded values in respect to the current profile.

2.2.9 Text Outputting Customisation

TOLC1. The state of every value should be able to be set to the default..

TOLC2. It should be able to restrict invalid input format.

TOLC3. It must be able to toggle on or off the text outputting function.

2.2.10 Logging Customisation

LC1. The state of each and every user input must be saved correctly in all profiles.

LC2. The state of every value should be able to be set to the default.

LC3. It should be able to restrict invalid input format.

LC4. It must be able to toggle on or off the logging function.

2.2.11 Invoke Delay Customisation

IDC1. The state of every value should be able to be set to the default.

IDC2. It should be able to restrict invalid input format.

IDC3. It must be able to correspond the input value to the loop delay time in the classification engine correctly.

2.2.12 Input Mode Customisation

IMC1. The state of every value should be able to be set to the default.

IMC2. It must correctly correspond the set mode to the input method used by the classifcation engine.

2.2.13 User Preset Management

UPM1. It must be able to let the user create a new preset of any name that is not too long.

UPM2. It must be able to save and load the preset that is created along with the values respective to it.

UPM3. It must be able to delete any non-default preset.

Chapter 3

Design Specification

3.1 System Architecture

High-level interactions between each system and module are show below in Figure 3.1.

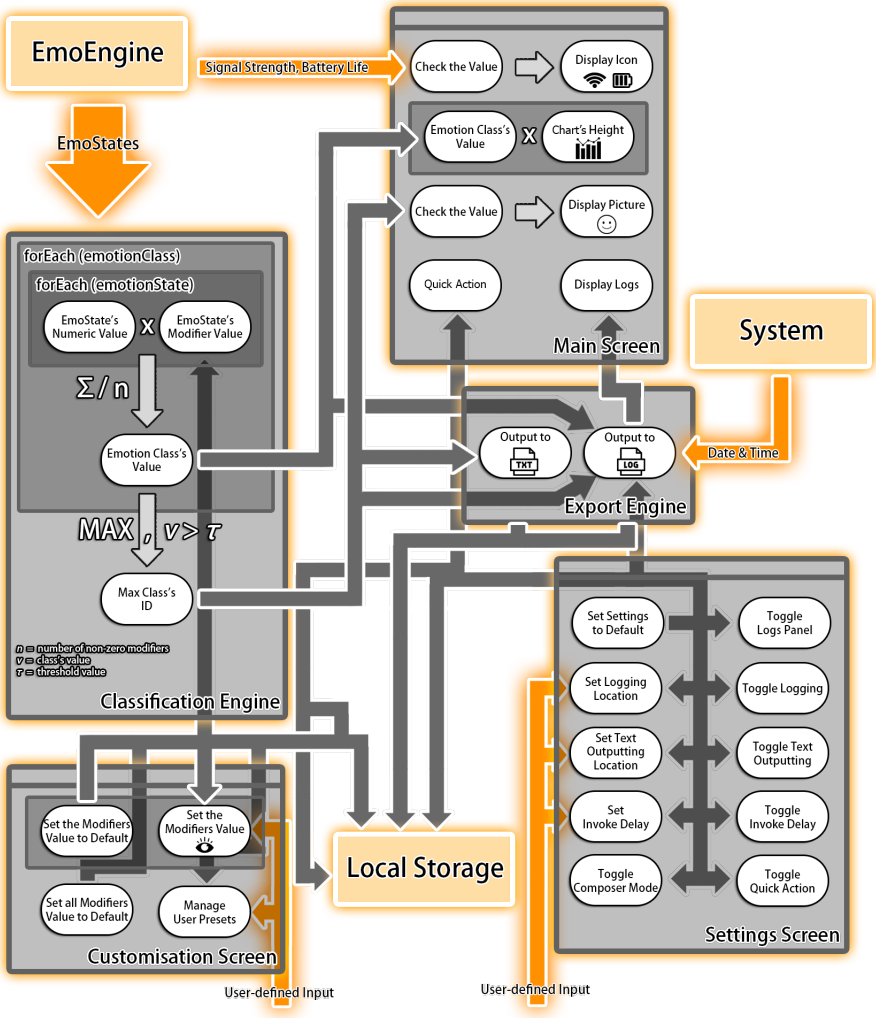


Figure 3.1: System Architecture

3.1.1 EmoEngine

'EmoEngine' is a fancy name that is used to describe the convenience that is the run-time library that could be easily invoked in ranges of different programming languages.

EmoEngine process the raw signal data sent from Emotiv headset into a more comprehensive form, which is an integer running from 0 to 1. They can be obtained by invoking the commands provided by the engine.

These more comprehensive form of data are called EmoStates.

3.1.2 Classification Engine

'Classification Engine' is a name given to a bundle of modules which work together in order to calculate the value of each emotion class and the maximum valued emotion class. These values are passed to another system to be used for processing and outputting.

The value of each emotion class is calculated by multiplying each of the modifier value of an EmoState with the EmoState value that is invoked from the EmoEngine. All of these values are summed up and divided by the number of non-zero modifiers defined by the user. This is done purely to weight the attributes in different classes, so that each class may become unique.

The emotion class with the maximum value is chosen based on basic Mathematical comparison.

3.1.3 Export Engine

'Export Engine' handles the outputting of data in textual form that consists of text file and log file. It takes the values output from the Classification Engine as inputs.

Text file writing is done simply by replacing the content with the maximum emotion class ID over and over again. With this, there is only one file to be looked for (for the output).

Log file writing, on the other hand, is done by appending new lines to the existing content. It calls on local date and time from the local operating system to be used as timestamps. Unlike text file writing, it logs both the maximum emotion class and the value of each emotion class since it may be useful in debugging. The behaviour of the writing, however, may be changed depending on the settings from the user.

3.1.4 Main Screen

The main screen handles the outputting of data in form of display, the display of headset status, the logging visualisation and user profile management. Most of the functions in this system are very straightforward in term of their duties.

3.1.5 Customisation Screen

The customisation screen oversees the modifiers' value of each EmoState in each emotion class. It exists because different users may have different readings from the headset.

When a new value is assigned, it is passed to the Classification Engine and saved in the local storage as a file based on the current preset.

When it is set to the default, the pre-defined value in the program is reassigned back both to the Classification Engine and the local storage.

User presets management includes the creation, deletion and saving of a preset. When a preset is created or deleted, a corresponded action is invoked to the local storage. When a preset is updated, the new values are rewritten over the old ones

3.1.6 Setting Screen

The setting screen is mainly about the ability to modify the behaviour of varieties of modules from basic things such as toggling the text outputting and logging to slightly advanced stuff as given below.

'Quick Action' allows the user to quickly toggle some functions in the main screen that are normally toggle-able only in this screen such as text outputting or logging.

'Invoke Delay' allows the user to specific the time period in which the program loops or invokes its computation methods.

'Logs Panel' simply enables the user to view the logs from within the program itself.

'Composer Mode' offers the ability to switch between 'device' or 'simulator' input mode.

3.1.7 Local Storage

Local storage represents a space or folder that is used to store variety of files.

3.1.8 System

System represents the operating system on the local machine.

3.2 Detailed Design

Since the program that is used to build Emotivity is Visual Studio, most of the interface parts are done purely drag and drop with little to manual coding.

These pseudo codes will provide you with the general low-level ideas on how the functions are implemented. The conversion from pseudo code to practical language would be an easy task compared to other design techniques.

3.2.1 Classification Engine

input : An array of EmoState values $emotionStateArray$

output: The maximum emotion class value $emotionClassMAX$

foreach $emotionClass$ ***in*** $emotionClassArray$ **do**
    foreach $emotionState$ ***in*** $emotionStateArray$ **do**
        $emotionValue$ =+ $emotionState \times emotionStateModifier$;
    end
    $averageValue$ = $emotionValue \div numberOfNoneZeroModifiers$;
    $emotionClass$.setValue($averageValue$);
end
$emotionClassMAX$ = indexOf(max($emotionClassArray$));

3.2.2 Export Engine

Text File Output

input : The maximum emotion class value $emotionClassMAX$

output: Updated output text file $outputFile$

while *EmoEngine is connected* **do**
    $outputWriter$ = openFile($outputFile$);
    $outputWriter$.write($emotionClassMAX$);
    closeFile($outputFile$);
end

Log File Output

input : An array of emotion class value $emotionClassValueArray$, the maximum emotion class value $emotionClassMAX$

output: Updated output log file $outputFile$, updated log panel $logPanel$

while *EmoEngine is connected* **do**
    $stringToWrite$ = '['+$system.date()$+$system.time()$+']'+'ARRAY:
     '+emotionClassValueArray+', MAX: '+emotionClassMAX;
    $outputWriter$ = openFile($outputFile$);
    $outputWriter$.writeLine($stringToWrite$);
    closeFile($outputFile$);
end

3.2.3 Main Screen

Output Display

input : The maximum emotion class name $emotionClassMaxName$

output: Updated output display $cloudContentImage$

while *EmoEngine is connected* **do**
  $mainScreen.cloudContentImage$.image =
  Image.fromFile($emotionClassMaxName$.png);
end

Classification Visualisation

input : An array of emotion class $emotionClass$

output: Updated emotion bar charts $barChart$

foreach $emotionClass$ ***in*** $emotionClassArray$ **do**
  barChanged =
  $emotionClassArray[emotionClass] \times mainScreen.barChart$.height;
  $mainScreen.barChart$.height = $barChanged$;
  iconBarChanged = $iconBar_BarGapWidth$ + $mainScreen.barChart$.height;
  $mainScreen.barIcon$.position.x = $iconBarChanged$;
end

Logging Display

input : An array of emotion class value $emotionClassValueArray$, the maximum emotion class value $emotionClassMAX$

output: Updated log panel $logPanel$

while *EmoEngine is connected* **do**
  $stringToWrite$ = '['+$system.date()$+$system.time()$+']'+'ARRAY:
  '+emotionClassValueArray+', MAX: '+emotionClassMAX+']';
  $logPanel$.append($stringToWrite$);
end

3.2.4 Customisation Screen

Loading States

input : Customisation file $customFile$

output: An array of emotion class $emotionClassArray$

$outputWriter$ = openFile($programDirectory$+$userDirectory$+$customFile$);
foreach $sectionHeader$ ***in*** $customFile$ **do**
    foreach $value$ ***in*** $sectionHeader$ **do**
        $emotionClassArray[sectionHeader].emotionStateModifierArray[value]$
        = $outputWriter$.readLine();
    end
end
$outputWriter$.closeFile();

Saving States

input : An array of emotion class $emotionClassArray$

output: Updated customisation file $customFile$

$outputWriter$ = openFile($programDirectory$+$userDirectory$+$customFile$);
foreach $emotionClass$ ***in*** $emotionClassArray$ **do**
    $outputWriter$.writeLine('['+$emotionClass$+']');
    foreach $emotionStateModifier$ ***in***
    $emotionClass.emotionStateModifierArray$ **do**
        $outputWriter$.writeLine($emotionStateModifierName$ =
        $emotionStateModifier$);
    end
end
$outputWriter$.closeFile();

3.2.5 Settings Screen

Invoke Delay

input : An integer number measured in milliseconds $invokeDelay$

while *EmoEngine is connected* **do**
    ...

    $Thread$.Sleep($invokeDelay$);
end

Loading States

input : Settings file $settingFile$

output: An array of settings $settingArray$

$outputWriter$ = openFile($programDirectory$+$settingFile$);
foreach $value$ ***in*** $sectionHeader$ **do**
    $settingArray[value]$ = $outputWriter$.readLine();
end
$outputWriter$.closeFile();

Saving States

input : An array of settings $settingArray$

output: Updated settings file $settingFile$

$outputWriter$ = openFile($programDirectory$+$customFile$);
foreach $emotionClass$ ***in*** $emotionClassArray$ **do**
    $outputWriter$.writeLine('['+$emotionClass$+']');
    foreach $emotionStateModifier$ ***in***
     $emotionClass.emotionStateModifierArray$ **do**
        $outputWriter$.writeLine($emotionStateModifierName$ =
         $emotionStateModifier$);
    end
end
$outputWriter$.closeFile();

Chapter 4

System Manual

4.1 Installation and Usage

Installation:

1. Extract the zipped file using any ZIP utility program of your choice.

Usage:

1. Prepare your Emotiv EPOC by following its standard procedures and put it on.
2. Plug in the wireless USB drive bundled with the device, while making sure it is properly paired.
3. Start the program.

4.2 Implementation Overview

A brief description of each file is explained down below as followed:

'emotivityBase.cs' is a file which contains base functions that are shared among the screens.

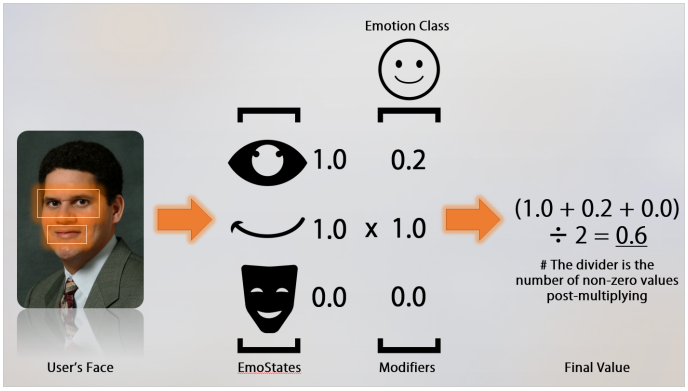
'emotivityAboutScreen.cs' is a file which maps functions to interface in the about screen.

'emotivityCustomiseScreen.cs' is a file which maps functions to interface in the customisation screen. It reads and writes preset data.

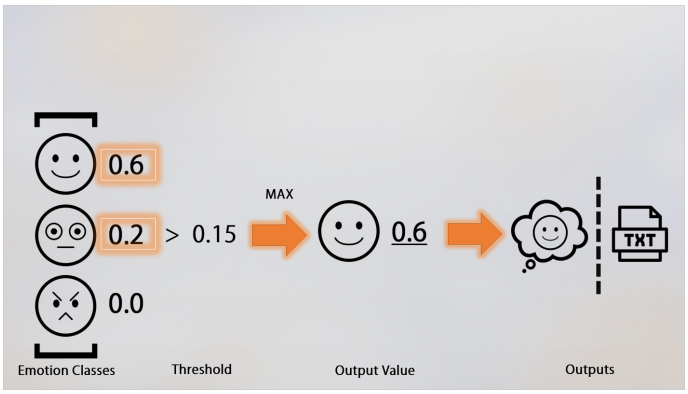
'emotivitySettingScreen.cs' is a file which maps functions to interface in the settings screen. It reads and write user settings data.

'emotivityMainScreen.cs' is a file which maps functions to interface in the main screen. It executes Emotivity main functions. It reads user data and perform according. It also oversees many other operations such as text outputing and logging when enabled.

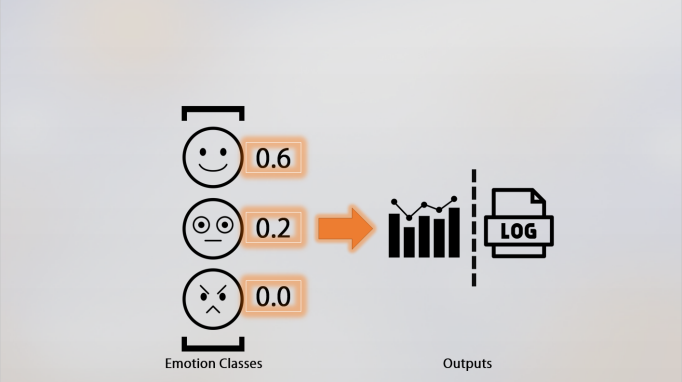
The diagrams below show how facial expressions are ultilised:



EmoStates are put into an array in order to be multipled by a respective array of modifiers for each emotion class. After performing multiplication, all values in the array are summed up and divided by the number of non-zero vaules post-multiplication. This vaule represents the emotion class's value.



After calculating the vaule of each emotion class, they are put into an array once again, to be compared to the threshold. The value that is the greatest in numeric value and surpasses the threshold value is chosen as the output value. This value is used to determined which emotion class gets to printed out as a text and displayed as a picture.



The array of emotion class values are further used to be displayed as bar charts and printed out as logs.

Software Versions:

1. Microsoft Visual Studio Express 2015 (C#), Version 14.0.251123.00 Update 2
2. .Net Emotiv SDK, Version 0.8.4.0
3. EDK API, Version 1.0.0.6

4.3 Feature List

1. Output Visualisation: Users will be able to see the output value visualised as a picture.

   Feature status: [✓]Complete [ ]Partial [ ]Initial

2. Classification Visualisation: Users will be able to see the statistical emotion class values visualised as bar charts.

   Feature status: [✓]Complete [ ]Partial [ ]Initial

3. Modifiers Customisations: Users will be able to customise the value of modifiers for each emotion class along with the overall threshold value. They will be able to save and load those settings under their desired name as well.

   Feature status: [✓]Complete [ ]Partial [ ]Initial

4. Program Settings: Users will be able to customise certain behaviours of the program such as the directory of the output text file is generated. They will be able to save and load those settings. This feature is primarily aimed to be beneficial to programmers.

   Feature status: [✓]Complete [ ]Partial [ ]Initial

5. Headset Status Display: Users will be able to see the signal strength and battery life of Emotiv EPOC.

   Feature status: [✓]Complete [ ]Partial [ ]Initial

6. Output and Logs: Users will be able to retrieve the output value in a text file and check the logs in a log file or in the program panel. This feature is primarily aimed to be beneficial to programmers.

   Feature status: [✓]Complete [ ]Partial [ ]Initial

7. Graphical User Interface: Users will be able to navigate through various features and functions offered by the program via comprehensive and smooth interfaces.

   Feature status: [✓]Complete [ ]Partial [ ]Initial

8. User Data Management: User defined data will be managed by directorial management and XML structured files. They will be able to relocate or manually edit those data as needed.

   Feature status: [✓]Complete [ ]Partial [ ]Initial

4.4 Test Results

There are two ways that tests are carried out: by using Xavier Composer, an input simulator tool, or by using the actual headset to get the inputs.

With the Composer, an undistrubed stream of inputs can be simulated to verify whether the algorithm of Emotivity yields the desired results that is the user's facial expression matches their desired emotion output. However, individual's expression may differ from other by some amount of factor, hence customisation of each modifier is implemented. This makes the set of tests dependent on the user themselves, rather than myself, the developer, to verify whether their expression matches their desired output. The only way that a program error could occur is that it is poorly written (please see source code for reference to see whether it is poorly written.)

With the headset, the integrity of the inputs is highly dependent on many factors such as the user's hair density or the condition of the headset's sensors. If the inputs are disorted or lost too much, the computation of emotion class cannot be carried out effectively, rendering the output mismatched with the user's desire.

Appendix A

Chapter 4

Emotion class computation in 'emotivityBase.cs':

```
// Computing the class value
public Single computeValue(Single[] emotionArray)
{
    Single computingValue = 0;
    Single modifiersSumValue = 0;

    for (int i = 0; i < emotionArray.Length; i++)
    {
        computingValue += emotionModifiers[i] * emotionArray[i];
        if ((emotionModifiers[i] * emotionArray[i]) > 0)
        {
            modifiersSumValue += emotionModifiers[i];
        }
    }
    if (modifiersSumValue == 0)
        modifiersSumValue = 1;
    // Dividing gives us values in static range across all classes despite of their
        modifiers
    // MIN: 0 MAX: 1
    if (computingValue >= 0) { computingValue = computingValue / modifiersSumValue; }
    else { computingValue = 0; }
    computingValue = Convert.ToSingle(Math.Round(computingValue, 2));

    emotionValue = computingValue;
    return computingValue;
}
```

Bibliography

[1] Emotion classification, Shaver et al. (2001): http://changingminds.org/explanations/emotions/basic%2

[2] C# tutorials and useful refernces:
https://www.tutorialspoint.com/csharp/
https://msdn.microsoft.com/en-us/library/aa288436(v=vs.71).aspx

[3] Emotiv community SDK with very helpful examples:
https://github.com/Emotiv/community-sdk

[4] Emotiv API references: http://emotiv.github.io/community-sdk/_iedk_8h.html