

Automatic emotion capture when viewing web-based media on a smartphone

Matthew Jones

School of Computer Science & Informatics
Cardiff University

Supervisor: Professor David Marshall

Moderator: Dr Yukun Lai

May 2016

Abstract

This Final Year Report documents the work that has been completed to develop a proof-of-concept system capable of recognising and measuring human emotion whilst viewing web-based media. This involves using a facial capture system, to obtain a set of key facial features, and to then apply a machine learning approach to these features to recognise particular emotions. Emotion recognition has received a lot of attention with many potential applications in areas such as healthcare, marketing, and social computing. The main focus of the project was to apply this technology to a mobile device, and to develop general-purpose tools to conduct emotion experiments. These tools were then validated through conducting an experiment to further understand how humans respond to different types of web-based media such as YouTube videos, Tweets, and web pages.

The developed system involves an iOS application capable of displaying different types of web-based media, performing facial capture using the mobile device's camera, and applying an emotion classifier to the obtained data. This application is supported by 2 web applications and APIs developed to set up emotion experiments and to analyse the results. A significant amount of time has also been spent researching and developing Support Vector Machine classifiers capable of accurate and robust classification of the emotions Anger, Contempt, Disgust, Fear, Happiness, Natural, Sadness, and Surprise.

The report details the purpose, approach, and implementation details for each of the system's components, with relevant theory explained where necessary. Evaluations of the system have been made against a set of user requirements, as well as through analysing the results from the validation experiment carried out with 16 participants.

Finally, the report concludes with analysis of the current state of the solution, with suggestions for potential future improvements and functionality, along with personal reflections and learning points from the project.

Acknowledgements

The success of this project has been underpinned by the work completed for the CUROP project "Facial Expression Analysis on a Smartphone" during Summer 2015. As such, I would like to thank Hristo Georgiev and Abhijat Biswas for their contributions to this project, and for all those at Cardiff University who provided me with the opportunity of undertaking the project.

I am very grateful to all those who gave up their time and participated in my experiment. The data and qualitative feedback gained during this process was invaluable.

Throughout the project, I have been lucky enough to receive advice and guidance on my project from PhD students within the School of Computer Science & Informatics, Cardiff University. Specifically, I would like to thank Beryl Noë, Nyala Noe, and Liam Turner for their support. I would also like to thank Professor Roger Whitaker for providing knowledge and ideas that helped to shape the project.

Finally, I would like to express my gratitude to my supervisor, Professor David Marshall, for his encouragement and support throughout this project, and throughout my time at University.

Contents

List of Figures	7
List of Tables	9
1 Introduction	10
1.1 Aims of project	10
1.2 Intended Audience	11
1.3 Approach	11
1.4 Assumptions	12
2 Background	14
2.1 Face tracking	14
2.1.1 Active Shape Models	15
2.1.2 FaceTracker Library	16
2.2 Emotion Classification	17
2.2.1 Support Vector Machines	19
2.2.2 Emotion databases	20
2.3 Existing solutions	22
2.3.1 IntraFace	22
2.3.2 EmoVu	23
2.3.3 Other work	24
3 Developed System	25
3.1 An API-centric web application capable of creating experiments and viewing experiment results	25
3.2 An API for accessing/modifying experiment related data	26
3.3 An iOS application capable of performing these emotion classification experiments	27
3.4 Emotion classifiers to measure different emotions	28

4	Specification & Design	29
4.1	System design	29
4.2	User requirements	30
4.2.1	Emotion experiment creation tool design	31
4.2.2	Emotion experiment capture tool design	32
4.2.3	Emotion experiment results tool design	34
4.3	API design	35
4.3.1	Experiment API	35
4.3.2	Result API	36
4.4	User interface design	37
5	Implementation	39
5.1	Tools used	39
5.1.1	Emotion experiment creation tool - Tools	39
5.1.2	Emotion experiment capture tool - Tools	40
5.1.3	Emotion experiment results tool - Tools	41
5.1.4	Emotion classifier - Tools	41
5.2	Emotion experiment creation tool	41
5.2.1	MEAN stack setup	41
5.2.2	Experiment API	42
5.2.3	Experiment creation web application	44
5.3	Emotion experiment capture tool	46
5.3.1	Application interface	46
5.3.2	Obtaining experiment information	47
5.3.3	Loading experiment items	48
5.3.4	Face tracking	51
5.3.5	Emotion classification	52
5.3.6	Recording experiment participants	53
5.4	Emotion experiment results tool	55
5.4.1	Result API	55
5.4.2	Experiment results web application	56
5.5	Emotion classifier	58
5.5.1	Obtaining features	58
5.5.2	Principal Component Analysis	60
5.5.3	SVM training	61
6	Results and Evaluation	63
6.1	Experimentation	63
6.1.1	Experiment design	63
6.1.2	Experiment execution	64

6.1.3	Experiment analysis	65
6.2	Further emotion classifier work	70
6.2.1	Experiment simulation	71
6.2.2	Multi-class classifier with distributed training data	72
6.2.3	Database specific multi-class classifiers	73
6.2.4	Multi-class classifier with a subset of emotion databases	74
6.3	Requirements evaluation	76
7	Conclusions	80
8	Future Work	82
8.1	Emotion classifier	82
8.2	Results API	83
8.3	Experiment design	84
8.4	Analysing/visualising experiment results	84
8.5	Emotion experiment capture tool	85
9	Reflection and Learning	87
9.1	Reflection	87
9.2	Learning	88
	Bibliography	91

List of Figures

2.1	A set of labelled facial landmarks that represent a human face	14
2.2	Root-mean-squared (RMS) errors between ground truth landmarks and result- ing fit from various face tracking methods over 5000 frames [9]	17
2.3	Comparison of the IntraFace [28] and CUROP iOS apps [7] when expressing Surprise	22
2.4	Screenshot of the web-based EmoVu demo [27]	23
3.1	Screenshots of the developed experiment creation web application	25
3.2	Screenshots of the developed experiment results web application	26
3.3	Screenshot of an Experiment API GET request	27
3.4	Screenshot of a Result API GET request	27
3.5	Screenshots from the iOS application used to conduct emotion classification experiments	28
4.1	Abstract flow chart for the experiment process	29
4.2	Sitemap design for the experiment creation web application	32
4.3	Abstract flowchart of the main tasks involved in an experiment on the iOS application	33
4.4	Sitemap design for the experiment results web application	34
4.5	Abstract data model for an Experiment	36
4.6	Abstract data model for a Result	36
4.7	UI mockup for the emotion capture iOS application	37
5.1	AngularJS form developed to facilitate experiment creation	45
5.2	XCode Storyboard for the experiment iOS application	47
5.3	Experiment interval screen displayed before each experiment item	51
5.4	Memory allocations instrument recordings from iOS Instruments development tool whilst viewing experiment items	55
5.5	Cross-validation plot for the emotion classifier used in experiments	62

6.1	Average emotion measurements for BBC News experiment item	66
6.2	Average emotion measurements for 'DadJokes' Twitter feed experiment item .	67
6.3	Average emotion measurements for Guinness World Record YouTube video experiment item	67
6.4	Average emotion measurements for Tweet experiment item	68
6.5	Comparison of average emotion measurement results for participants viewing a cute animals Twitter item. Results are from the original classifier used in the validation experiment (left) and a classifier with distributed training data (right)	73
6.6	Comparison of average emotion measurement results for participants viewing a news blooper item. Results are from the original classifier used in the validation experiment (left) and a classifier with no GEMEP-FERA [25] training data (right)	75
6.7	Experiment participant videos available through File Sharing in iTunes	77
6.8	Examples of the different types of content supported by the experiment iOS app	78

List of Tables

2.1	Distribution of emotion-labelled images used to build emotion classifier	21
5.1	Mapping of Implementation sections to source code archives and GitHub repositories	39
5.2	Experiment API routes using CRUD model	43
6.1	Details of experiment content used in the validation experiment	63
6.2	Cross-validation results of database-specific emotion classifiers	74

Chapter 1

Introduction

Every day, we view different types of web-based media such as Tweets, online videos, and articles. Each of these types of media can produce a different emotional response such as happiness, anger, or surprise. Automatic classification of these responses has received a lot of attention in the image and video analysis communities, as well as in social computing. For example, AI researchers at Carnegie Mellon University's Human Sensing Laboratory recently released work on new emotion detection techniques [1]. Microsoft's Project Oxford has also recently released new vision APIs for emotion detection in images [2]. This existing work has focused on methods of detecting emotion, whereas this project is more about applying this technology to a real-time application on a mobile device.

1.1 Aims of project

The main aim of this project is to develop a proof of concept system that allows emotion classification experiments to be conducted using a mobile device. Within the scope of this project, an *emotion classification experiment* involves a participant viewing a small number of web-based media items such as YouTube Videos, Tweets, or websites, whilst their emotional response is captured in the background via a mobile device's camera. To validate this aim, an experiment with a small number of participants will be designed and executed using the developed system.

The system will be designed in components that will provide general tools for emotion classification. The main objectives to meet the aims of the project, as outlined in the initial plan [3], are:

- To develop an API for accessing/modifying experiment related data
- To develop an API-centric web application capable of creating experiments and viewing experiment results

- To develop an iOS application capable of performing these emotion classification experiments
- To further develop emotion classifiers to measure different emotions

To provide context, a high-level summary of the work completed for each of these objectives is provided within the Developed System chapter (Page 25).

1.2 Intended Audience

Emotional classification has potential impact in areas such as healthcare, marketing, and social computing. Some example scenarios of how the technology could be used in these areas include:

- *Healthcare* - Monitoring conditions including depression or anxiety based on facial expressions of patients. This technology is already starting to be used to help children with autism to recognise and express emotions through a Samsung developed Android application [4].
- *Marketing* - Gaining insight into consumers through facial reactions to advertisements. Companies including Unilever, PepsiCo [5], and Bentley [6] are already using emotion recognition technology to better understand their customers.
- *Social* - Integrating the technology into social media/dating/video conferencing applications to help users read the facial expressions of those they are interacting with.

This project should particularly help to understand how feasible emotion classification is on a mobile device, and may start to provide insight into how users respond to different types of content. With further development, it is possible that this project could contribute to future research in the University.

1.3 Approach

The approach taken for this project involves working through 3 main stages:

1. *Research stage* - Researching the existing work that has been completed in this area, and sourcing information required for the development stage such as the tool/libraries to use and available emotion databases for emotion classifier training.
2. *Development stage* - Developing the main components that will make up the system including the API, web applications, and iOS application. It will also involve the development of an emotion classifier.

3. *Validation stage* - Validating the developed system through designing, conducting, and analysing a series of user experiments, with guidance from Professor Roger Whittaker and PhD Students in the School of Computer Science & Informatics.

There may be some overlap between these stages, such as when technical research is required during development, or when validation is required of an early version of the system. When working on a particular stage, different pieces of work will be carried out in parallel to help to avoid any problems or delays that may have an impact on the delivery of the project.

1.4 Assumptions

Some assumptions have been made in the project to make it feasible to deliver a proof-of-concept system in the allocated timeframe. The main assumptions are related to the conditions that an emotion classification experiment will be carried out in:

- *Adequate lighting* - An experiment should be carried out in a location that is well-lit so that individual facial features can be detected by a face tracker. Methods of trying to cope with different lighting conditions such as histogram equalisation will be considered, but these methods will only have a limited effect.
- *Pose constraints* - An experiment participant should be facing the mobile device's camera directly. Pose estimation methods have not been considered for the face tracker, and the experiment results may be inaccurate if the experiment is completed using another pose.
- *Device position* - In addition to the pose constraints, an experiment participant should hold/mount the mobile device at a sufficient distance so that their full face is visible by the device's camera.
- *Device platform* - The device used in the experiment will run the iOS platform. Attempting to build an application for multiple platforms in the allocated time would be infeasible. This platform was chosen based on knowledge of the platform and existing work that has been completed.
- *Internet access* - The system's components communicate with each other using an API. Some form of Internet access such as 3G/WiFi, will be required when completing an experiment to retrieve experiment information, to load web-based content, and to send experiment results back to the API.

It is also assumed that part of the project will be based on work completed during a CUROP project undertaken in Summer 2015 [7]. The CUROP project also had face tracking and emotion classification components, built for the iOS platform. Specifically, the code used to set up and interface with a face tracking library will be used. Also, the emotion classifier developed

in the CUROP project will initially be used to test the system, until a new emotion classifier is developed in the latter stages of the project. This classifier will also act as a benchmark when completing further emotion classifier work.

Chapter 2

Background

Within the project, there are two key technologies that drive the emotion classification system - a face tracker and an emotion classifier. Face tracking involves detecting and tracking a set of points of the face. Emotion classification involves using these points, to understand someone's current facial expression, and to use that knowledge to give a set predictions of how likely particular emotions are being expressed. This chapter outlines both of these technologies, including their role in the project, with details provided for why particular decisions were made when using these technologies. It also provides an overview of some of the existing solutions in the current research field related to the project.

2.1 Face tracking

The basis of the emotion recognition in this project is to use facial expressions to give an indication of a person's emotions. In order to know a person's facial expression, a set of key facial landmarks (Fig. 2.1) such as points around the mouth, and jaw, need to be detected and tracked in real time.

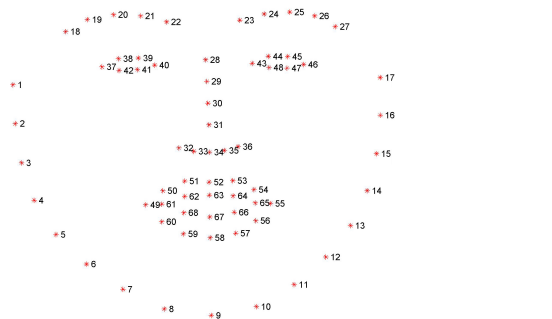


Figure 2.1: A set of labelled facial landmarks that represent a human face

As this tracker will be used for a real-time application on a mobile device, there are some requirements for the face tracker. Firstly, the tracker needs to provide access to the raw facial landmark data in order to calculate features for the emotion classifier. Secondly, for the tracker to be capable of running on a mobile device. It will need to be lightweight in terms of memory and CPU requirements, especially as there will be other tasks running on the iOS app that will be using resources such as streaming web-based content and performing emotion classification. Finally, the tracker will need to be reliable to detect changes in movement. This is particularly important in emotion classification as changes in facial expressions can often involve only subtle facial changes.

There are various approaches that can be taken to track faces. Two approaches that are often considered with this type of research are using an Active Appearance Model (AAM) or using an Active Shape Model (ASM). They both involve trying to fit known statistical models that represent an object such as a face, to an unseen example of the object i.e. an image containing a face. Previous work in the School has found that ASM provides faster and more accurate tracking over AAM, whilst meeting the other outlined tracker requirements [8]. Therefore, an ASM approach will be used in the project for face tracking.

2.1.1 Active Shape Models

An Active Shape Model (ASM) will be used in the project in order to detect and track a participant's face during an experiment. An ASM is a statistical model of the shapes of an object i.e. a face, which iteratively deforms to fit an example of the object in a new image [11]. The shapes within the model are constrained by a Point Distribution Model (PDM). A PDM is used for describing features that have a general shape but are not rigid and are subject to variation. The PDM is built through a training process that involves looking at normalised images, and performing Principal Component Analysis on the facial landmarks within the images. This results in a statistical model that provides the range of variation for each facial landmark to constrain the approximations made by the ASM [12].

The general process of applying an ASM representing a face, to a new image, involves alternating between the following steps:

- Generate a suggested face by looking in the image around each facial landmark for a better position for the landmark. This normally involves looking for strong edges or calculating distance measures.
- Conform the suggested face to the PDM.

This process is repeated until the improvements made from changing facial landmark positions becomes negligible.

The main disadvantage when considering the use of an ASM is the complexity involved in building the underlying statistical model. An ASM is heavily dependent on the training data it is provided with, and it will struggle to recognise any variation not found in the training data. Due to the limited time in the project, a pre-built ASM face tracker will be used - the FaceTracker library, authored by Jason Saragih and maintained by Kyle McDonald [10]. This library has already been trained on more than 750,000 varied faces using the CMU Multi-PIE Face Database [13], which significantly reduces the dependence on training data.

2.1.2 FaceTracker Library

The specific implementation of ASM face tracking that has been chosen for this project is the FaceTracker Library, developed originally by Jason Saragih. The library is written in C++, and provides deformable tracking of 66 facial landmarks. One of the major reasons for choosing this library is that it is open-source. All of the tracking data is accessible from the code, and the library can easily be built for multiple platforms such as iOS/Desktop. The face tracker will be integrated into the iOS application component of the system, to obtain the test features required to complete emotion classification. The face tracker will also be needed for the Desktop environment during the emotion classifier training process, to extract training features from training data.

Another reason for choosing this library is it is robust. An emotion classification experiment will involve viewing multiple items of varying durations, with the face tracker needing to work well for the duration of the experiment. The FaceTracker library is suitable for this application as it is capable of performing well over time and is able to recover well from any failed attempts of tracking through automatically resetting the ASM process. When the library was originally developed, analysis was carried out against other face tracking methods. The results found that FaceTracker library (shown in pink in Fig. 2.2) had the best performance in terms of ability to recover from failures, and overall tracking success.

A potential drawback of using this particular ASM implementation is that the library uses constraints and relationships between certain facial landmarks in order to improve tracking speed. This is particularly noticeable with the facial landmarks that make up the mouth where subtle changes in facial expression, such as moving one corner of the mouth are not detected. Although this limits the accuracy of data that can be detected, the improvements in speed will allow the library to work well on a mobile platform.

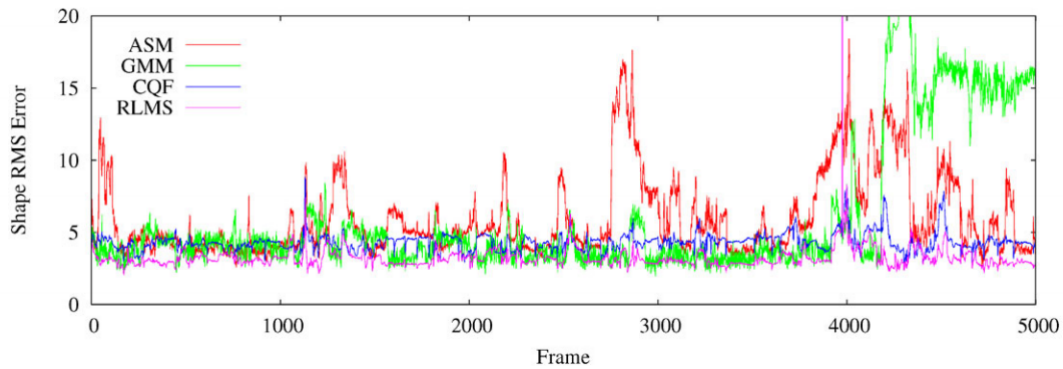


Figure 2.2: Root-mean-squared (RMS) errors between ground truth landmarks and resulting fit from various face tracking methods over 5000 frames [9]

Provided with the library is a pre-trained generalised ASM model. As mentioned in the ASM overview (Page 16), this means that time will be saved through not having to train a new model. However, the provided model is not modifiable, and so there is no possibility of trying to complete further training with new faces to personalise the model to an experiment participant. As the model has been trained on a large database of varied faces, the general model should still be able to recognise most faces that it will encounter. Whilst there has been recent research into using a personalised approach to SVMs [1], this approach will not be considered in this project. Comparing the general approach taken to a personalised approach could be possible future work for this project.

2.2 Emotion Classification

In the scope of this project, emotion classification involves using data from the facial tracker component of the system to obtain probabilistic predictions of how likely particular emotions are being expressed at a given time. One of the main decisions for the project, was deciding which emotions the proof-of-concept system would support. Within Psychology, there are various theories that are used to describe the basic/core emotions. After reviewing existing work in this area, the following emotions have been considered:

- Anger
- Contempt
- Disgust
- Fear
- Happiness
- Natural

- Sadness
- Surprise

This list is made up of the 6 basic emotions stated in Ekman's original List of Basic Emotions [14], as well as the addition of 'Contempt', that was added to the list by Ekman in 1990 [15]. Whilst other emotions appeared in the list in 1990, and are involved in other theories, it is not always possible to encode these emotions via facial expressions. Existing work, particularly databases of emotion-labelled images, have already used this set or a subset of these emotions. 'Natural' has also been considered in the system as this emotion is often available in emotion databases as the individuals who are recorded for this work will start from a neutral expression. Existing work in emotion recognition, particularly emotion image databases, have already used this set or a subset of these emotions. Using this set of emotions will make the work more compatible with existing work in this area, and it will make it easier to source training data.

In order to achieve these predictions, the approach that will be taken will involve calculating a series of distance measures from the facial landmarks provided by the face tracker. These distance measures will indicate the expression the face currently has. There are 86 distance measures in total:

- 17 measures from the jaw points to the centre of the nose
- 5 measures from the left eyebrow points to the centre of the right eye
- 5 measures from the right eyebrow points to the centre of the right eye
- 5 measures from the nose bridge points to the centre of the nose
- 6 measures from the left eye points to the centre of the left eye
- 6 measures from the right eye points to the centre of the right eye
- 18 measures from the mouth points to the centre of the nose
- 9 measures from the mouth points to the centre of the mouth
- 5 measures from the left eyebrow points to the right eyebrow points
- 5 measures from the left eyebrow points to the centre of the nose
- 5 measures from the right eyebrow points to the centre of the nose

These distances are loosely based on the Facial Action Coding System (FACS) [16]. FACS is a system that encodes different human facial movements such as raising the eyebrows, or stretching the lips into Action Codes. A combination of these coded movements can be used

to code any facial expression [17].

This is just one possible approach for classifying emotion. Another possible approach for emotion recognition is Eigenface based recognition. This involves applying Principal Component Analysis to a set of face images in order to extract a set of eigenface images. The weights of these output images can then be used for emotion recognition in unseen images [18]. There is also the Artificial Intelligence approach to emotion recognition that involves deep learning. This approach is briefly discussed during the analysis of existing solutions (Page 22).

2.2.1 Support Vector Machines

The algorithm that has been used for emotion classification in the project is a Support Vector Machine (SVM). An SVM is a popular classification method that finds a linear separating hyperplane between classes. This is achieved through mapping the training features into a higher dimension space to find the hyperplane with the maximal margin, using a kernel function [19]. For this emotion classification task, the classes will be 8 emotions (Page 17). The features used by the classifier will be made up of the outlined distance measures calculated from tracked facial landmarks (Fig. 2.1). In order to normalise all of these measures for the classifier, the measures are normalised through dividing by the distance between the eyes. This distance was chosen as it remains the same when different facial expressions are expressed. It is necessary to normalise the features as there will be a lot variety in the training data across different databases, as well as in the test data. Other distance measure combinations have been previously considered during the CUROP project [7], but when the different feature sets were compared during SVM training, this feature set provided the best accuracy.

The main drawback of using an SVM is like many classifiers, they can be quite sensitive to the training data, and there may be biases towards particular classes. It will therefore be important to use a large enough set of training data for each emotion. Another drawback of using an SVM is that there is still a large amount of research taking place for the optimal design of multi-class classifiers that achieve acceptable results. One of the reasons for choosing the LIBSVM machine learning library [20] to complete the SVM classification is because it has multi-class classification support, amongst other reasons.

LIBSVM

The LIBSVM machine learning library [20] has been used in the project to build a SVM classifier and to test it on unseen data, i.e., the detected facial landmarks of an experiment participant. As mentioned, one of the main reasons for choosing this library is its support for multi-class classification. This means that predictions for the 8 emotions can be achieved through a single classifier, as opposed to having 8 individual classifiers. LIBSVM uses a voting strategy to

achieve this. There are other approaches to multi-class classification but these are beyond the scope of this project.

Another reason for choosing this library is that it is open source. Like with the face tracker, one of the key requirements for the classifier library is access to the data it produces. By having access to the source code of the library, all of the result data can be extracted, and wrappers can be written as necessary for different applications i.e. predicting emotion on the iOS application/training an emotion classifier on the Desktop. The exact process used to build the emotion classifier using LIBSVM will be covered in the Implementation section (Page 61).

2.2.2 Emotion databases

In order to accurately recognise emotion, the SVM requires sufficient training data. The training data required is images of human faces with different facial expressions that have been manually labelled to indicate the emotion being expressed. Before this project began, an emotion classifier was built with 2 databases:

- *Extended Cohn-Kanade Dataset (CK+)*. This database contains 486 short video sequences from 97 individuals, with validated sequence-based emotion labels [21].
- *Karolinska Directed Emotional Faces (KDEF)*. This database contains approximately 4900 images, from 70 individuals expressing 7 different emotions, photographed at different camera angles [22].

Both of these databases were originally developed for psychological and medical research purposes. More specifically, the content of these databases was designed to be particularly suitable for human emotion experiments such as the ones that this project aims to conduct [22]. Both databases contained a large volume of data to train a classifier with. This was important to have sufficient examples of each emotion being expressed. There was also variation amongst the faces that featured in the databases, such as differences in gender, age, and race. This was important to represent the possible variations that may be found on unseen faces. Unfortunately, this classifier had biases towards particular emotions, and struggled with certain faces. To rebuild a more accurate and robust classifier, more training data was needed.

With help from my supervisor, an additional 3 databases were sourced and used to build a new emotion classifier for this project:

- *The Japanese Female Facial Expression (JAFFE) Database*. This database contains 213 images, from 10 individuals expressing 7 different emotions to various degrees [23].

- *Radboud Faces Database (RaFD)*. This database contains approximately 8000 images, from 67 individuals expressing 8 different emotions, with different gaze directions and camera angles [24].
- *GEneva Multimodal Emotion Portrayals - Facial Expression Recognition and Analysis challenge dataset (GEMEP-FERA)*. This database contains 242 short video sequences from 7 individuals, with validated sequence-based emotion labels [25].

These databases were also chosen based on the volume of data they contained, as well as the variety of faces that featured in the databases. The databases were also chosen based on the ease and cost of obtaining the data. All of the databases were available for free for research purposes, and were downloaded from the database websites. Whilst there were other emotion databases available, they often had a cost and/or involved waiting for them to be physically delivered. For example, although the CMU Multi-PIE Face Database [13] contains more than 750,000 images, to obtain it, it would have cost \$350 and it would have had to have been physically sent on a hard-drive. Sourcing this database would have likely caused a delay to the project's timeline and meant that less classifier work could have been completed.

Due to the face tracker library used in the project, only images/videos taken of individuals from a frontal view could be used. Table 2.1 shows the distribution of the training data:

	CK+	KDEF	JAFFE	RAFD	GEMEP-FERA	Total
Angry	90	280	60	402	994	1826
Contempt	36	0	0	402	0	438
Disgust	118	280	58	0	0	456
Fear	50	280	64	402	1024	1820
Happiness	138	280	62	402	1128	2010
Natural	0	280	60	402	0	742
Sadness	56	280	62	402	1474	2274
Surprise	166	280	60	402	0	908
Total	654	1960	426	2814	4620	10474

Table 2.1: Distribution of emotion-labelled images used to build emotion classifier

This distribution shows that there is a significant difference in the data available between emotions. There are 2274 images labelled with 'Sadness', whereas there are only 438 for 'Contempt' and 456 for 'Disgust'. Ideally, there would be more of a normal distribution to this data, but the emotion databases do not always feature all the emotions the system will support. This will need to be kept in mind when conducting experiments and analysing the results.

2.3 Existing solutions

As mentioned in the Introduction (Page 10), emotion classification is a very popular research area, that has the potential to be applied to many different applications. As a result, there are many existing projects that are completing similar work to what is being done in this project. Two particular projects, IntraFace and EmoVu, stood out whilst researching existing solutions, as the work in these projects aligns the most with this project's objectives. Other notable work in this area will also be briefly discussed.

2.3.1 IntraFace

IntraFace is a software package developed by researchers from Carnegie Mellon University and the University of Pittsburgh. They have recently released software with face tracking, pose estimation, and expression functionalities, as well as releasing an application on the iOS App Store to showcase the expression functionality [28]. The main focus of their work has been on developing the approach used to complete emotion classification. IntraFace also uses an SVM, but rather than using a general SVM, they use a personalised approach. This works through simultaneously learning an emotion classifier whilst reweighting the training data that is most relevant to the unseen face [1]. This is a different focus to this project, which is more concerned with the applications of using this kind of technology on a mobile device.

To analyse the IntraFace expression functionality, the IntraFace iOS application was compared to the emotion classification iOS application developed for the CUROP project [7], which uses a very similar approach to this project. When expressing emotions common to both applications such as 'Surprise', there did not seem to be a significant difference in measurement between the applications (Fig. 2.3).

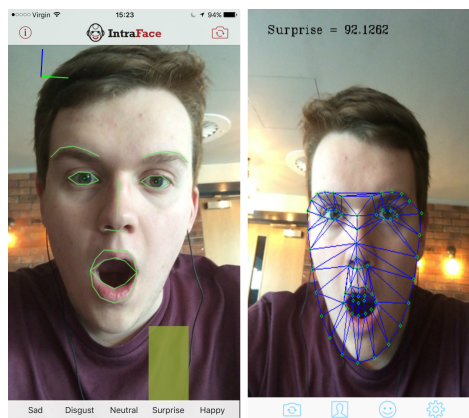


Figure 2.3: Comparison of the IntraFace [28] and CUROP iOS apps [7] when expressing Surprise

However, the IntraFace application only shows the strongest emotion measurement to the user, as a coloured bar, so it is difficult to know how good the measurements are. This lack of data means it is difficult to compare the two approaches practically. On 30 March 2016, the underlying software used in the IntraFace iOS application became available for public use. Unfortunately, this was too late in the project timeline to use, but it has been noted as a potential piece of future work to conduct more tests and comparisons on both these approaches.

2.3.2 EmoVu

EmoVu is commercial emotion recognition software developed by the Artificial Intelligence company Eyeris. They offer a C++ mobile SDK designed to be used in third-party applications to obtain information on a user's emotions as well as other demographic information such as age group and gender. Whilst there was not enough time in the project to test this Mobile SDK, a web-based demo that has the same kinds of functionality was available to test (Fig. 2.4) as well as reviewing their published documentation and research [26].

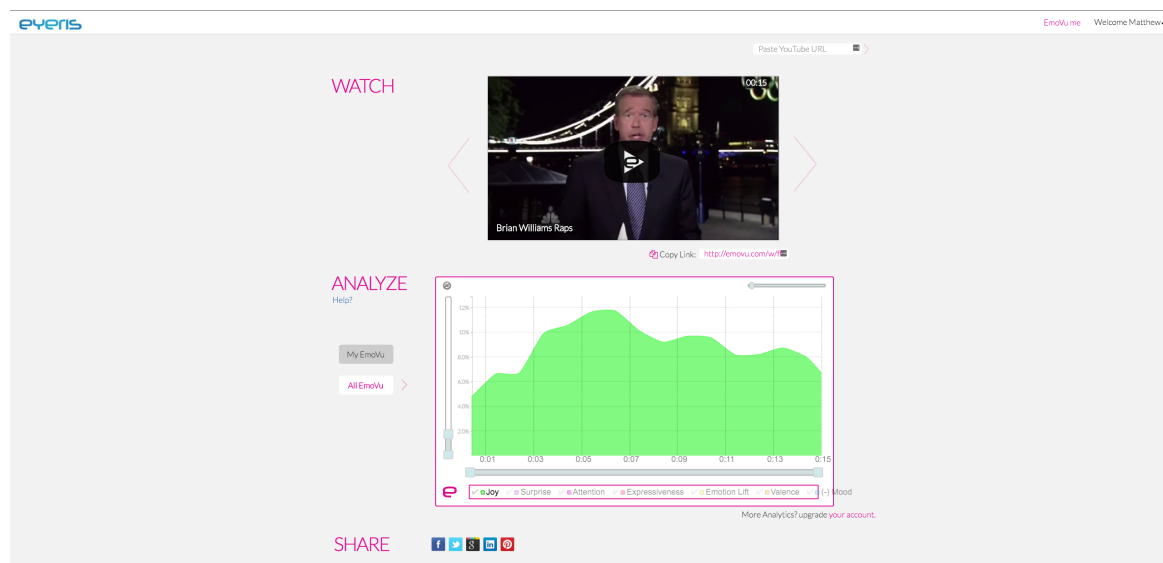


Figure 2.4: Screenshot of the web-based EmoVu demo [27]

The web demo was fairly impressive, with metrics such as engagement and expressiveness available based on the emotion measurements [26]. It is important to note that this demo was completed on a fairly powerful Desktop PC. Without trialling the Mobile SDK, it is difficult to know if such results could be achieved using a mobile device's camera. As the system developed in this project has been designed to be modular, there is scope in the future to trial this SDK with the other tools developed in this project. It is unclear on if there would be any cost or

limitations to using this SDK for research use.

Another key difference between EmoVu and this project is the approach taken to complete emotion classification. As EmoVu is an Artificial Intelligence company, they have used a deep learning approach to develop this software, compared to the SVM approach used in this project. Whilst deep learning has received a lot of attention in the Computer Vision community and would have been an interesting approach, it would have required significantly more data than what could have realistically been sourced and used in the 12 weeks of the project.

2.3.3 Other work

Aside from Intraface [1], and Emovu [26], there are many other examples of work in the current research field that align with this project. Whilst discussing them all is beyond the remit of this report, there are some notable pieces of work to be aware of that have informed this project to some extent.

One of these notable projects is Microsoft Cognitive Services, also known as Project Oxford by Microsoft [2]. They use an Artificial Intelligence approach similar to EmoVu, and have released many Vision APIs for Face [29], Emotion [30], and general Computer Vision tasks [31]. Whilst their Emotion API was available during the project, it currently focuses on support for the Windows platform, which would make it difficult to use in or test this project without a significant amount of extra development work. This work is notable because of the publicity it has gained for emotion recognition in general. Microsoft released a web-based tool to the public that provided emotion scores for images uploaded to their site [32]. The tool became 'viral', and it highlighted some potential use cases for emotion recognition technology.

Another notable project is Affectiva [33]. This is commercial software that has similar functionalities and approach as EmoVu. What is particularly notable about the project is the volume of emotion data they have collected and use in their software. Their website claims to have the world's largest emotion data repository, with nearly 4 million faces analysed, from more than 75 countries. This work informed this project by highlighting the importance of having a large volume of varied emotion data in order to achieve accurate results. Consequently, a significant effort will be made to source different types of emotion databases to use in this project.

Chapter 3

Developed System

This chapter will provide a high level summary of the proof-of-concept system developed for the project. This will be achieved through providing an overview of the work completed in the project with respect to each of the project's main objectives (Page 10), with screenshots where necessary. The purpose of this chapter is to provide context for the rest of this report. Subsequent chapters will then discuss the details of the design, implementation, and evaluation of this system.

3.1 An API-centric web application capable of creating experiments and viewing experiment results

Before being able to conduct an emotion classification experiment on a mobile device, information such as the web-based media that a participant will view, and the duration of the experiment needs to be defined. To meet part of this objective, an API-centric experiment creation web application was developed to allow a user to create an experiment that can be conducted using the system (Fig. 3.1).

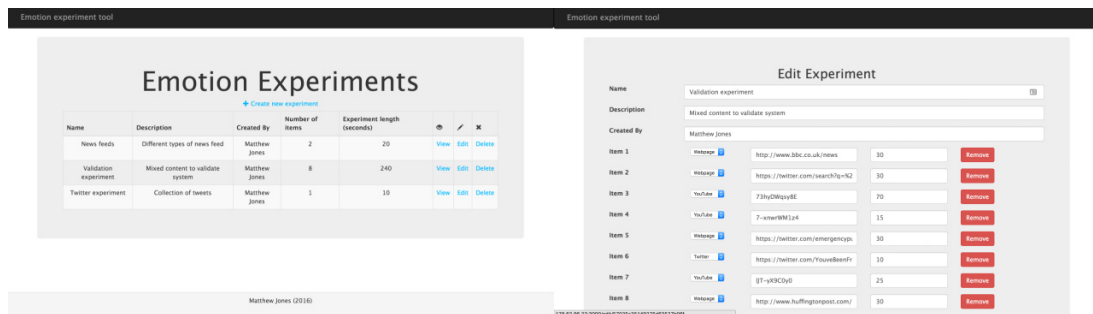


Figure 3.1: Screenshots of the developed experiment creation web application

This web application provides basic facilities that allow a user to add multiple experiment items of varying durations, as well as some summary information about the experiment such as the creator and a brief experiment description, through a form.

To meet the other part of the objective for viewing experiment results, an experiment results web application was developed (Fig. 3.2).

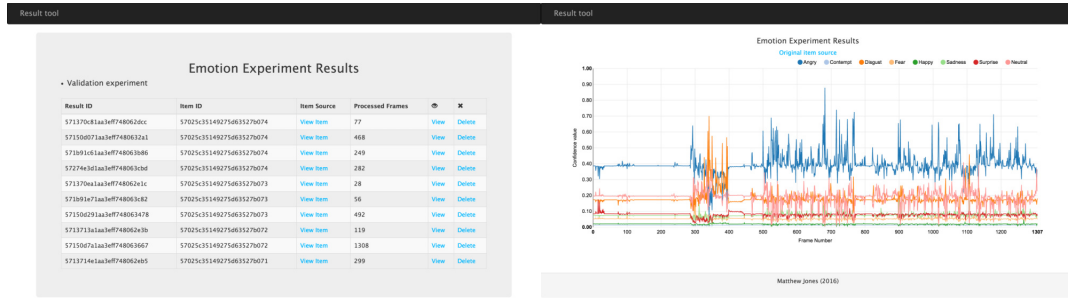


Figure 3.2: Screenshots of the developed experiment results web application

This web application provides an interface to the results captured during experiments. Its main purpose is to allow the downloading of experiment data, and to visualise the emotion predictions recorded during an experiment.

3.2 An API for accessing/modifying experiment related data

The system developed is data-driven, with data being generated by both the creation of experiments, and the results recorded during an experiment. To meet this objective, and to support the previously discussed web applications, 2 APIs have been developed.

The first API that has been built is used to manage all information relating to an experiment (Fig. 3.3).

As seen in the Experiment API example, for each experiment, there are summary fields such as the experiment name and description as well as an array of experiment items. Each of these items have fields for display time, data source, and content type.

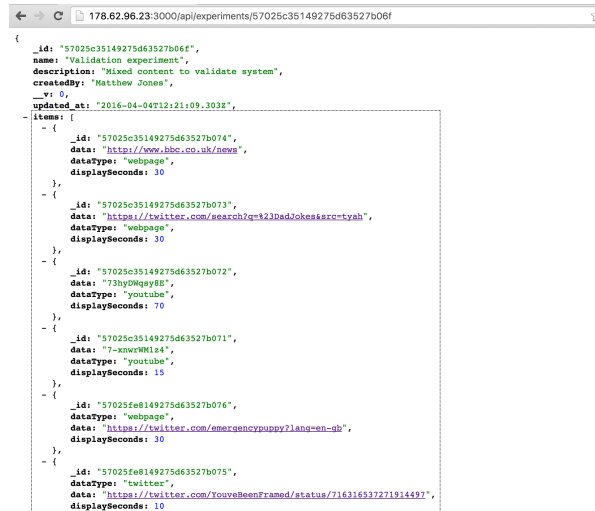


Figure 3.3: Screenshot of an Experiment API GET request

The second API that has been built is used to manage all of the information relating to the results of each experiment item (Fig. 3.4).



Figure 3.4: Screenshot of a Result API GET request

As seen in the Result API example, this includes information on the experiment item that corresponds to that particular result, as well as the main field of this API, the result data field, that is used to store all of the emotion predictions at a frame-level, and is also capable of storing all of the facial landmark data captured by the face tracker.

3.3 An iOS application capable of performing these emotion classification experiments

The main component of the system that has been developed is an iOS application capable of conducting emotion classification experiments with participants (Fig. 3.5).

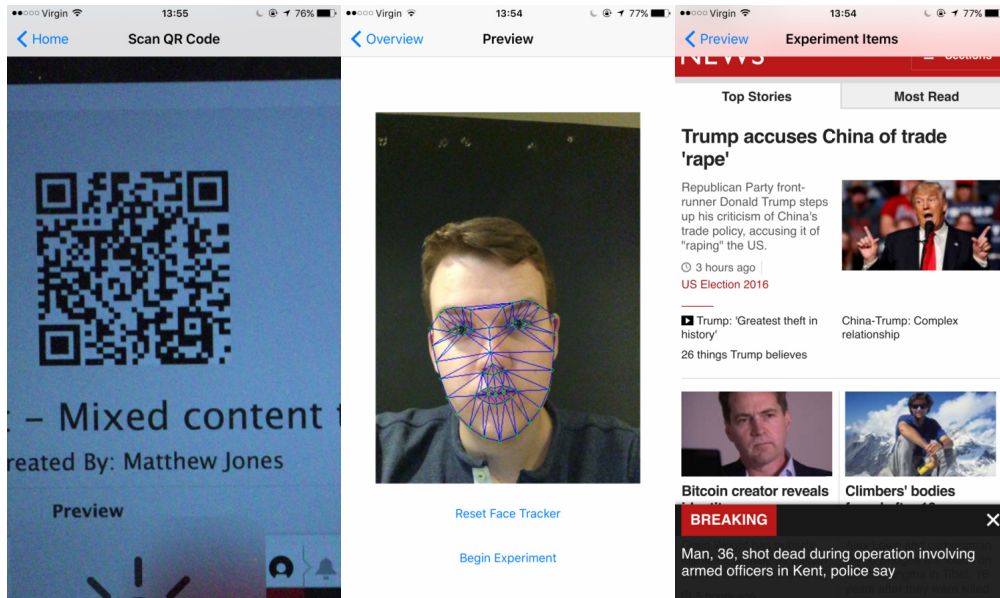


Figure 3.5: Screenshots from the iOS application used to conduct emotion classification experiments

In order to meet this objective, the application needed a variety of functionalities. Initially, it reads in a QR code generated by the experiments creation web application, in order to access the experiment information it needs from the Experiment API. It then has to retrieve and display each web-based experiment item, for predefined amounts of time, in an appropriate view such as in a YouTube player or a browser. Whilst the experiment items are being displayed to a participant, a facial capture system is running in the background of the application to acquire the facial landmarks of the participant, using the front camera of the mobile device. The captured facial landmarks then act as input for an emotion classifier, that will produce a set of emotion predictions for each captured frame. All of the emotion predictions for an experiment are then sent back to the Result API.

3.4 Emotion classifiers to measure different emotions

The last objective of the project was to further develop emotion classifier work that had been started during a CUROP project [7]. A significant amount of time in the project has been spent developing classifiers capable of accurately and robustly classifying emotion. A Python-based training pipeline was developed to acquire features from all of the data within the obtained emotion databases (Page 20), and to use these training features with the LIBSVM library [20] to train a multi-class SVM classifier. The classifier with the best training cross-validation results was then integrated into the iOS application for the experiment phase of the project.

Specification & Design

This chapter provides details relating to the design of the system. This includes outlining the main experiment process, a set of user requirements, the functionality required of each of the system's components, user interface design, and API design.

4.1 System design

After outlining the main components that would make up the system, it was necessary to consider how all of this work would fit together into a coherent system. This was achieved through designing the main experiment process that the system would need to complete (Fig. 4.1).

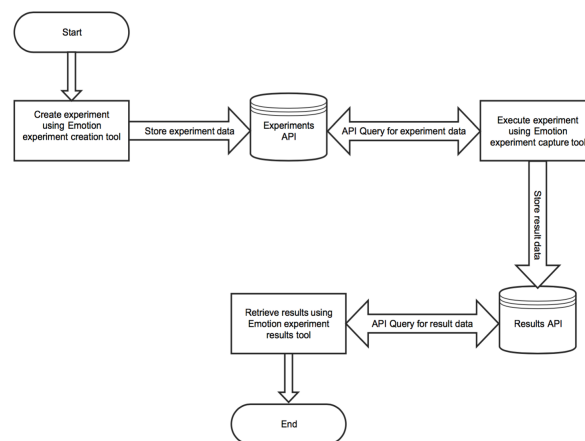


Figure 4.1: Abstract flow chart for the experiment process

This flow chart shows how a typical experiment will be carried out using the system. The first step in the flow chart is to design the experiment using the *Emotion experiment creation*

tool. This will be a web application that will provide the user with multiple input fields, to specify the information necessary to create an experiment. The main responsibility of this component is to set up the experiment so that the other components can access this data and produce results for the experiment items.

After creating an experiment, an experiment will be conducted with participants using an *Emotion experiment capture tool*. This will take the form of an iOS application, and it will handle the execution of a designed experiment. It will use data from the experiment creation tool to load in experiment items, and to capture results such as emotion predictions and facial tracking data using the mobile device's camera. This is where the main work of the system will take place, and where all of the meaningful data will be captured.

Once a suitable number of experiment participants have completed the experiment, the process will finish with viewing/analysing experiment results. This will mainly involve looking at the emotion predictions recorded during the experiments through the *Emotion experiment results tool* in a meaningful way such as through charts or tables.

Each component will not have direct communication with each other, but will use Application Programming Interfaces (APIs) to access/modify necessary data. This approach of encapsulating components has been used so that there no component relies on the implementation of another component, only the API. By designing the process in this way, individual components can be reused/changed/replaced as modules in the future, with no side effects for the rest of the system as long as the components continue to interface via the API.

4.2 User requirements

Before starting the development of the outlined system, a set of basic user requirements were created. These requirements outline the expectations of the system and will be used during Evaluation (Page 76). As the system is intended to be a proof-of-concept, different levels of requirements have been specified, in order to prioritise particular system expectations over others.

Must Have...

- A means of a user setting up an emotion experiment to be carried out. This will involve specifying information such as what the experiment items will be and how long each item will be viewed for.
- An iOS application capable of carrying out these emotion experiments. This will involve tasks such as displaying content, and recording emotional response.

- A means of a user collecting results from an experiment. This will involve a facility that allows you to obtain all results for a particular experiment.

Should Have...

- A facility that allows the analysis and visualisation of results
- A means of recording experiment participants to review with experiment results
- Support for different types of web-based content that can be used as an experiment item

Could Have...

- A login system to allow an experiment and the subsequent results to be viewed only by the user who set up the experiment
- Emotion metrics based on the emotion measurements gathered during an experiment
- Synchronisation of original experiment items and result data when viewing experiment results

4.2.1 Emotion experiment creation tool design

The emotion experiment creation tool is required in the system in order to allow a user to specify the content they would like participants to view during an experiment, and for how long. Whilst it would be possible to manually hardcode this information inside the experiments iOS application, it would limit the functionality and usability of the system considerably. By building a basic web application, that is API-centric, a user is able to create/modify/delete multiple experiments that are all capable of running on the experiments iOS application with no configuration changes required in the application.

To understand the functionality required of this tool, a basic sitemap was produced using Creately [34] to design the structure of the web application (Fig. 4.2).

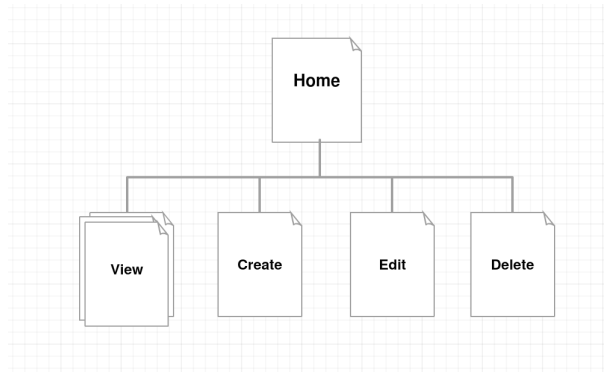


Figure 4.2: Sitemap design for the experiment creation web application

This diagram outlines the main views of the application:

- **Home** - This view will provide a listing of all of the available experiments. Information such as total experiment length, and experiment name will be available in a table. This view will also provide the experiment specific links for the 'Edit' and 'Delete' views, and a means of navigating to the 'Create' view.
- **Create** - This view will contain a form that will provide all of the necessary input fields to create an experiment such as text boxes and dropdown boxes. Once the form has been submitted, a user will be redirected to the 'Home' view and will see the newly created experiment listed.
- **Edit** - This view will provide similar input fields as the 'Create' view, but the components will already have been loaded with the experiment information the user wants to edit. Some form of experiment identifier will be passed to this view to load this information. Once the information has been updated, a user will be redirected to the 'Home' view and will see the modified experiment listed.
- **Delete** - This view will be used to delete a created experiment from the system. Once confirmed, the user will be redirected back to the 'Home' view, where the experiment will no longer be listed.
- **View** - This view will provide a summary of all the information for a single experiment. Some form of experiment identifier will be passed to this view to load the experiment specific information such as the name, creator, and items. The view will also present some kind of code that can be inputted into the experiment capture tool.

4.2.2 Emotion experiment capture tool design

The main aim of the project is to carry out experiments on a mobile device. As such, this component will be the most important in the system, and as already stated, will be developed

as an iOS application.

A basic flow chart has been created to define the experiment process that will be carried out by the application (Fig. 4.3). These tasks will be used to design the user interface, and will be used to structure the development of the application.

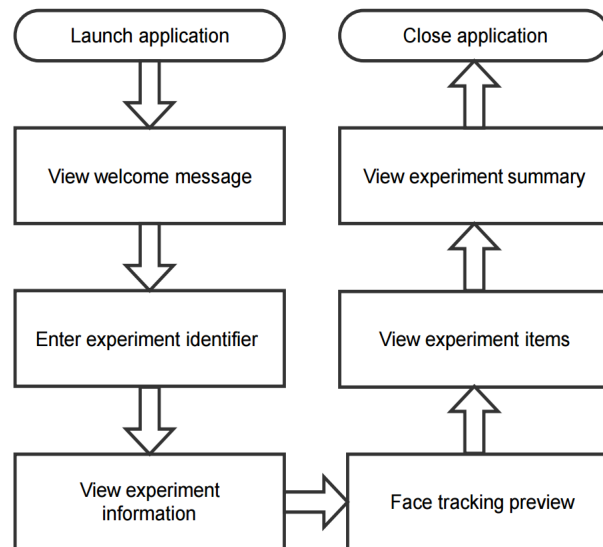


Figure 4.3: Abstract flowchart of the main tasks involved in an experiment on the iOS application

The main tasks are:

- **View welcome message** - A short summary of the application when a user launches it. There will be an option to proceed to the next screen once this information has been read.
- **Enter experiment identifier** - The participant will be asked to enter an experiment identifier from the experiment creation tool, to retrieve experiment information. This will likely involve using the camera to scan a code, as typing an identifier may be time consuming. Once entered, the next screen will load automatically.
- **View experiment information** - Once experiment information has been retrieved, a summary of the experiment will be shown including experiment name and creator. There will be an option to proceed to the next screen once this information has been read.
- **Face tracking preview** - A preview screen that will show the front camera view overlaid with the currently tracked facial landmarks. This will allow a user to ensure the device is positioned appropriately for the tracker to detect landmarks as expected. Once a user is happy with the tracker output, there will be an option to begin the experiment.

- **View experiment items** - This will involve loading each of the experiment items. Content will be loaded differently according to its type, and the content will be shown for a varying number of seconds defined during experiment creation. Whilst a user is viewing this content, at each frame captured by the front camera, the face tracker will be applied, and the resultant data will be used to predict emotion measurements using an emotion classifier. The data will then be sent to the Results API after each experiment item.
- **View experiment summary** - Once the experiment has ended, a short summary will load to indicate to the participant that no more items will be displayed.

4.2.3 Emotion experiment results tool design

The emotion experiments results tool is required in the system in order to provide an interface to the Result API. Although all of the result data will be accessible through the Result API, the volume of data that this API will hold means that a basic web interface will be a better solution. Ideally this tool should also offer means of querying/analysing/visualising results, however some of this functionality may need to be sacrificed over other development/experiment work. At a minimum, it will be designed to allow a basic means of retrieving/viewing results.

To understand the functionality required of this tool, a basic sitemap was produced in Creately [34] to design the structure of the web application (Fig. 4.4).

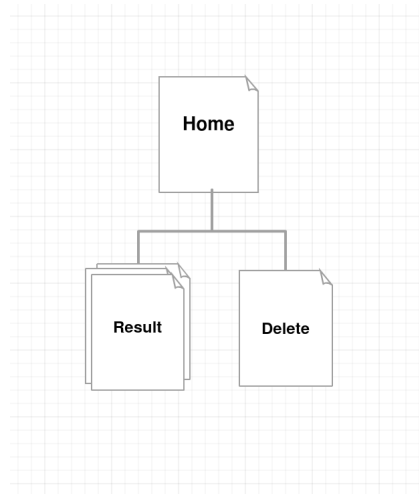


Figure 4.4: Sitemap design for the experiment results web application

- **Home** - This view will provide a summary of all of the available experiment results. There should be some form of sorting to this view, such as by experiment or by participant. A

summary of each result including the experiment and experiment item identifier will be available, along with result-specific links to the 'Result' view.

- **Result** - This view will display all of the available for a single result. This will likely be displayed visually such as through a line chart, due to the amount of data per result. There will also be tools to download the stored data. Some form of result identifier will be passed to this view to load the requested result.
- **Delete** - This view will be used to delete an experiment result from the system. Once confirmed, the user will be redirected to the 'Home' view, where the result will no longer be listed.

This web application will be developed after the Emotion experiment creation web application using a very similar approach and tools. As this has application has less requirements than the creation tool, and shares some of the basic functionality, this component of the project should not take long to develop.

4.3 API design

The system will be data-driven, and will need well-designed API's in order to manage all of the data generated through experiments. After considering the applications that the API would be used for, it was decided to split the API into 2 API's - one that would model an *Experiment*, and one that would model an *Result*. By doing this, it reduces the complexity of the API work, through not having to model a result and an experiment together. It also makes querying and other API actions easier for the distinct applications of the system. Although there will be 2 APIs, they will be implemented in a very similar way, and to make it easier as a proof-of-concept, they may share some data fields such as experiment item details.

4.3.1 Experiment API

The Experiment API is used by the Emotion experiment creation tool to store all experiment data. It is then used by the Emotion experiment capture tool to retrieve this data to display the relevant information to a user during an experiment. After considering the information needs of these applications, an Experiment data model was designed (Fig. 4.5).

Each experiment instance represents a single experiment that has been created by a user. A small number of experiment summary fields are used to provide an experiment participant with some context - *ExperimentName*, *ExperimentDescription*, and *CreatedBy*. The date and time of the most recent update to experiment information is also stored in a *UpdatedAt* field.

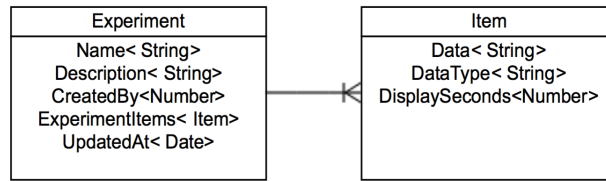


Figure 4.5: Abstract data model for an Experiment

The main part of an Experiment is the *ExperimentItems* field. As shown in the diagram, the *ExperimentItems* field of a single experiment will be made up of one or more *Item* objects. Each *Item* object stores all of the information necessary to display a single experiment item to a user in an experiment. The *Data* field provides the source of the content, which will most likely be a URL. The *DataType* field has been included to distinguish different types of content such as YouTube videos or a Web page. Finally, the *DisplaySeconds* is a field that allows a user to specify how long they would like an experiment participant to view an item.

4.3.2 Result API

The Result API is used by the Emotion experiment capture tool to store all of the generated result data. It is then used by the Emotion experiment results tool to present this stored data to a user. After considering the information needs of these applications, a Result data model was designed (Fig. 4.6).

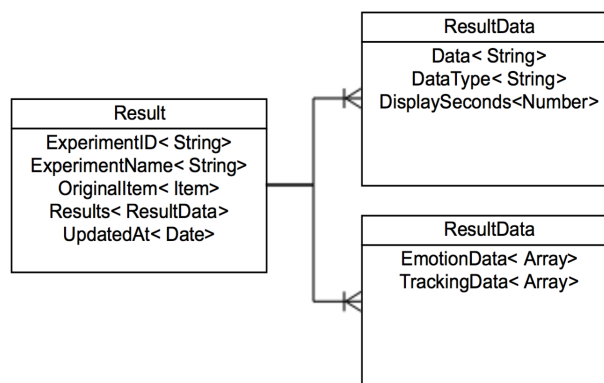


Figure 4.6: Abstract data model for a Result

Each Result instance represents the results of a user viewing a single experiment item. For

contextual purposes, fields from the experiment that generated the results are stored - *ExperimentName*, *ExperimentID*, and *OriginalItem*. Whilst this information could be looked up through the Experiment API later, as an experiment can be edited at any point, it is important to retain the information used in that particular experiment, to enable accurate analysis and comparison of results. The date and time that the result was recorded is also stored, to help with the analysis of results.

The main part of a Result is the *Results* field. As shown in the diagram, the *Results* field of a single Result will be made up of one or more *ResultData* objects. Each *ResultData* object has 2 fields - *TrackingData* and *EmotionData*. For each frame processed during an experiment, an array entry will be created in *TrackingData* to record the co-ordinates of the facial landmarks in the frame. This entry will be an array of numbers representing the 66 tracked landmarks. An array entry will also be created in *EmotionData* for each frame, to record the 8 emotion measurements obtained from the emotion classifier. This entry will be 8 key-value pairs comprised of the emotion i.e. 'Happy' or 'Sad', and the corresponding measurement from the classifier.

4.4 User interface design

The main component that will be interacted with by users is the emotion experiment capture tool. To ensure that this tool supported the user requirements, a basic wireframe design of the iOS application (Fig. 4.7) was designed using the Balsamiq Mockups wireframing software [35].

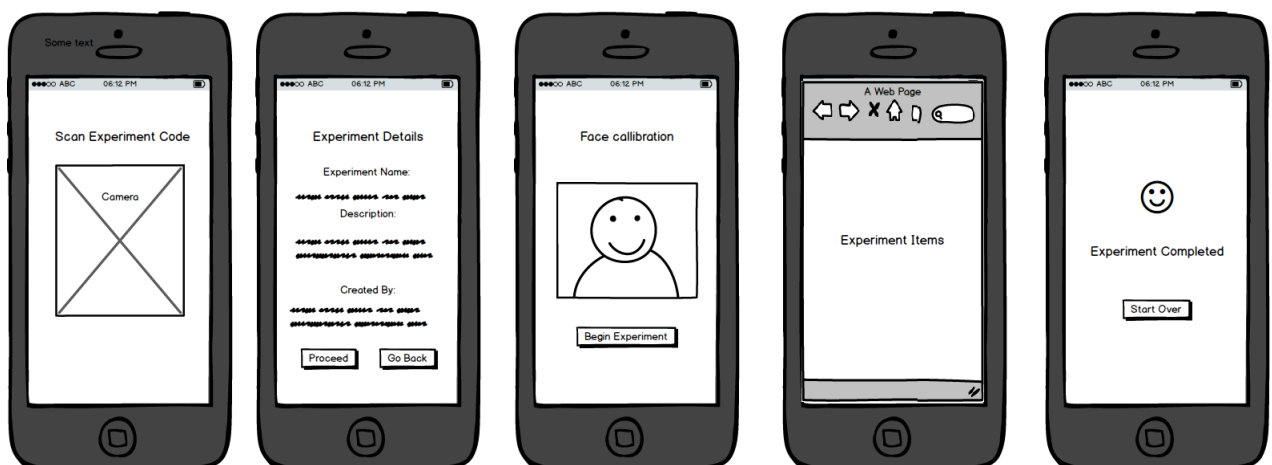


Figure 4.7: UI mockup for the emotion capture iOS application

This wireframing process involved prototyping different ideas for how a participant would interact with the iOS application during an experiment, and what content would need to be shown. This process structured the application into views, and outlined the main UI elements that would be needed. Some of the key design decisions that were made as a result of these mockups were:

- *How a user would retrieve experiment information?* After initially designing the application with a text input box where you could manually enter an experiment identifier, it was decided that this would take too long for a large number of participants. Instead, the application was designed so that a user scans a generated experiment QR code using the device camera.
- *How a user could check that the face tracker was working?* It was important that a user was able to see if the face tracker was working before viewing any experiment items. Initially, the option of being able to view a preview at any point in the application was considered but this complicated the application's design, and it would not have been compatible with all of the interface components such as full screen video players. Also, a user may have been more focused on the preview throughout the experiment, instead of viewing the experiment items. Instead, a separate calibration view was designed that allowed a user to view what the face tracker was detecting before starting an experiment.

Having this design provided a specification to work to during development, and although design changes were made after receiving feedback, time was saved through making design decisions using these inexpensive mockup tools compared to making these decisions during development.

As the emotion experiment creation tool and emotion experiment results tool will be accessed by only a small number of people, and have very basic requirements, there will be less of a focus on designing a custom user interface for these components. The user interface for these components will rely on the popular front-end framework, Bootstrap [36]. This framework provides standard HTML elements and page templates that mean that even if the design of these components were to drastically change, it will involve not much work to change.

Chapter 5

Implementation

This chapter will document the development of each of the system's components. Although the emotion classifier is part of the Emotion experiment capture tool, a significant amount of work was completed, and will be discussed in a separate section. The source code for all work has been attached as zip files. The code for each component is also available through GitHub repositories. Table 5.1 provides the mappings of these implementations to the report sections.

Report section	Implementation	GitHub Repository
Emotion experiment creation tool	ExperimentCreationApp.zip	https://github.com/mrhysjones/experiments-creation-app
Emotion experiment capture tool	ExperimentIOSApp.zip	https://github.com/mrhysjones/emotive-web
Emotion experiment results tool	ExperimentResultApp.zip	https://github.com/mrhysjones/experiment-results-app
Emotion classifier	EmotionClassifier.zip	https://github.com/mrhysjones/emotion-classifiers

Table 5.1: Mapping of Implementation sections to source code archives and GitHub repositories

5.1 Tools used

5.1.1 Emotion experiment creation tool - Tools

The experiment creation web application was developed mainly in the JavaScript language using the popular MEAN development stack [37]. There are 4 components to this stack:

- **MongoDB** - A document-based NoSQL database that will hold all of the experiment data. MongoDB is designed to make it easy to make changes to the database's schema and configuration, without any significant issues. This was an important attribute, as this is a research project that may require several changes throughout development, with ideally little time needing to be spent on database administration [38].

- **ExpressJS** - A web framework designed to work with the Node.js framework. This framework provides several HTTP utility methods that allow an API to be developed easily [39].
- **AngularJS** - A web application framework that extends HTML functionality and is used for binding data to HTML. For example, AngularJS can be used to bind experiment API data to an HTML table. One of the functionalities the framework was chosen for is *ng-repeat* that can be used to clone HTML elements for each item in a collection such as a list of experiment names [40].
- **NodeJS** - A server-side environment that allows JavaScript to be used on the back-end as well as on the front-end.

This software bundle was used to quickly build a functional web application, as well as the Experiment API. The 4 components are designed to work well with each other, and there are commands that can be used to bootstrap some of the code in the project.

The Bootstrap front-end library [36] was used to develop the web application's user interface. The library contains HTML and CSS-based templates, that were used to develop the application's views, as well as custom web components that were used to support the main functionality of the application such as tables, and form controls.

The qrcode-generator library [41] was used in the project to provide the experiment QR-code functionality required. This is a standard library used for generating QR codes that has support for many programming languages, and that avoids having to write this functionality from scratch. By providing the library with data i.e. an experiment identifier, it is able to generate and display a QR code with this information on demand.

5.1.2 Emotion experiment capture tool - Tools

The emotion experiment iOS application was built in XCode, using the Objective-C language. This language was chosen based on compatibility with the libraries used in the project, familiarity of the language, and existing work already completed in Objective-C. Writing the project in the recently released iOS language, Swift, was considered, but the learning curve, and lack of library support would have made it considerably more difficult to achieve the required functionality. Several built-in iOS frameworks were used in the project such as AVFoundation for working with the device camera and CoreImage for image processing.

The FaceTracker C++ library [10] was used in the project, with a previously developed wrapper, to perform all of the face tracking functionality. The LIBSVM C++ library [20] was used in the project, with a previously developed wrapper, to perform all of the emotion

classification, such as scaling and predicting test data, using the developed emotion classifier.

To display different types of content, third-party libraries were used. The Fabric library, developed by Twitter [44], was used to display Tweet experiment items. Tweets could have been displayed on the app without this library, however this library ensures that Twitter guidelines are met, and it has built-in functionality that allows a user to interact with displayed Tweets. The YouTube IFrame Player API [43] was used to handle displaying YouTube videos.

Finally, the CEMovieMaker library [42] was used to support the experiment participant recording functionality. It provides a means of generating a .mov video file from an array of raw video frames.

5.1.3 Emotion experiment results tool - Tools

The experiment results web application was built using the same tools as the Emotion experiment creation tool (Page 39).

In addition to those tools, the Angular-nvd3 charting library [45] was used in this component in order to produce line charts of experiment results. This library was chosen as it was developed for the AngularJS framework, already used in the project, and it provides extensive chart options through an API.

5.1.4 Emotion classifier - Tools

To train an emotion classifier, a series of Python scripts were written to carry out different training stages such as extracting features, performing Principal Component Analysis, and classifier cross-validation/training using LIBSVM binaries [20]. To assist with the Principal Component Analysis, the PCA module from the matplotlib Python library was used [47]. The training data for the classifier was obtained using the FaceTracker C++ library [10]. Some basic computer vision tasks such as extracting video frames were achieved using the OpenCV Python library [46].

5.2 Emotion experiment creation tool

5.2.1 MEAN stack setup

To use the MEAN development stack in the project, there were some prerequisites that had to be installed. The following software had to be installed before any of the work could be completed:

- *NodeJS* - The server-side environment to be used in the project
- *npm* - The package manager used to install and manage packages
- *ExpressJS* - Web application framework that runs on NodeJS
- *MongoDB* - Document-oriented NoSQL database that will store all Experiment data

The installation of this software is relatively simple and well documented online. The specifics of how the software was installed will not be discussed here.

After setting up all of the prerequisites, the Express generator in the ExpressJS framework was used to bootstrap the basic structure and files required for this component (Listing 5.1).

Listing 5.1: Bootstrapping the Experiment creation tool using ExpressJS

```
1 express -e experimentApp
```

This command set up the basic structure and code for both the API and the web application work including views, routes, and stylesheets. Whilst some tweaking was required to this structure throughout development, it was much quicker than having to set up all of these files and folders manually.

5.2.2 Experiment API

In order to build an API, an Experiment Model had to be defined based on the model designed in API Design (Page 35). The Mongoose API tool was used to model the Experiment object (Listing 5.2). Whilst MongoDB is schemaless, Mongoose provides API validation and enforces the schema to keep a consistent structure.

Listing 5.2: Experiment Mongoose model

```
1 var mongoose = require('mongoose');
2
3 var ExperimentSchema = new mongoose.Schema({
4   name: {type: String, required: true},
5   description: {type: String, required: true},
6   createdBy: {type: String, required: true},
7   items: [{
8     data: {type: String, required: true},
9     dataType: {type: String, enum: ['twitter', 'youtube', 'webpage'], required: true},
10    displaySeconds: {type: Number, default: 30}
11  ]},
12   updated_at: { type: Date, default: Date.now },
13 });
14
15 module.exports = mongoose.model('Experiment', ExperimentSchema);
```

The *name*, *description* and *createdBy* fields are strings, that are all required when providing experiment information. The *items* field is an array that can hold one or more experiment items. Each object in the *items* array, will contain a *data* string containing the item source, a string enum, *dataType*, that must take the value of 'youtube', 'twitter', or 'webpage' to indicate the item type, and a number field, *displaySeconds*, to indicate how long the item will be displayed for during an experiment. The *updated_at* field has been included to keep a record of the last modification made to an experiment's information.

After writing this model, some basic API routes were defined using the router module within ExpressJS. The routes developed were based on the Create-Read-Update-Delete (CRUD) model (Table 5.2).

Resource (URI)	POST (Create)	GET (Read)	PUT (Update)	DELETE (Delete)
/api/experiments	create new experiment	list experiments	error	error
/api/experiments/:id	error	show experiment :id	update experiment :id	delete experiment :id

Table 5.2: Experiment API routes using CRUD model

Mongoose provides a query API, and the Router module in ExpressJS provides a means of specifying these different requests for particular routes. Listing 5.3 shows all of the code required to set up the GET /api/experiments/:id route, used to retrieve a specific experiment from the API.

Listing 5.3: /api/experiments/:id GET route

```

1 var express = require('express');
2 var router = express.Router();
3 var Experiment = require('../models/Experiments.js');
4
5 /* GET /api/experiments/id */
6 router.get('/:id', function(req, res, next) {
7   Experiment.findById(req.params.id, function (err, post) {
8     if (err) return next(err);
9     res.json(post);
10  });
11 });

```

Lines 1-3 instantiate the ExpressJS Router module and the Experiment Mongoose model. Line 6-7 retrieve the experiment ID from the API request and use the ID to retrieve the requested experiment using the Mongoose query API. If there is an error with this query, Line 8 will send an error as the API response. Otherwise, Line 9 will return the JSON response containing the requested experiment.

A similar piece of code was used for the /api/experiments/:id PUT route, used to update a specific experiment (Listing 5.4).

Listing 5.4: `/api/experiments/:id` PUT route

```
1 router.put('/:id', function(req, res, next) {
2   Experiment.findByIdAndUpdate(req.params.id, req.body, function (err,
      post) {
3     if (err) return next(err);
4     res.json(post);
5   });
6 });
```

The main difference for this route compared with the GET `/api/experiments:id` route is Line 2. The *findByIdAndUpdate* method is used from the Mongoose query API, and the request body, *req.body*, is used to update the experiment's JSON data.

By using the Mongoose query API, and the ExpressJS Router module, all of the required API routes did not require code any more complex than these 2 examples.

5.2.3 Experiment creation web application

After developing all of the API routes, a basic web application was built, using AngularJS, that would consume the Experiment API.

'Home' View

This view needed to provide a list of all created experiments from the Experiments API. A custom AngularJS experiment service was created using AngularJS `$resource`, capable of interacting with all of the Experiment API's routes (Listing 5.5).

Listing 5.5: AngularJS service implemented to interact with the Experiment API

```
1 angular.module("ExperimentService", []).factory('ExperimentService',
      ['$resource', function($resource){
2     return $resource('/api/experiments/:id', null, {
3       'update': { method:'PUT' },
4     });
5   }])
```

The majority of API requests are handled by default by the `$resource` library, with only the PUT (Update) route needing to be manually specified on Lines 2 and 3. After this service was created, a controller for the 'Home' view was created that could make a GET request using this service and store all the experiment data needed. An AngularJS table was then used to list the following fields for each experiment:

- Experiment name
- Experiment description

- Experiment creator
- Number of items (length of *items* array)
- Experiment length (total of the *displaySeconds* field)

As well as this information, for each experiment, buttons were added to allow a user to view/edit/delete an experiment.

'Create' View

This view involved created a basic AngularJS form with all of the input controls needed to make a POST request to the Experiment API (Fig. 5.1).

Figure 5.1: AngularJS form developed to facilitate experiment creation

The main challenge with this view was allowing a user to add one or more experiment items. To achieve this, 2 short methods were created to handle adding/removing an experiment item to the form through pushing and splicing elements of the *experimentItems* array that holds all of the item information (Listing 5.6).

Listing 5.6: Code to handle adding/removing experiment items from an experiment

```

1 $scope.addExperimentItem = function(){
2     var newItem = $scope.experimentItems.length + 1;
3     $scope.experimentItems.push({'dataType': 'youtube'});
4 }
5 $scope.removeExperimentItem = function(index){
6     $scope.experimentItems.splice(index,1);
7 }

```

The 'Edit' view used a very similar view and logic as the 'Create' view, with the form data preloaded through a GET request to the developed experiment service, and a PUT request used on form submission instead of a POST request.

'View Experiment' View

To view an individual experiment, a controller was implemented to retrieve an experiment with a particular ID from the Experiments API. The majority of this view is just outputting experiment fields using similar techniques as the 'Home' view. The only extra requirement for this view was to generate a QR code containing the experiment ID. This was achieved using the qrcode-generator library [41] (Listing 5.7).

Listing 5.7: AngularJS directive used to generate experiment QR code

```
1 <qrcode data="{{currenthost}}/api/experiments/{{experiment._id}}" size  
   =150></qrcode>
```

This QR code will be scanned by the emotion experiment iOS application to retrieve experiment information.

5.3 Emotion experiment capture tool

5.3.1 Application interface

Before any of the key functionality was added to the application, the application's interface was created in a XCode Storyboard (Fig. 5.2).

From this storyboard, 7 view controllers were specified:

- **HomeController** - View used to show welcome text on launch of the application
- **QRCodeReaderController** - View used to display and scan an experiment QR code using the device camera
- **ExperimentSummaryController** - View used to provide a summary of the experiment that has been retrieved from the API
- **TrackingPreviewViewController** - View used to display a preview of the face tracker output overlaid on a camera view
- **ExperimentItemViewController** - View used to display different experiment items to a user
- **ItemIntervalController** - View used as an interval between experiment items
- **ExperimentEndController** - View used to inform a user that the experiment has finished

The navigation between these view controllers was also set up in this Storyboard through specifying segues (transitions between views) and through using an iOS Navigation Controller [48].

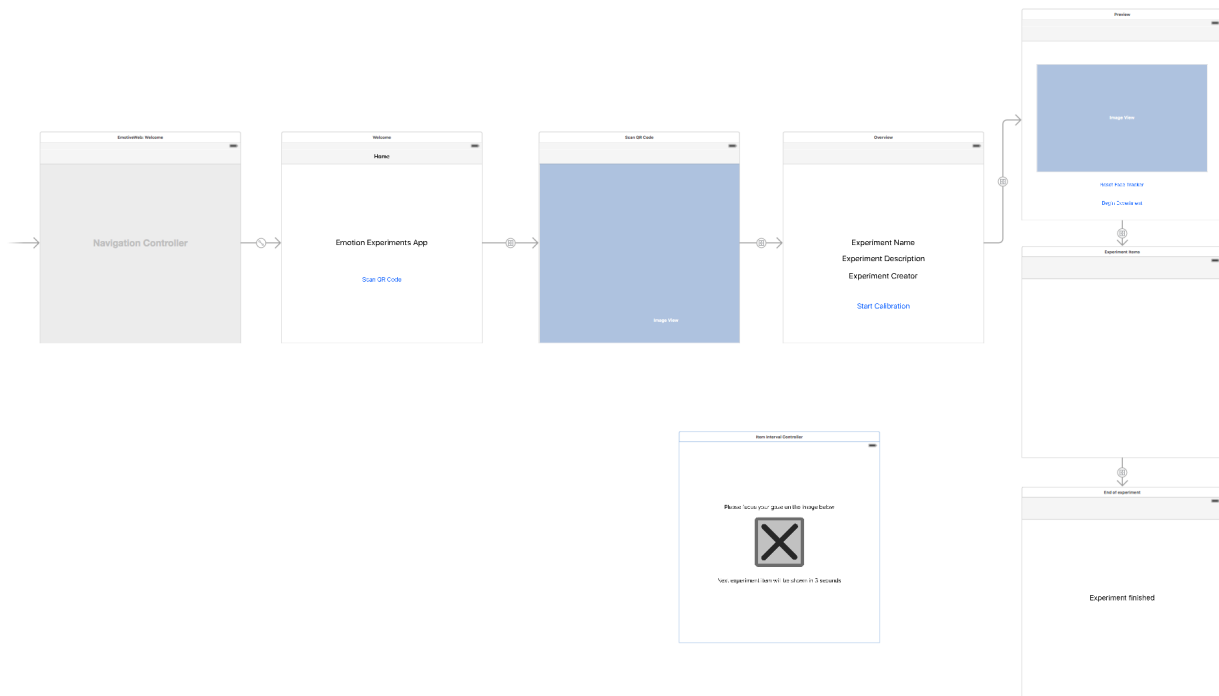


Figure 5.2: XCode Storyboard for the experiment iOS application

5.3.2 Obtaining experiment information

In order to start the experiment process, the QR code from the experiment creation tool needed to be read by the device to send an API request to the Experiment API. An *Experiment* Objective-C class was created to model all of the necessary fields and methods needed for the experiment. This is a singleton class, so once the information has been obtained through the API request embedded in the QR code, the instance with all of the experiment information can be accessed across the entire application.

The QR code reading functionality was developed using an *AVCaptureSession* object with the device's front camera as input and the built-in *AVCaptureMetadataOutput* class acting as output (Listing 5.8).

Listing 5.8: *AVCaptureMetadataOutputObjectsDelegate* delegate method used to capture QR Code data

```

1  -(void)captureOutput:(AVCaptureOutput *)captureOutput
    didOutputMetadataObjects:(NSArray *)metadataObjects fromConnection:(
    AVCaptureConnection *)connection{
2      if (metadataObjects != nil && [metadataObjects count] > 0) {

```

```

3      AVMetadataMachineReadableCodeObject *metadataObj = [
4          metadataObjects objectAtIndex:0];
5      if ([[metadataObj type] isEqualToString:
6          AVMetadataObjectTypeQRCode]) {
7          [_captureSession stopRunning];
8          qrcode = metadataObj.stringValue;
9          Experiment *exp = [Experiment getInstance];
10         [exp getExperimentInfo:@"http://" stringByAppendingString:
11             qrcode]];
12         dispatch_async(dispatch_get_main_queue(), ^{
13             [self performSegueWithIdentifier:@"QRCodeSegue" sender:
14                 self];
15         });
16     }
17 }
18 }
19 }

```

Lines 2-4 are used to detect if the machine readable code detected is a QR code. Line 5 then stops the *AVCaptureSession* once a code has been detected, and Line 6 obtains the string value embedded in the QR code. Lines 7 and 8 are then used to get an instance of the *Experiment* singleton class, and to retrieve the experiment information using a class method *getExperimentInfo* and the API request from the QR code. Finally, Lines 9-11 will perform a segue to the *ExperimentSummaryController* on the application's main thread.

5.3.3 Loading experiment items

The application has been designed to support 3 main types of experiment item:

1. Webpages
2. YouTube videos
3. Tweets

In addition to loading these items, an interval view has been developed, to be shown before each item. In order to display each item for the correct amount of time, the *displaySeconds* field has been used, and Dispatch Queues [49] have been used to schedule particular views after a certain amount of time.

Webpage items

To load webpage items, the iOS *UIWebView* class was used. This view could be preloaded with a URL, and would act like a browser, allowing a participant to navigate through the content displayed. This web view was loaded as a subview programmatically whenever required (Listing 5.9).

Listing 5.9: Loading a webpage as a subview

```
1 [self clearSubviews];
2 UIWebView *webview = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0,
    self.view.frame.size.width, self.view.frame.size.height)];
3 webview.scalesPageToFit = YES;
4 webview.autoresizesSubviews = YES;
5 webview.autoresizingMask=(UIViewAutoresizingFlexibleHeight |
    UIViewAutoresizingFlexibleWidth);
6 [webview setBackgroundColor:[UIColor clearColor]];
7 NSURL *targetURL = [NSURL URLWithString:data];
8 NSURLRequest *request = [NSURLRequest requestWithURL:targetURL];
9 [webview loadRequest:request];
10 [self.view addSubview:webview];
```

Initially, all existing subviews i.e. other items, are removed from the screen. A fullscreen UIWebView is then initialised and configured to scale pages, resize, and to have a clear background colour. The URL for the experiment item is then used to make a HTTP request to load data into the UIWebView. Finally, the web view is loaded onto the screen as a subview.

YouTube items

To load YouTube videos in the application, the YouTube IFrame Player API [43] was used. This API provided a YouTube player view that could be created and loaded with any video URL. This player view was loaded as a subview programmatically whenever required (Listing 5.10).

Listing 5.10: Loading a YouTube video as a subview

```
1 [self clearSubviews];
2 YTPlayerView *youtubeView = [[YTPlayerView alloc] initWithFrame:
    CGRectMake(0, 0, self.view.frame.size.width, self.view.frame.size.
    height)];
3 [self.view addSubview:youtubeView];
4 [youtubeView loadWithVideoId:data];
```

Like with the webpage items, initially all existing subviews are removed. A fullscreen YouTube player is then created using the YouTube IFrame Player API, before being added to the screen and loaded with the experiment item's video ID.

Tweet items

To load Tweet items, the Fabric library [44] was used. This library provided a single Tweet view that could be created and loaded with any Tweet ID. This Tweet view was loaded as a subview programmatically whenever required (Listing 5.11).

Listing 5.11: Loading a single Tweet as a subview

```

1 [self clearSubViews];
2 NSString* tweetID = [self getTweetIDFromURL:data];
3 TWTRAPIClient *client = [[TWTRAPIClient alloc] init];
4 [client loadTweetWithID:tweetID completion:^(TWTRTweet *tweet, NSError *
    error) {
5     if (tweet) {
6         TWTRTweetView *tweetView = [[TWTRTweetView alloc]
            initWithTweet:tweet];
7         [self.view addSubview:tweetView];
8         tweetView.center = [self.view convertPoint:self.view.
            center fromView:self.view.superview];
9     } else {
10        NSLog(@"Error loading Tweet: %@", [error
            localizedDescription]);
11    }
12 }];

```

Like with the other types of items, initially all existing subviews are removed. A utility method *getTweetIDFromURL* is then used to retrieve the Tweet ID from the full Tweet URL. A Twitter API client object is then initialised, and the client is used to retrieve the Tweet using the retrieved Tweet ID. After this, a Tweet view is created using the Tweet data from the client, and it is loaded as a subview onto the centre of the screen.

Experiment item intervals

The experiment item interval view was developed after a meeting with my Supervisor and PhD students in the School. They highlighted the importance in giving a participant time to prepare for each experiment item, so that the results would not be skewed by the previous item viewed. To achieve this, a basic view was designed to be shown for 3 seconds before each experiment item (Fig. 5.3).

This was not only used to give participants a short break between viewing items, but also as a means of trying to normalise the recorded videos, through prompting participants to refocus their gaze on a particular part of the screen.

This view was loaded in a similar way as the other items in an experiment using Dispatch Queues, and adding it as a subview.

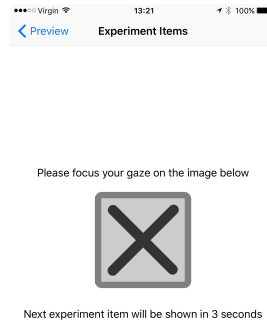


Figure 5.3: Experiment interval screen displayed before each experiment item

5.3.4 Face tracking

The majority of the code to interface with and use the FaceTracker library in the project has previously been developed during a CUROP project, and previous face tracking work in the School [8]. However, this project required slightly different functionality for the face tracker, so some changes have been to the main Objective C++ wrapper.

To use the FaceTracker library in this project, all of the C++ files were imported as they were referenced by the wrapper. The ASM model files used by the library were also imported and referenced by the wrapper.

Both the TrackingPreviewViewController and the ExperimentItemViewController use an *AVCaptureSession* to capture frames from the front camera, and to save them into a buffer for processing. A delegate method is then used to call the main FaceTracker wrapper method from these controllers. This method will retrieve an image from the image buffer, perform some basic image operations, before passing the image to an appropriate tracking method dependent on the controller that made a call to the wrapper.

If the call was made by the TrackingPreviewController, then the tracker will be applied to the image, and the fitted model will be drawn onto the image, and shown on the preview view (Listing 5.12).

Listing 5.12: Tracking functionality used by TrackingPreviewViewController

```

1  if(model.Track(gray,wSize,fpd,nIter,clamp,fTol,fcheck) == 0) {
2      [self draw];
3      failed = false;
4  }else{

```

```

5         [self resetModel];
6         failed = true;
7     }

```

If the tracker is successful in fitting the model, the *draw* method is called that uses the point and line shapes from the OpenCV library to overlay the fitted model onto the image frame. If the tracker is unsuccessful, the model is reset, and the ASM process will start again when the next frame is tracked.

If the call was made by the *ExperimentItemViewController*, then the tracker will be applied to the image, and the resulting landmarks for emotion classification (Listing 5.13).

Listing 5.13: Tracking functionality used by *ExperimentItemViewController*

```

1  if(model.Track(gray,wSize,fpd,nIter,clamp,fTol,fcheck) == 0) {
2      failed = false
3      vect2test(model._shape, test);
4      pca_project(test, eigv, mu, sigma, eigsize, feat);
5      scaledValues = [svm scaleData:trainRangePathString test:feat];
6      predictedValues = [svm predictData:scaledValues];
7      [self outputData];
8  }else{
9      [self resetModel];
10     failed = true;
11 }

```

If the tracker is successful, the facial landmark points are extracted from the fitted model, and converted into a set of distance measures, using the *vect2test* method within the face tracker wrapper. These distance measures are the same normalised measures used during emotion classifier training.

Data from the PCA step of the emotion classifier training process is then used to perform PCA on these test features. The *scaleData* and *predictData*, perform the emotion classification (Page 52).

The output of each run of this tracker code is a set of emotion measurements that is appended to a results array, and sent to the Results API after each experiment item, with the other summary information.

5.3.5 Emotion classification

The emotion classification functionality was achieved through using an SVM wrapper developed during the CUROP project that combined the *svm-scale*, and *svm-predict* LIBSVM functions into a single file. The usage of the methods in this project has been highlighted in the Face

tracking section (Page 52).

Before any classification can take place, the SVM model has to be loaded into the application. This involves providing the path to the *emotions.train.pca.model* file produced during training, and using a *loadModel* method from the LIBSVM library.

In the tracking method, the *scaleData* is called on the PCA test data. The code in this method is very similar to the LIBSVM *svm-scale* code. It will scale this data to the range file produced during emotion classifier training, *emotions.train.pca.range*.

Once the test features have been scaled, the *predictData* is called on the scaled test data. This method is very similar to the LIBSVM *svm-predict* code. It uses the *svm_predict_probability* method from LIBSVM to return probability estimates for each emotion.

5.3.6 Recording experiment participants

One of the user requirements for this application was to record and save videos of participants viewing experiment items to the device. This was a challenging requirement to meet as the application was already completing tasks such as face tracking, emotion classification, and presenting different types of web-based content.

Each frame that is processed by the application is available as an Objective-C UIImage. After some research, the CEMovieMaker library [42] was found, that was able to convert an array of these UIImages into a single .mov file. The only modification that had to be made to the library was the frame rate used when generating the video file.

When this library was initially integrated into the application, the application was able to handle the extra memory requirements for short experiment items. However, for experiment items that exceeded ~30 seconds, the application would crash with a memory error. After some investigation, it appeared that the for longer items, more UIImages would have accumulated in memory, but none of the memory could be released until the video had been generated at the end of the experiment item.

To fix this issue, a solution was found where instead of waiting until the end of an experiment item to generate a video, a video would be generated every 300 result frames, so that the memory requirements never exceeded 300 frames (Listing 5.14).

Listing 5.14: Adding camera frames to an UIImage array and saving to a video file at regular intervals

```
1 | -(void)addVideoFrame:(UIImage *)frame{
```

```

2     [videoFrames addObject:frame];
3     if ([videoFrames count] == 300){
4         [self saveVideoFrames];
5     }
6 }
7 -(void) saveVideoFrames{
8     NSDictionary *settings = [CEMovieMaker videoSettingsWithCodec:
9         AVVideoCodecH264 withWidth:320 andHeight:480];
10    NSString* videoFileName = [self generateVideoName];
11    self.movieMaker = [[CEMovieMaker alloc] initWithSettings:settings
12        videoName:videoFileName];
13    [self.movieMaker createMovieFromImages:[videoFrames copy]
14        withCompletion:^(NSURL *fileURL){
15        NSLog(@"%@", fileURL);
16    }];
17    [videoFrames removeAllObjects];
18 }

```

The *addVideoFrame* method is called every time a camera frame has been tracked and classified. Line 2 adds the frame to the *videoFrames* array. Lines 3-5 check if there are 300 frames in the array, and will call the *saveVideoFrames* method accordingly. Line 8 specifies the *CEMovieMaker* library settings including the generated video's codec, and the height/width of the video, taken from the frame size. Line 9 will call a utility method that has been developed to generate a video file name such as *57025c35149275d63527b06f-570abbe2149275d63527b07a-1460453142.mov* where the file name is made up of the experiment ID, item ID, and the current UNIX time. Lines 11-13 will execute the *createMoviesFromImages* method from the *CEMovieMaker* library. Finally, Line 14 will remove all of the frames from the array to release some of the application's memory.

This solution was tested with experiment items up to 5 minutes in length, and the application was able to save these videos at more regular intervals and not crash (Fig. 5.4).

As seen in the Instruments plot, there is a pattern where the memory will accumulate for a short while before being released due to the frame data being saved and released every 300 frames. The videos can then be merged together once they have been obtained from the device. A basic script was written in Python to achieve this for the experiment phase of the project.

To make it easier for users to obtain these videos from the device after the experiment, a small configuration change was made to the application to support iTunes File Sharing. This involved enabling the *UIFileSharingEnabled* key within the application's *info.plist* file.

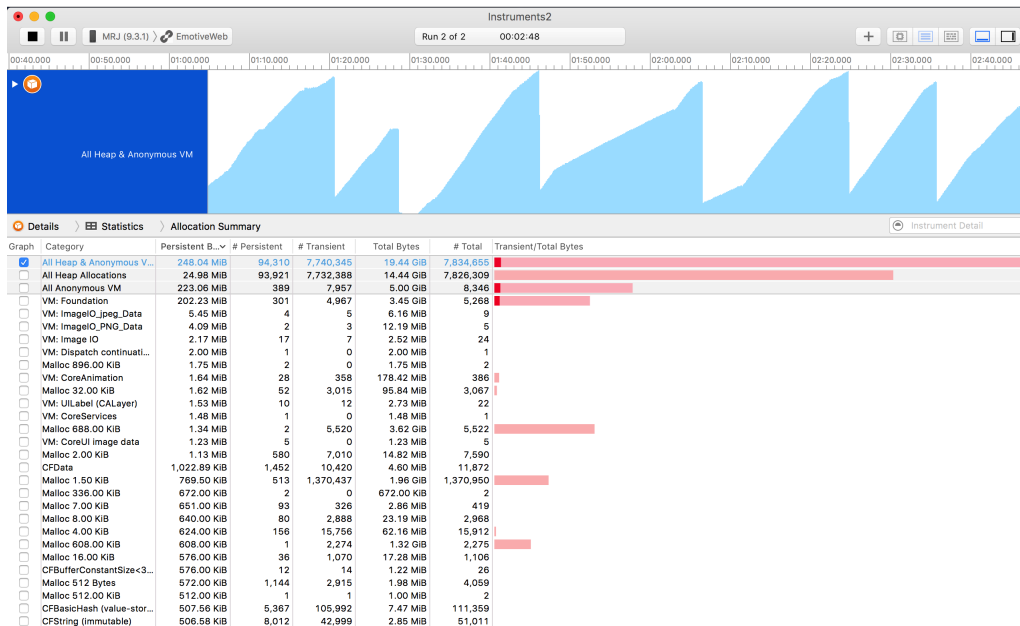


Figure 5.4: Memory allocations instrument recordings from iOS Instruments development tool whilst viewing experiment items

5.4 Emotion experiment results tool

The emotion experiment results tool uses a very similar implementation to the emotion experiment creation tool. The work described in the MEAN stack setup for the emotion experiment creation tool (Page 42), and the majority of the API work is the same. Therefore, only the major differences between the 2 implementations will be discussed in this section.

5.4.1 Result API

As with the Experiment API, the Mongoose API tool was used to model a Result object for the API (Listing 5.15).

Listing 5.15: Result Mongoose model

```

1 var mongoose = require('mongoose');
2
3 var ResultSchema = new mongoose.Schema({
4   experimentID: {type: String, required: true},
5   experimentName: {type: String, required: true},
6   itemData: [{
7     itemID: {type: String, required: true},
8     data: {type: String, required: true},
9     dataType: {type: String, enum: ['twitter', 'youtube', 'webpage'],
10      required: true},
11     displaySeconds: {type: Number, required: true}

```

```

11  }],
12  resultData: [{
13      emotionData: [{
14          angry: Number,
15          contempt: Number,
16          disgust: Number,
17          fear: Number,
18          happy: Number,
19          sadness: Number,
20          surprise: Number,
21          neutral: Number
22      }],
23      trackingData: [{
24          data: Array
25      }]
26  }],
27  updated_at: { type: Date, default: Date.now },
28  });
29
30 module.exports = mongoose.model('Result', ResultSchema);

```

Some of the fields in this schema are from the Experiment schema - *name*, *itemData*, and the generated *experimentID*. These fields are used to make it easier to reference experiment information in other components of the system. The main field in this API is the *resultData* array. Each element of this array represents the results for a single frame, and it is made up of 2 fields - *emotionData* and *trackingData*. The *emotionData* contains 8 key-value pairs for the emotion measurements. The *trackingData* field is an optional field that allows all of the facial landmark data to be available through the Result API. Finally, the *updated_at* field keeps a record of when the result was last modified.

This API used the same CRUD model to form the API routes as the Experiments API. The only difference is that the PUT route has not been implemented for the Results API, as the system does not support editing results. The code used to implement the Experiments API routes was re-used, with just the schema being changed to the Results Mongoose schema.

5.4.2 Experiment results web application

After developing all of the API routes, a basic web application with 2 main views was created using AngularJS.

'Home' View

This view had similar functionality as the 'Home' view in the experiments creation tool, to list a summary of all information available from the Result API. As with the experiment creation tool,

a results service was created to handle all of the Result API routes. This service only needed the default API routes that \$resource handles as there was no need to have a PUT route for this API.

A controller was created to use this service, and to store all result data. The following information was listed for each result:

- Result ID
- Item ID
- Item Source
- Number of processed frames (length of *resultData* array)

All of these results were grouped by experiment in AngularJS tables, and the tables were sorted by the experiment item ID field. For each item result, buttons were added to allow a user to view/delete a result.

'Result' View

This view was created to show a basic visualisation of a particular item's result. To achieve this, an existing AngularJS charting library was used, Angular-nvD3 [45]. The library is based heavily on D3.js, and allowed a list of options to be set up once in a controller, and to be reused by all results. The only additional work that was required was developing a small function capable of converting the emotion data from the API into a suitable format for this charting library (Listing 5.16).

Listing 5.16: JavaScript function to convert emotion data into a format suitable for charting

```
1 function emotionChartData(emotionData, emotion){
2     var emotions = [];
3     for(var i = 0; i < emotionData.length; i++) {
4         emotions.push({x: i, y: emotionData[i][emotion]});
5     }
6     return emotions;
7 }
8 }
```

This script had to be run for each of the 8 emotions available in the data. After converting this data, each series was appropriately labelled by their emotion, and plotted onto a Angular-nvD3 line chart.

5.5 Emotion classifier

The emotion classifier is a key underlying component of the system, that predicts the confidence level of different emotions being expressed using facial landmark data. In order to build the classifier, several steps needed to be carried out:

1. Extract facial landmark data from images
2. Categorise images by their class label
3. Calculate distance measures based on facial landmark points, and perform normalisation on
4. Project distance measure vectors and perform Principal Component Analysis
5. Use LIBSVM to train a classifier based on the PCA data

5.5.1 Obtaining features

Video frame extraction

Before being able to extract features, some of the frames had to be extracted from one of them emotion databases, GEMEP-FERA. To do this, a Python script was created that used OpenCV to extract the key frames and to save them as a separate training images (Listing 5.17).

Listing 5.17: Extracting frames from GEMEP-FERA video sequences

```
1 for subfolder in os.listdir('Datasets/GEMEP-FERA'):  
2     for vid in glob.glob('Datasets/GEMEP-FERA/' + subfolder + '/*.avi'):  
3         vidcap = cv2.VideoCapture(vid)  
4         count = 0;  
5         while success:  
6             success, image = vidcap.read()  
7             cv2.imwrite("%d.jpg" % count, image)  
8             count += 1
```

Lines 1 and 2 iterate through the GEMEP-FERA folder structure to find each video file name. Line 3 uses the file name to instantiate an OpenCV VideoCapture object with the video file. Lines 4-8 then use the VideoCapture object to read the video frame by frame, to save each frame as a .jpg image file.

Applying FaceTracker to training data

A python script *facetrack.py* was created to take images/video frames from emotion databases and to produce vectors of facial landmark points. These vectors were then categorised into

folders using their class label i.e. emotion. This was a basic script that used the FaceTracker library to obtain the points, and to store them in a .vector file. It then used knowledge about how the particular emotion database stored emotion information to put the vector file into an appropriate emotion folder. Listing 5.18 shows how this script was applied to the CK+ database [21].

Listing 5.18: Obtaining facial landmark data from CK+ database

```

1 for episode in os.listdir('Datasets/CK+/Emotion/' + sequence):
2     for index in glob.glob("Datasets/CK+/Emotion/" + sequence + "/"
        + episode + "/*.txt"):
3         index = os.path.basename(index)
4         index = index[:index.index("_emotion")]
5         for i, index_content in enumerate(open("Datasets/CK+/
            Emotion/" + sequence + "/" + episode + "/" + index +
            "_emotion.txt")):
6             emotion = emotions[int(float(index_content.strip(
                )))])
7             if not emotion in data: data[emotion] = []
8             os.system("cp Datasets/CK+/cohn-kanade-images/"
                + sequence + "/" + episode + "/" + index + ".
                png; ./face_tracker " + index + ".png; mv
                " + index + "* Datasets/CK+/cohn-kanade-
                images/" + sequence + "/" + episode + "/")
9             data[emotion].append("Datasets/CK+/cohn-kanade-
                images/" + sequence + "/" + episode + "/" +
                index + ".vector")

```

Lines 1 and 2 are used to iterate through the CK+ folder structure, to obtain the emotion of all of the training images from the 'Emotion' folder that stores the emotion of each image in a text file. Lines 3-7 are used to obtain the name of the referenced training image from the text file, and to obtain the emotion being expressed in the image. All of the facial landmarks for all of the training images are stored in a *data* array. Line 8 is used to run the FaceTracker library on a training image, and to move the resultant vector file into an appropriate folder. Line 9 then appends the location of the vector to the correct emotion in the *data* array, to move all of. There were only a small number of problems when developing this functionality relating to incorrect file paths being used in the script. The CK+ database was the most difficult to obtain the class from in the script as it was stored in a separate text file. For all of the other databases, the emotion information was part of the file name.

Once vector files of facial landmarks had been obtained for all training images, information in the *data* array was then used to move the vector files into different folders for each emotion.

Calculating distance measures

The emotion classifier used 86 different distance measures as its features, as described in the Support Vector Machines section (Page 19). A python script *calculatefeatures.py* was developed to produce an *emotions.train* text file that would contain all of these measures for each vector, categorised by the emotions of the vectors. The script itself was fairly basic, and involved iterating through different combinations of the facial landmarks to produce the distance measures. All distance measures were then normalised by the distance between the eyes.

5.5.2 Principal Component Analysis

A Python script, *pca.py*, was developed to perform Principal Component Analysis on the obtained feature vectors. The script (Listing 5.19) relied heavily on the PCA module within the matplotlib library [47].

Listing 5.19: Performing PCA on obtained feature vectors

```
1 results = PCA(np.array(data))
2 archive = open("pca_archive_wt.txt", "w")
3 for v in results.Wt: archive.write(",".join([str(float(x)) for x in v])
4     + "\n")
5 archive.close()
6 archive = open("pca_archive_mu.txt", "w")
7 archive.write(",".join([str(float(x)) for x in results.mu]) + "\n")
8 archive.close()
9 archive = open("pca_archive_sigma.txt", "w")
10 archive.write(",".join([str(float(x)) for x in results.sigma]) + "\n")
11 archive.close()
12 fout = open("emotions.train.pca", "w")
13 for line in open("emotions.train"):
14     temp = []
15     for el in line[2:].strip().split("_"):
16         temp.append(float(el[el.index(":")+1:]))
17     fout.write(line[:2] + "_" + ".join([str(str(i+1) + ":" + str(index))
18         for i, index in enumerate(results.project(np.array(temp),
19             0.001))]) + "\n")
20 fout.close()
```

Line 1 uses the PCA class within matplotlib to project the data onto a reduced set of dimensions. The input given to this class is all of the vector data as an array represented as number of observations (number of distance measures) X number of dimensions (number of vectors). Once this has been done, a number of class variables from the *results* object are extracted to avoid having to reproduce the PCA later. Lines 2-4 extracts wt, the weight vector for projecting a vector into PCA space, and saves it to a text file. Lines 5-7 extract mu, an array

of means for the vector data, and saves it to a text file. Lines 8-10 extract sigma, an array of standard deviations for the vector data, and saves it to a text file. Lines 11-17 then will extract the principal components (reduced set of the vectors), for SVM training, into a file *emotions.train.pca*.

5.5.3 SVM training

A python script, *svm.py*, was created to scale training data, perform training, and cross-validation, through the LIBSVM library. The majority of the script involves running commands on the LIBSVM binaries.

Firstly, the training data was scaled using the *svm-scale* binary (Listing 5.20).

Listing 5.20: Scaling training data using svm-scale

```
1 ./svm-scale -s "emotions.train.pca.range" "emotions.train.pca" > "
  emotions.train.pca.scale"
```

This command uses scaling parameters, *emotions.train.pca.range* and the training data *emotions.train.pca* from *pca.py*. The *-s* command specifies that training data is being scaled as opposed to scaling test data. The output of this command is then saved to a *emotions.train.pca.scale* range file.

After scaling, the best parameters for the SVM kernel were found through cross-validation using the *grid.py* file from the LIBSVM library (Listing 5.21).

Listing 5.21: Obtaining SVM parameters through cross-validation

```
1 python grid.py -svmtrain "./svm-train" -gnuplot "/usr/local/Cellar/
  gnuplot/5.0.2/bin/gnuplot" "emotions.train.pca.scale"
```

This command performs 5-fold cross validation on the scaled training data, *emotions.train.pca.scale*, to find the best *c* and Gamma SVM parameters. The flag *-gnuplot "/usr/local/Cellar/gnuplot/5.0.2/bin/gnuplot"* is used to produce cross-validation plots during this process.

The final parameters that were chosen were a value of 8 for the *C* parameter, and a value of 8 for the Gamma parameter. The *C* parameter indicates the cost of classification where a large *C* gives low bias and high variance, and a low *C* gives higher bias and lower variance. The Gamma parameter is the parameter for the Gaussian radial basis function kernel used by the SVM. The final cross-validate rate achieved by the classifier was 90.902% (Fig. 5.5). After obtaining the best values of these parameters, the *svm-train* binary was used to train the classifier (Listing 5.22).

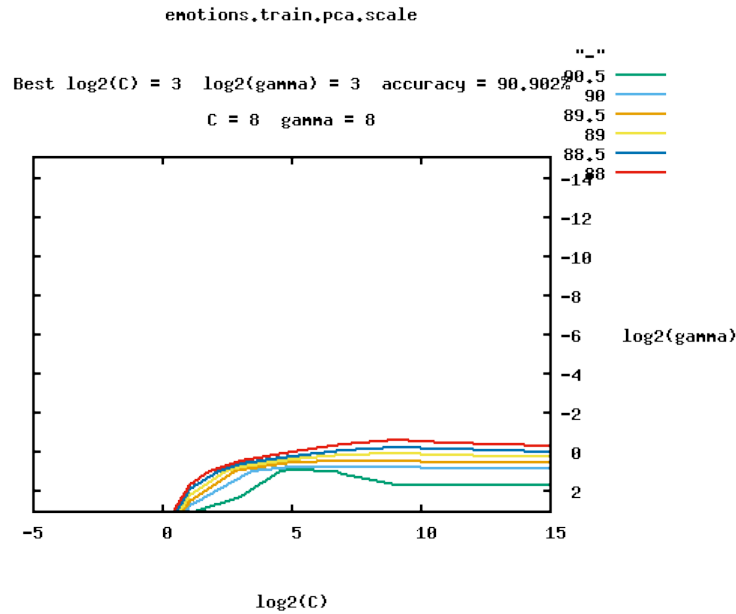


Figure 5.5: Cross-validation plot for the emotion classifier used in experiments

Listing 5.22: Training the emotion classifier using svm-train

```
1 ./svm-train -b 1 -c 8 -g 8 "emotions.train.pca.scale" "emotions.train.pca.model"
```

This command runs the svm-train binary with the *c* and *Gamma* parameters of 8. The *-b* boolean flag is used to specify that the model should be trained to provide probability estimates. These estimates are what the emotion measurements will be based on.

This process produces the *emotions.train.pca.model* SVM model and the *emotions.train.pca.range* range file that are used by the iOS application to load the trained classifier to make emotion predictions during experiments.

Chapter 6

Results and Evaluation

This chapter documents the main work that has been completed to evaluate the developed system. As stated in the project approach (Page 12), an experiment was planned to validate the system. The design, execution and analysis of this experiment is discussed here. Subsequent work on the emotion classifier after analysis of the validation experiment results is also outlined. Finally, the system is evaluated against the set of user requirements specified during the design of the system (Page 31).

6.1 Experimentation

6.1.1 Experiment design

For the validation experiment, all of the different types of content the system supports (Webpages, YouTube videos, Tweets) were used as experiment items. Table 6.1 provides an overview of the 8 items chosen for the experiment. With only a limited time to design the experiment, it was difficult to find items to evoke emotions such as *Anger* or *Sadness*, without potentially offending some of the experiment participants. Using items that produced these kinds of emotions would also raise ethical concerns. Whilst the designed experiment does focus more on the emotions *Happy*, *Natural*, and *Surprise*, items such as the BBC news feed have the potential of producing other responses.

Source	Content Type	Description	Display time
http://www.bbc.co.uk/news	Webpage	BBC News homepage	30s
https://twitter.com/search?q=%23DadJokes&src=tyah	Webpage	Twitter feed of tweets tagged with 'DadJokes'	30s
https://www.youtube.com/watch?v=73hyDWqsy8E	Video	Fastest talking Guinness World Record attempt video	70s
https://www.youtube.com/watch?v=7-xnwrWM1z4	Video	News report blooper video	15s
https://twitter.com/emergencypuppy?lang=en-gb	Webpage	Twitter page for 'Emergency Cute Stuff' account	30s
https://twitter.com/YouveBeenFramed/status/716316537271914497	Tweet	'You've Been Framed' tweet containing GIF	10s
https://www.youtube.com/watch?v=1JT-yX9C0y0	Video	Mercedes-Benz advert featuring dancing chickens	25s
http://www.huffingtonpost.com/good-news/	Webpage	Good News section of Huffington Post website	30s

Table 6.1: Details of experiment content used in the validation experiment

The webpages used in the experiment were chosen on the basic criteria of having dynamic content such as headlines from a news website or a feed from a social media website. Although

participants may be familiar with the sites chosen, as the content regularly changes on these websites, the majority of the content should be unfamiliar. Each website can be interacted with as you would normally in a browser, and a user had 30 seconds to interact with each site.

The videos in the experiment were chosen using some basic criteria. The videos had to be short in length and have a small number of key events that would produce an emotional response that could be cross-referenced against the emotion data with. It was also important that the videos chosen were relatively unknown and not 'viral' as participants should ideally be viewing the videos for the first time. To help to source videos that met this criteria, 2 of the videos were taken from the EmoVu web-based demo [27]. The third video in the experiment was sourced through looking at user made playlist on YouTube. To try to make the videos even more unfamiliar to participants, the videos were downloaded from YouTube, and re-uploaded as private videos, only accessible through the experiment. By re-uploading the videos, metadata such as the original video title could be removed so that a participant would have less information before viewing the video.

The tweet used in the experiment was chosen fairly randomly to produce a Happy or Surprise response. In addition to the tweet text, there is a GIF attached from the popular TV show 'You've Been Framed'. Although it was tweeted from a fairly popular account, the volume of tweets even from this account means that it is very unlikely to have been seen by a participant.

After discussions with PhD Students in the School about the design of my experiment, it was suggested to also include a feedback form at the end of the experiment. A basic paper-based general comments form was designed for participants to fill in. This gave them the opportunity to provide information on what they thought about the items shown, their responses, the design of the experiment iOS app, and any general comments about the experiment process.

6.1.2 Experiment execution

During a single week in April, 16 participants carried out the designed experiment. The set-up for each experiment was kept as consistent as possible through using the same library room, lighting conditions, mobile device, and mount to hold the device. By mounting the phone, it helped to stabilise the frames that were captured, and it ensured that the participant would be looking at the phone directly, and not at an angle if they had held the phone themselves.

For each participant, frame-level emotion measurements were recorded. Videos of the participant viewing each item were also captured to help to validate and analyse the results. All participants were made aware of how their data would be used in this project. All of this data, as well as some summary data is available in the *ResultData.zip* file attached with this report.

The process that was followed for each experiment was:

1. Provide an overview of the project, what the experiment involves, and how the experiment data will be used
2. Calibrate the phone mount to the participant using the face tracker preview available in the iOS application
3. Allow the participant to view and interact with the experiment items
4. Ask the user for some general comments about the experiment
5. Retrieve the video data from the phone
6. Retrieve the emotion measurements using the Results API

There were a small number of occasions where the experiment had to be repeated when a user accidentally closed the application. Generally, this experiment process lasted only 15 minutes.

6.1.3 Experiment analysis

Emotion data analysis

After executing the experiment, the experiment data for each participant was collated, and a short Python script was written to produce average emotion measurements for each item. This summary data is available in the *Summary Data* folder in the *ResultsData.zip* file. Each experiment item's averages was plotted in MATLAB during analysis, and these plots are also available in the *Summary Data* folder. A subset of these plots will be used as part of this analysis.

After analysing all of the experiment data, generally it appears that the emotion measurements appear to be significantly biased towards particular emotions compared to others. As a result, some of the emotion measurements that were expected when designing this experiment have not been seen during this analysis. To rectify this, some further emotion classifier work has taken place after the experiment (Page 70).

Webpage items There were 4 webpages used in the experiment in total. Of these 4 webpages, 2 of them were feeds from news websites, and 2 of them were feeds from Twitter.

When viewing the results of participants viewing the BBC News homepage, it appears that there was no significant emotional response measured (Fig. 6.1).

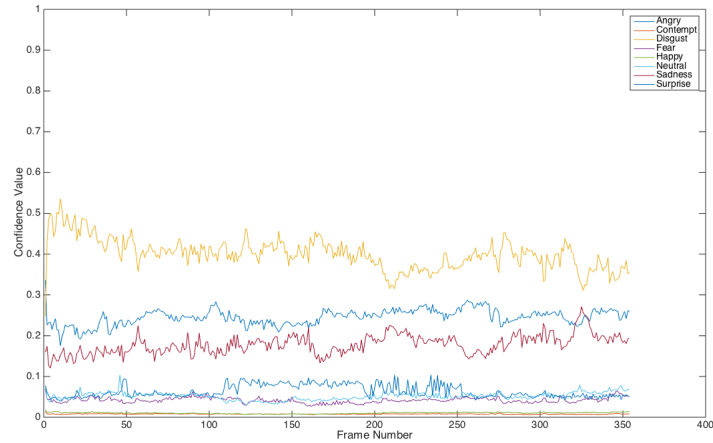


Figure 6.1: Average emotion measurements for BBC News experiment item

The highest measurements were from the *Angry* and *Disgust* emotions. Whilst these are likely responses when viewing news websites, it appears these measurements are more of a result of problems with the emotion classifier compared to the true response of participants. Apart from these 2 emotions, the only other emotion that has a measurement higher than *Angry* or *Disgust* at any point in the experiment is *Sadness*. Although the measurement is fairly low, it is still significantly higher than any of the other 5 emotions in the experiment, and is not attributed to an issue with the emotion classifier. When reviewing the raw videos recorded of participants viewing the BBC News item, and comparing it to the videos of participants viewing other items, this item produces the least amount of emotional response. This could be for several reasons such as already being familiar with the news headlines shown, the order of the item in the experiment, or the lack of media other than text in the item.

When viewing the results of participants viewing a feed of Tweets tagged with 'DadJokes', a pattern similar to the BBC News item emerged in the results (Fig. 6.2).

This item was designed to produce responses of *Happy*. Whilst there are a small number of peaks for this emotion, particularly between frames 100 and 130, there are still other emotions that have significantly higher measurements - *Angry*, and *Disgust*. This was one of the main results used to discover problems with the emotion classifier. The positive of this item's results is that there was a significantly more evident emotional response from participants in the raw videos when viewing this item compared with the BBC news item. When extending that comparison to the participant videos available of both news websites, and the videos available of both social media feeds, it appears that participants generally responded more to social media feeds over websites.

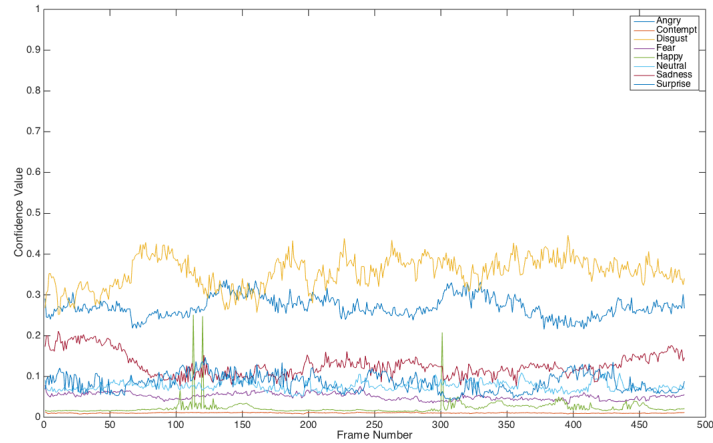
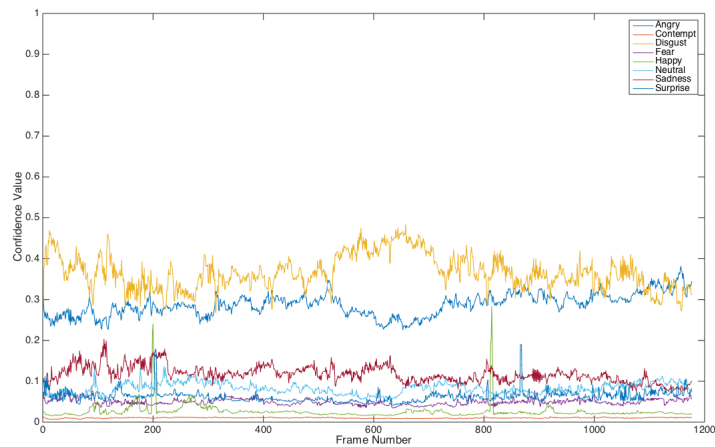


Figure 6.2: Average emotion measurements for 'DadJokes' Twitter feed experiment item

YouTube items There were 3 YouTube videos used in the experiment. All 3 were chosen to produce the responses of *Happy* or *Surprise*. After reviewing the emotion data, the results of participants viewing these items are marginally better than the webpage item results. For example, in the results for the Guinness World Record video, there are a small number of peaks for *Surprise* and *Happy* (Fig. 6.3).



A

Figure 6.3: Average emotion measurements for Guinness World Record YouTube video experiment item

When looking at the original video, these peaks appear to be around the points when certain key events happen such as the contestant starts the record attempt (\sim Frame 200), and when she is close to completing the record (\sim Frame 800). However, there are still problems with the *Angry* and *Disgust* emotions still having high measurements. More time was spent

reviewing the raw videos of participants viewing the YouTube videos to see if any of these high measurements could be explained. However, there is clear evidence that the intended responses of *Happy* and *Surprise* are achieved, to the extent that this data could be manually relabelled and used as examples of these emotions in a future classifier training process.

Tweet items The only Tweet used in the experiment was from a popular UK TV show - "You've Been Framed". The Tweet was designed to test the system's ability to detect the emotions *Happy* and *Surprise*. Figure 6.4 shows a plot of the average emotion measurements whilst viewing this tweet across all participants.

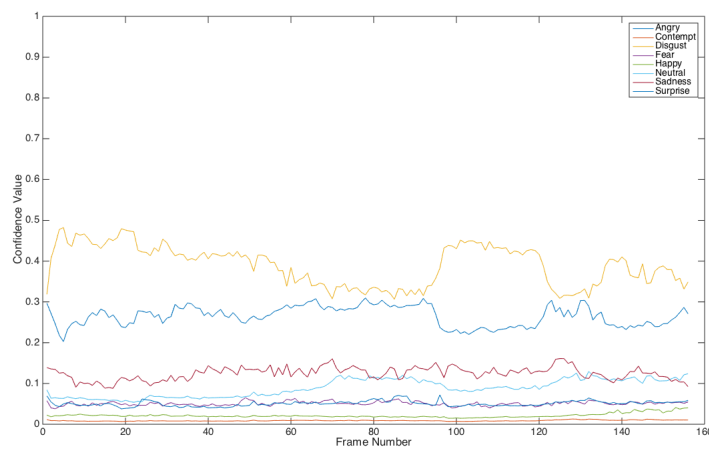


Figure 6.4: Average emotion measurements for Tweet experiment item

As seen in this plot, the confidence values for these emotions are significantly lower than those for *Angry* or *Disgust*. The highest recorded value for *Happy* across all participants whilst viewing this item was ~ 0.65 , and the highest recorded value for *Surprise* was ~ 0.61 . After reviewing the raw video data of participants viewing this Tweet, it appears that the majority of the participants did not respond with the intended response of *Happy* or *Surprise*. This indicates that there may be an issue with the specific tweet that was chosen for the experiment. With the item only having a small amount of content and being shown for 10 seconds, it was not always guaranteed to achieve the intended response. To improve the experiment, a variety of Tweets could have been used to see if it would be possible to get a significant emotional response from a single Tweet. Whilst there is some video data that suggests the results for *Happy* and *Surprise* should be higher than they were, these measurements seem to be lower than expected due to both the classifier's limitations and the particular Tweet chosen for this experiment.

Feedback analysis

In addition to the numerical results, after each experiment, participants were asked to provide a general comment about the experiment. The survey asked them to think about the content they viewed, their responses, the design of the iOS application, and the guidance given throughout the experiment. All of this feedback was then collated into a *Result info.xlsx* spreadsheet, available in the *ResultsData.zip* file. After reviewing these comments, there were a small number of points that were raised consistently across the participants, that provide more understanding of the strengths and weaknesses of the experiment.

Firstly, participants commented that the mount used to hold the phone throughout the experiment wasn't particularly sturdy and that the device seemed to shake when interacting with the application. Whilst all of the participants were able to interact with experiment items, some resorted to holding the device throughout the experiment. This may have had an effect on the emotion measurements as 2 of the participants said they were often more focused on keeping the device still compared to viewing the experiment items. The choice to use a mount instead of asking participants to hold the device was to control the conditions more, particularly their pose, and to keep all of the experiment participant video data as consistent as possible. If future experiments were to take place, then the setup used may need to be considered more.

Secondly, it was noted by participants that generally they hadn't seen any of the experiment items before. The majority of the participants were familiar with the BBC News website and Twitter, but as the content on these websites changes so often, the articles/tweets on these sites were still new to them. This is a positive for the designed experiment, as the content was designed to be viewed for the first time. Only 1 participant wrote that they had seen one of the YouTube videos before. If more experiments were to be carried out, it would be important to continue to source items that participants are unlikely to have seen.

One piece of feedback that came from a small number of participants was that the face tracker did seem not cope with features such as beards and glasses very well, when they were shown the face tracker preview screen on the iOS application. This is a general problem that occurs in face tracking and this was mitigated to an extent by participants removing their glasses where possible. However when viewing the data of participants with such features, there seem to be more anomalies in the data compared to participants without these features. That suggests that face tracker model may have incorrectly deformed to faces at times, and affected the emotion measurements. In a future experiment, this could be further mitigated by having a larger sample of participants, and through considering other face tracking libraries or further training of the current library to be more robust to these features.

A point that was raised by multiple participants was that there was no content in the experiment that produce the responses Anger, Fear, or Sadness, despite them being listed on the project description they were provided with. As noted in the experiment design, it was difficult to find content that would produce these emotions without offending some of the participants. When researching content to show in the experiment, there were a small number of videos that had the potential to produce these responses, but it was decided against for this validation experiment, due to ethical concerns. If the same experiment was completed again with participants, it would be worth making it clearer in the information given to participants that the experiment was focused on a subset of the emotions, as some people were expecting content that would produce anger, sadness and fear. If other experiments was carried out using this tool in the future, then more time could be spent considering content for these emotions.

Finally, there were some mixed comments about the types of content used in the experiment, and how long each item was shown for. The general consensus after reviewing feedback and looking at the raw experiment video data is that participants responded significantly more to the YouTube videos compared to any other type of content. More than one participant wrote in their feedback that the news websites shown didn't produce an obvious emotional response, and they would have preferred more video items. There are many possible reasons why these news websites didn't produce a response with participants. The content viewed on these sites is very time-specific, and the news stories shown on the sites may be irrelevant or already known by the participants. Also, the websites used were feeds of news headlines, with little media to view other than text.

6.2 Further emotion classifier work

As noted in the experiment analysis (Page 64), some of the emotion measurements that the emotion classifier produced during experiments were not expected, especially when compared with the raw experiment video data and knowledge about the content a participant was viewing. After meeting with my supervisor, and reviewing both the experiment results and the process used to build the emotion classifier, some potential issues were raised.

To try and address some of the issues in the remaining time of the project, 3 pieces of investigative work were identified and carried out:

1. Build a multi-class classifier similar to the one used in the experiment but with more evenly distributed data
2. Build a multi-class classifier for each of the emotion databases used in the original training process

3. Build multi-class classifiers with one or more of the emotion databases removed from the training process

Whilst it was feasible to build these classifiers through modifying the Python scripts produced for the original training process, it was assumed that due to time-constraints, there would be no time to conduct another experiment with participants. Instead, to see if these pieces of work had any effect on the emotion measurement results, some basic analysis was carried out through applying the investigative classifiers to the raw video data from the validation experiment, as well as using data provided by the LIBSVM library. A significant piece of future work will be carrying on with these investigations.

6.2.1 Experiment simulation

As mentioned, recompleting an experiment with participants using a new classifier was not feasible, due to time constraints. In order to test some of the investigative work, a series of Python scripts were created to simulate an emotion experiment on the Desktop environment. The experiment participant videos from the validation experiment were used as test data for all of the simulations. This allowed the results of the experiment simulations to be directly compared to the results achieved by the original classifier in the experiment.

The first script that was developed was a script to iterate through all of the experiment participant videos, and extract the individual frames for each video. This was necessary as the classifier pipeline used in the project processes data frame by frame. This was achieved using a very similar OpenCV method that was used to split up the video sequences in the GEMEP-FERA emotion database for the original training process (Page 58).

After the frames were extracted, a Python function was developed to obtain emotion measurements for a single frame, using a specified classifier. This process was kept as similar as possible as the experiments carried out using the iOS application. The steps involved for a particular frame were:

1. Obtain the facial landmarks from the frame using the FaceTracker library [10]
2. Compute the same set of 86 distance measures used in the training process from the test facial landmarks
3. Project the test data onto the original PCA space using PCA data saved during classifier training.
4. Scale the test PCA data to the range file generated during classifier training using the LIBSVM svm-scale binary [20]

5. Obtain a set of probabilistic predictions for each emotion from the classifier using a modified version of the `svm_predict_probability` function from the LIBSVM `svm-predict` binary

Once this function had been developed, a script was built to iterate through all of the experiment frames, and to call the function to obtain emotion measurements for each item and experiment participant. All of this data is then collated into CSV-formatted files, so that similar analysis as the original results can take place.

As this work was related to testing emotion classifiers, the Python scripts developed are available in the *EmotionClassifier.zip* file. All of the investigative classifiers discussed in subsequent sections are also available in this file.

6.2.2 Multi-class classifier with distributed training data

Before starting the original training process, it was noted that the training data available for each emotion was not very distributed, with 2274 training images available for 'Sadness', but only 438 training images available for 'Contempt'. After the biases found with the original classifier during analysis, to investigate the effect of this uneven distribution, a new multi-class classifier was produced. The same training pipeline was used, but with all of the emotions having an equally distributed number of training images. As stated, the lowest available number of images for any of the emotions was 438, for 'Contempt', so 438 images were used for each emotion. The images that were used in the new classifier were sampled from each of the emotion databases, with the number of images available in each database acting as the sampling weight.

The distributed classifier achieved a cross-validation rate of 91.8165%. It is worth noting that the amount of data that this cross-validation was applied to was significantly less than original classifier.

After applying the distributed classifier to some of the experiment participant data using the simulation scripts, some of the results produced appear to be more accurate and representative of the experiment video data in comparison to the original classifier. The main example of this was when the average results were compared for the cute animals Twitter feed item (Fig. 6.5).

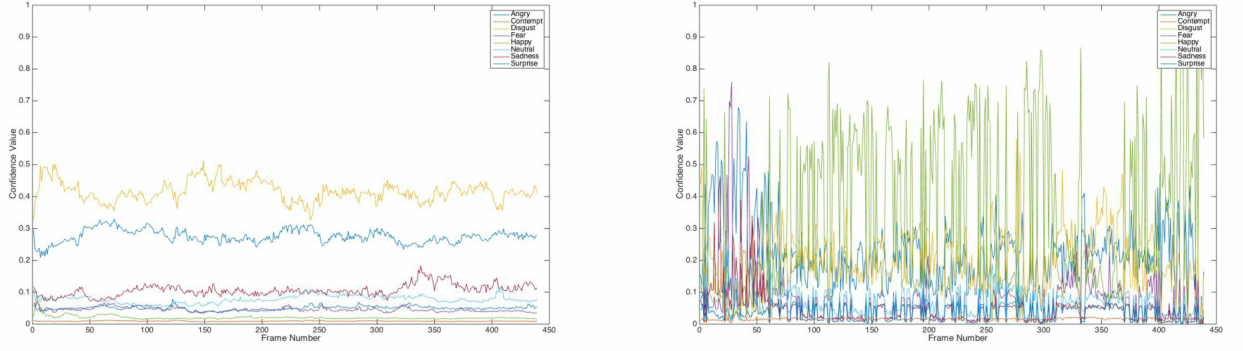


Figure 6.5: Comparison of average emotion measurement results for participants viewing a cute animals Twitter item. Results are from the original classifier used in the validation experiment (left) and a classifier with distributed training data (right)

This item was designed to produce a 'Happy' response for participants, and in the experiment videos, it appears this response was achieved in the majority of cases. However, the results from the original classifier had very low measurements for 'Happy', with the highest probability prediction being less than 0.1. The results from the same item using the distributed classifier look significantly different. 'Happy' (shown in green on the plot) is the dominant emotion in the result, with the highest probability predicted as 0.8670. It is clear that the results for this experiment item were dramatically improved. Similar results have been seen after comparing other the results for other items designed to be 'Happy' or 'Surprise', including the Guinness World Record YouTube video and even the single You've Been Framed tweet. Generally, it appears that the distributed classifier is capable of making much higher predictions for 'Happy' and 'Surprise' than in the original classifier.

However, whilst reviewing other experiment results from this distributed classifier, it appears that there are still sometimes slight biases towards the 'Anger' and 'Disgust' emotions. Whilst these are weaker biases than seen in the original classifier, it is still important to investigate the effects of particular databases on the classifier, to see if the data from a particular database may be overfitting these emotions. As planned, classifiers for each emotion database will be built and tested.

6.2.3 Database specific multi-class classifiers

To understand the potential effects of particular databases on the original classifier, a set of classifiers were trained using each database separately. Table 6.2 outlines the cross-validation rates achieved by each database-specific classifier. As no database used has data for all of the supported emotions, the emotions that the classifiers have been trained on are listed along with the number of images used in the training process. An emotion that all of these classifiers

Emotion database	Emotions available	Total training images	Cross-validation rate
CK+ [21]	Anger, Contempt, Disgust, Fear, Happiness, Sadness, Surprise	654	86.5443%
KDEF [22]	Anger, Disgust, Fear, Happiness, Natural, Sadness, Surprise	1960	84.0816%
JAFFE [23]	Anger, Disgust, Fear, Happiness, Natural, Sadness, Surprise	426	90.0474 %
RAFD [24]	Anger, Contempt, Fear, Happiness, Natural, Sadness, Surprise	2814	91.3912%
GEMEP-FERA [25]	Anger, Fear, Happiness, Sadness	4620	98.6102%

Table 6.2: Cross-validation results of database-specific emotion classifiers

share, and that appeared to produce the main bias in the original classifier is 'Anger'. After applying each of the emotion database specific classifiers to the experiment participant data, the GEMEP-FERA classifier appeared to consistently give much higher 'Anger' predictions compared to any other database's classifier.

The GEMEP-FERA classifier has more than double the training data compared to any of the other emotion databases. However, the classifier only has data for 4 of the 8 supported emotions. The images used from this database was extracted from video sequences, with the emotion labels only being available at a video-level. This may have affected the accuracy of the labelling of the extracted images compared to a database of discrete training images. After considering how much this database contributed to the original classifier process, and seeing the higher 'Anger' predictions when comparing the database specific classifiers, a multi-class classifier was developed without this database to further investigate the effects of this database.

6.2.4 Multi-class classifier with a subset of emotion databases

As mentioned in the previous section, the GEMEP-FERA database [25] has a significant amount more training data than any other database, but only covers 4 out of the 8 emotions that the classifier supports. To investigate if this particular database was overfitting the classifier and producing some of the biases, a new classifier was built using the original training process, but with the GEMEP-FERA data excluded.

The classifier built had a cross-validation rate of 86.3539%. After applying the classifier to some of the raw experiment participant videos, it appears that the 'Anger' and 'Disgust' biases have been significantly reduced. Also, by removing the training images from this database, it has made the training data generally more distributed, and the measurements from other emotions are higher than the original classifier. The main example of this was when the average results were compared for the news blooper YouTube video item (Fig. 6.6).

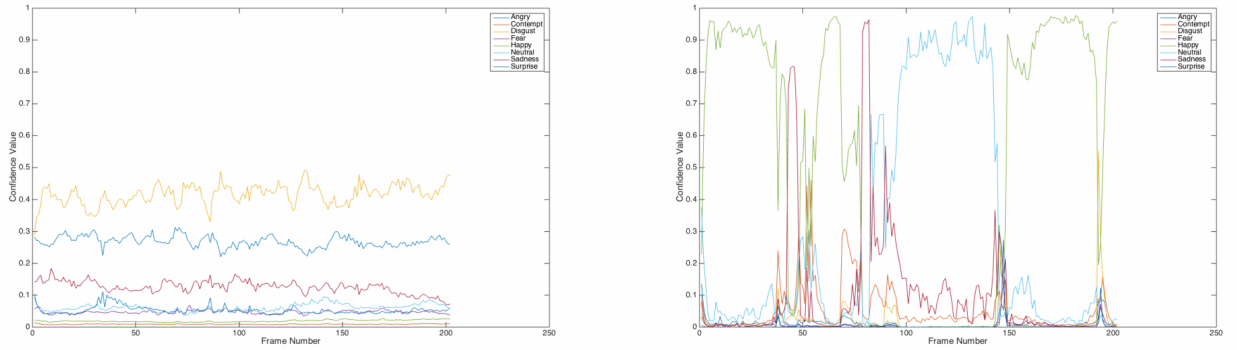


Figure 6.6: Comparison of average emotion measurement results for participants viewing a news blooper item. Results are from the original classifier used in the validation experiment (left) and a classifier with no GEMEP-FERA [25] training data (right)

The news blooper item was a short funny YouTube clip designed to produce 'Happy' response. In the average results for the original classifier, the measurements for 'Happy' were never higher than 0.05. Across all participants, the highest recorded measurement for 'Happy' was 0.1. This is still significantly lower than the average results for 'Anger' and 'Disgust', whose lowest averages were 0.2751, and 0.2305 respectively. The average results achieved for the classifier without the GEMEP-FERA data were completely different. This was particularly evident when comparing the average result charts. For the majority of the experiment item, the intended response of 'Happy' appears to be the dominant emotion, with 'Neutral', and 'Sadness' also recording high values. For a segment of the experiment item's video, a reporter is describing a robbery story, which may have contributed to the 'Sadness' results. However, after analysing all of the experiment items, it appears that the measurements for 'Sadness' are generally much higher than in the original classifier, even in cases where participants are clearly not expressing this emotion in the raw participant videos. This suggests that this classifier has reduced some biases, but introduced another bias towards another emotion.

This piece of investigative work further highlights that there are still issues with the classifier training process, as the process appears to be very sensitive to the training data it is given. By removing the entire GEMEP-FERA database in this classifier, the biases towards 'Anger' and 'Disgust' have been significantly reduced, whilst introducing a new bias towards 'Sadness'. Although the quantity and distribution of the GEMEP-FERA database was not ideal for the original process in comparison to the other databases, in future work, it is important this database is not completely disregarded as there is still some important data that could be selected and used. The distributed classifier used a reduced set of the GEMEP-FERA database and also reduced the 'Anger' and 'Disgust' emotion biases, but not to the same degree as this classifier.

6.3 Requirements evaluation

Before developing the system, a set of basic user requirements were established (Page 31). As part of this evaluation, the system has been analysed against these set of requirements. For each of the requirements, it is explained to what extent the requirement has been met, along with a screenshot if necessary.

Requirement: A means of a user setting up an emotion experiment to be carried out

Priority: Must have

Achieved: Yes

Details: A web application has been developed that allows a user to set up an experiment (Fig. 3.1). There is a 'create' view that allows a user to provide all of the content details, as well as to provide summary details about the experiment such as experiment name and creator. The experiment can then be later modified using a similar 'edit' view, and all created experiments can be viewed on a 'summary' view.

Requirement: An iOS application capable of carrying out emotion experiments

Priority: Must Have

Achieved: Yes

Details: An iOS application has been developed that allows experiment data to be read in, emotion measurements to be captured, and for this data to sent back to the results API. The iOS application was designed to carry out experiments, and this requirement was met by using the application in a validation experiment with 16 people.

Requirement: A means of a user collecting results from an experiment

Priority: Must Have

Achieved: Yes

Details: A results API has been created that allows all of the data captured in an experiment to be collected through a HTTP GET request (Fig. 3.4). Result data can either be collected in bulk through one request, or a particular result can be retrieved through the API by using the generated result ID, available in the results web application.

Requirement: A facility that allows the analysis and visualisation of results

Priority: Should Have

Achieved: Partially

Details: A basic web application has been created that allows the visualisation of experiment results (Fig. 3.2). There is a 'summary' view that shows all of the available results at an item-level for each participant. There is also a 'result' view that shows a line plot of all the emotion measurements captured for a specific result. The reason why this requirement is

partially met, is because there is no real analysis functionality to the system other than manually looking at the graphs. The visualisation functionality is also fixed, and no summary visualisations are available at an experiment level across all participants. This was due to time-constraints associated with needing to carry out a validation experiment. However, the data can easily be downloaded through the API, and visualised/analysed using another tool.

Requirement: A means of recording experiment participants to review with experiment results

Priority: Should Have

Achieved: Yes

Details: For each experiment item viewed by a participant, all of the processed camera frames are saved into array and at the end of the item, a video is generated and saved into the Documents folder of the application on the device. The application has been set up to support iTunes File Sharing, so that all of the videos generated by the application can easily be obtained over USB in iTunes (Fig. 6.7). Each video is named based on the experiment ID, item ID, and time the result was captured.

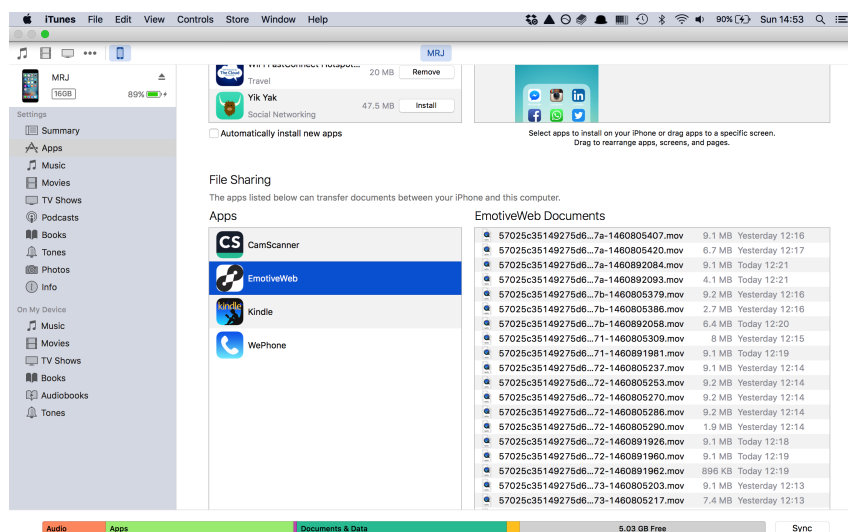


Figure 6.7: Experiment participant videos available through File Sharing in iTunes

Requirement: Support for different types of web-based content that can be used as an experiment

Priority: Should Have

Achieved: Yes

Details:

The iOS application currently uses a combination of system libraries and third party libraries to support 3 different types of content (Fig. 6.8). A single web page can be loaded using an iOS

web view, and the web page can be interacted with like a normal browser. A tweet can be loaded and interacted with using the Fabric library [44]. A YouTube video can be loaded and viewed using the YouTube IFrame Player API [43]. All of these types of content were used in the validation experiment. After reviewing libraries for other types of content, it would be trivial to add more types of content. The 3 types of content supported by the application were considered the priorities for conducting experiments.

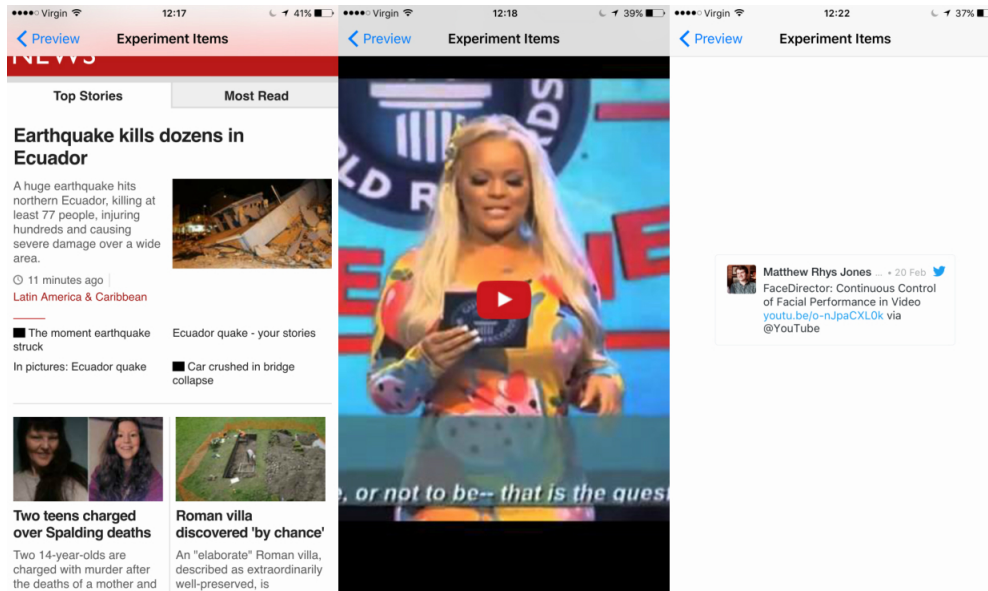


Figure 6.8: Examples of the different types of content supported by the experiment iOS app

Requirement: A login system to allow an experiment and the subsequent results to be viewed only by the user who set up the experiment

Priority: Could Have

Achieved: No

Details: This work was considered after the main development tasks were completed. As this is a proof-of-concept system that was only used by a small number of people, there were other tasks that took priority. In comparison to the other work developed already, the login system should not take much time to implement if required. There are features in the web libraries used in the project that could help to develop this functionality.

Requirement: Emotion metrics based on the emotion measurements gathered during an experiment

Priority: Could Have

Achieved: No

Details: This requirement was considered after seeing the capabilities of some of the existing

commercial solutions. Unfortunately, achieving metrics such as attention or expressiveness would involve a significant amount of research and time, and more work is still required on the raw emotion measurements before these metrics can be considered.

Requirement: Synchronisation of original experiment items and result data when viewing experiment results

Priority: Could Have

Achieved: No

Details: This would have been a useful feature for the experiment results tool, especially with time-sensitive experiment items such as news/social media feeds, but carrying out the validation experiment and completing further classifier work prioritised this work. This has been taken into account for future work.

Conclusions

To conclude, I have thoroughly enjoyed working on this project, and I am happy with what has been achieved in the short timeframe. The main aim of the project was to develop a proof of concept that allowed emotion classification experiments using a mobile device. The system that has been developed comprises of an iOS application, along with 2 web applications and 2 APIs. Whilst there are a small number of known issues with the system, it was still able to be used to conduct a validation emotion classification experiment with 16 participants. Therefore, I believe the general aim of the project has been met. All of the 'Must Have' requirements and the majority of the 'Should Have' requirements that were outlined at the start of the project, have been completed. The remaining requirements were not entirely completed due to the project's time constraints, but the work involved should not be as complex as the work that has already been achieved in the project. The major work involved has been discussed in the Future Work section (Page 82).

The main issue that occurred during the project was the emotion classifier used during the validation experiment. The results of the experiment carried out with the system were not as positive as expected, and highlighted some issues with the classifier, particularly biases towards particular emotions. Luckily, there was still time in the project to develop new classifiers that considered factors such as the emotion databases used and the distribution of training data more. After rerunning the classifier on experiment participant videos, and comparing the results, it appears some progress has been made to develop more accurate and robust classifiers. However, there are still known issues with all of the classifiers developed during this investigative work such as biases moving from one emotion to another, or there not being enough training examples for certain emotions. Generally, the classifier training process appears to be very sensitive to the training data it is given. Ideally, I would have liked more time at the end of the project to consider other training data combinations, or to potentially source more data. Unfortunately, due to the time constraints of the project, this was not possible, but it has been noted as a major piece of future work.

The developed system has been designed to modular, and to provide general-purpose tools for emotion classification. Several emotion classifiers have been built, along with an iOS application capable of carrying out experiments, and web-based tools to support this process. All of the work in this project has been documented, and is available to review on GitHub. I believe there is definite potential for some of the project to be further developed or re-used in future research. At a minimum, the project highlights what is possible in this research area on a mobile device, and highlights some important factors to consider when working on a similar project, particularly when developing emotion classifiers.

Chapter 8

Future Work

After developing this proof-of-concept system, there are now many possible pieces of future work. Some of the key pieces of work that could be considered if this work is continued have been outlined below.

8.1 Emotion classifier

One of the key improvements for the system would be to conduct more analysis and improvements on the emotion classifier. For the majority of this project, an emotion classifier from the "Facial Expression Analysis on Smartphone" CUROP project was used in order to get a viable proof-of-concept system working as soon as possible. This classifier was a good enough starting point for the project, but had biases towards particular emotions, primarily due to a lack of training data when it was developed. To try and resolve this, in the first few weeks of this project, time was spent investigating and obtaining more training data. In the latter half of the project, a new emotion classifier was developed using the 2 original emotion databases from the CUROP project, alongside 3 new emotion databases that were sourced during this project. As the iOS application component of the system was designed to allow the ability of changing classifiers easily, the newly developed classifier was tested and integrated into the project before the main project experiment took place. Whilst this classifier initially appeared to be an improvement on the classifier developed during the CUROP project, issues were discovered whilst conducting experiments, and more classifier work was completed. At this point, more analysis and improvements are still needed. Some potential pieces of future work that could be completed include:

- *Utilise video data generated during emotion experiments in the training of a new classifier.* The iOS application component is capable of recording participants whilst they complete an experiment, and this video data can easily be retrieved. This data could then be reviewed, manually labelled, and added to the set of training data. Whilst this would be time consuming, it would allow a lot of data to be sourced without needing to obtain any

more 3rd party databases. Also, this data would be more similar to the test data than the current training data from databases as it will have come from the same system.

- *Analyse the features used by the classifier.* Currently 86 distance measures are calculated from facial landmarks, and these are then subject to Principal Component Analysis (PCA). This process currently produces 18 vectors, but if the PCA results were analysed more, the majority of the variation needed for the emotion classification may be found in a smaller number of vectors. With a suitable amount of data, analysis of this process could take place using a tool such as Python or MATLAB.
- *Consider a different approach to building the classifier.* Recent research from Carnegie Mellon University's Human Sensing Laboratory has found that a personalised approach to building an SVM classifier can achieve better results than a generic SVM classifier like the one used in the project through simultaneously learning a classifier and reweighting training data. Some of the C++ functions from this research are available online, and it would be interesting to compare results from this type of classifier with the current classifier [1]. With the recent public release of Google's machine intelligence library TensorFlow [50], it would also be more feasible to consider a deep learning approach to building the classifier.

8.2 Results API

Another future improvement and one that was only envisaged towards the end of the project is improving the scalability of the system's experiment results API. As experiment results generate a large amount of data that needs to be sent/retrieved using the API, having an API that is able to cope with this level of data will be important. This only became an issue towards the end of the project when several participants had completed an experiment. Although the results were reachable through the API, there were a small number of crashes on the server that hosts the API whilst doing so. Some potential pieces of future work that could be completed include:

- *Review the API's data model.* The data model that is used by the API was designed at the start of the project and including a field for facial tracking data. As this data is now easily available through applying the facial tracker to the recorded experiment participant video, the API's data model could be changed to remove the tracking data field. This should make a noticeable difference as each result frame produces 86 key/value pairs of tracking data.
- *Implement additional API calls.* At present, to view a summary of all available results, a GET request is used to retrieve all available results data. When this call is made, the majority of the data returned is not needed for a summary such as the facial tracking

data and the classifier data. Additional API calls that allowed only the fields required for a summary would reduce the API's load significantly.

- *Investigate the server's configuration.* Even if improvements are made to the API, the server itself still may not be able to cope with many requests at one time, due the resources it has been configured with. The server used in the project currently has 512MB RAM and 20GB storage. If future experiments were planned, it would be advisable to investigate and possibly upgrade the server's configuration.

8.3 Experiment design

Due to the limited time in the project, there was only time to design and execute a single experiment to validate the system. If this system was used for similar emotion experiments in the future, it would be a good idea to spend more time on the design of the experiment. Whilst I have been able to integrate some feedback from PhD students within the school, I believe that more work could be done to make the experiment results more useful than just to validate the system. Some potential pieces of future work that could be completed include:

- *Add a short questionnaire to the experiment.* The experiment that was carried out to validate the system only allowed participants to add general comments at the end of the experiment. In order to establish ground truth and to get more insight into particular results, it may be useful to have some kind of questionnaire with more structure such as asking a user to state the response they believe they had to an item, and to compare this with the response measured by the system.
- *Investigate timings more.* It is difficult to know exactly how long particular experiment items and delays between items should be shown to experiment participants for. For the purposes of this project, a hard coded delay was added to the iOS application of 3 seconds, and arbitrary times were chosen based on content length. It is important that different timings are considered in future experiments as they have important psychological implications that could affect the results. This would involve trialling different timings in experiments, and researching any existing psychological work that may assist with setting these timings.

8.4 Analysing/visualising experiment results

An area of the project that could use additional work is the functionality available when viewing experiment results. Currently, a user is only able to view a graph for a single experiment item from a single experiment participant. This is an intuitive way of seeing how a participant

responded to a particular piece of content but it doesn't give much scope for further analysis or visualisation. Some potential pieces of work that could be completed include:

- *Implement the ability of downloading of result data.* If the user is able to download result data for a single item or a whole experiment in a suitable format i.e. CSV, they would be able to use this data for other applications such as in MATLAB or Python. This would involve developing functions or using a library to convert the Result API JSON into a CSV format, and providing a link to this data on the web application. This process is currently manually completed using a Python script once the API result data has been obtained.
- *Implement the ability to select emotions.* Currently, the chart produced by the system shows the results for all emotions. It would be useful to allow a user to select a subset of these emotions if they are only interested in particular ones. This would involve adding UI controls such as checkboxes for each emotion, and then implementing some code to keep track of the state of these checkboxes for displaying the chart.
- *Implement more means of comparing experiment results.* Currently if a user would like to compare the results for multiple participants who viewed the same experiment item, they would have to manually open the results web application multiple times. A potential feature would be to allow a user to view the result charts for multiple participants in the same view. This would involve developing a new view where a user could enter multiple result IDs, and a similar visualisation could then be generated for each ID entered.
- *Introduce summary statistics.* Summary statistics such as number of participants, minimum/maximum confidence values for each emotion, and some summary plots could appear on the web application. Some of these have been manually calculated for this project, but automatic calculation of these statistics would be a useful feature on the web application. This would require a significant amount of code to retrieve all of the result data, and to process these statistics in the background. One potential solution would involve using the data tools available within D3 [51], such as *d3.deviation*, *d3.mean*, and *d3.min*. This is an underlying library already available in the project.

8.5 Emotion experiment capture tool

Whilst the current experiments iOS application was able to carry out the validation experiment successfully, there are still potential improvements/new features that could be added to the application. Some potential pieces of work that could be completed include:

- *Additional support for different types of content.* The system's ability to support a type of content is dependent on an appropriate means of displaying the content on the iOS

application. In the future, it may be interesting to add more support for different types of content. For image content, images from the photo sharing site Flickr [52] could be integrated using the FlickrKit library [53]. If audio content was considered more, SoundCloud [54] content could be integrated using the Embedly library [55]

- *Implement screen capturing capability.* An experiment may involve time-specific content that will differ between participants, and between experiment execution and analysis. For example, if an experiment participant views a Twitter feed in the morning, by the afternoon, the content will likely be different. To counter this, a potential feature would be to capture the contents of the device screen during an experiment. This could be achieved through taking regularly timed screenshots of the screen whilst an experiment is taking place, and to retrieve these screenshots using iTunes when the participant videos are retrieved. This would provide more context on the exact content that was viewed when analysing experiment results.
- *Improved navigation for webpage experiment items.* A small issue noted by participants in the validation experiment was that the webpage experiment items did not allow you to navigate back a page if you interacted with content. This was due to having limited screen space, and not wanting to introduce a navigation bar for webpage items. The *UIWebView* class used to display the webpages does maintain a list of all previously webpages, and has built-in support for 'Back' and 'Forward' buttons [56]. If a larger device was used, or this functionality became a requirement, it would not require much additional work.

Reflection and Learning

9.1 Reflection

On reflection, if I could change how I approached the implementation of the system, I would have made the development phase more test-driven. Having code tests in the project, particularly in the API and web applications, would have allowed me to identify issues such as the system's scalability at an earlier point than just through manual testing. Testing would also have made the development more structured through having to clearly understand the system requirements and features to implement before writing any code. I spent a long time debugging issues such as the stability of the API and data formatting errors, with little indication of where the code was failing. Whilst writing tests would have taken up some time in the project, I would have been able to identify and resolve such issues more easily because of the information from the test results.

Another area of improvement in the project would have been to allocate more time to the experiment phase. Whilst I was able to complete an experiment with an acceptable number of participants, the experiment had the potential to be more interesting than it was. I only considered designing my main experiment after finishing the development of the entire system, despite the full system not being needed to design an experiment. This identifies a potential weakness in the approach I took for the project, where I divided the project into discrete phases. Ideally, the experiment should have been designed as soon as it was feasible, to give me more time to meet with the PhD students involved in my project, and to use more of their feedback whilst still in the main development phase of the project. I was able to use some of the feedback given such as adding in intervals between experiment items shown, but some of the feedback ended up as future work due to time constraints. The experiment was designed to validate the system, and I do think that the experiment executed achieved that. However, there was scope to improve the experiment if my approach had been slightly different.

Finally, it would have been useful to be more aware of the final report deliverable, and to try to spread some of the report work throughout the project. Although I made notes as I completed different tasks, these were often quite vague and didn't help a great deal when writing my report, apart from some of the code documentation I produced. I didn't properly start to write the final report until after I had completed the main objectives of the project. In hindsight, it would have been beneficial to complete sections such as the 'Background' and 'Specification & Design' earlier in the project, to reduce the amount of writing work required at the end of the project. Also, by having to think about and write these sections, it would have likely had a positive effect on the project implementation and experiments.

9.2 Learning

This project involved a variety of development work that has allowed me to expand my software development skills. Although I had prior experience in all of the programming languages used in the project, many of the tasks I had to complete required a much deeper understanding than I possessed prior to starting. The main example of this was the development of the emotion experiments iOS application. I had previously developed an iOS application that had face tracking and emotion classification functionality, but this project required extra levels of functionality that I had no experience of. I had to become familiar with various third-party libraries such as the YouTube iOS Helper [43] and Twitter Fabric [44], as well as having to learn about network programming in Objective-C to connect the application to other components of the system. These are just a couple of examples of the exposure I have had to different technologies. By having complete autonomy over the development work, I have become more competent in reviewing different languages/libraries/tools, and I feel confident in making development decisions based on suitability for the task, instead of just choosing what I am familiar with. I believe that this will be a useful skill to have in future development work.

Another area of development during the project was my research skills. In particular, I have learnt the importance of staying up-to-date with the current research field. Being aware of existing work, has informed many of the decisions made in the project. One example of this is when I was aware of a previous final year project in the School [8], that had used a face tracker on the iOS platform. Being aware and reading about this work allowed me to understand the reasons for choosing an ASM face tracker in theory, as well as seeing the practicalities of using a particular ASM library on a mobile platform. If I had not been aware of this prior work, I would have spent more of the project researching and testing other face tracker approaches that may have been completely unsuitable. Another example of how existing research has informed my project was when I was choosing which emotions to support in the system. As I had already researched existing emotion databases, and read papers about other work on emotion classification, I was aware that very similar sets of emotions were used in the different

pieces of work. If I had chosen an arbitrary set of emotions, it is unlikely that I would have had as much success in the project, as my work would have been less compatible with existing work. Having this understanding of current work has also helped me to critically evaluate and compare my work throughout the project.

Time management in the project was extremely important, to deliver what I had outlined in initial plan. The main learning point for me whilst managing the project was how to respond to unexpected changes to the planned project timeline. There were a couple of occasions where the project encountered a delay due to an unforeseen technical problem or for personal reasons. For example, whilst trying to develop an emotion classifier, one of the hard drives containing all of the training data I planned to use became corrupt. I was able to deal with problems like this through being open with my supervisor as early as possible and through also ensuring that I had a list of tasks that could be completed at any one time, so that if no progress could be made on a task, I didn't need to spend a long time thinking of what to do instead. Another area of my project management skills that I developed was managing source code. In my CUROP project, although I used Github to manage the project's codebase, I did not regularly commit to the repository and there would often be large changes in functionality between commits. As there were many components and features to implement in this project, I ensured that I set up repositories at the start of the project, and that I made regular commits whenever there was a change in the project. By managing the project's code in this way, I was able to commit a stable version of a component to the repository, and then I could test a new experimental feature without making the component unstable. By having more manageable commits, it also became easier to review the development progress made, against the project time plan defined in the Initial Plan [3] .

List of Abbreviations

- **AAM** Active Appearance Model
- **API** Application program interface
- **ASM** Active Shape Model
- **CK+** Extended Cohn-Kanade Dataset
- **CRUD** Create-Read-Update-Delete
- **CUROP** Cardiff Undergraduate Research Opportunities Programme
- **FACS** Facial Action Coding System
- **FERA** Face and Gesture Recognition
- **GEMEP** Geneva Multimodal Emotion Portrayals
- **JAFPE** The Japanese Female Facial Expression (Database)
- **JSON** JavaScript Object Notation
- **KDEF** Karolinska Directed Emotional Faces
- **MEAN** MongoDB, ExpressJS, AngularJS, NodeJS
- **PDM** Point Distribution Model
- **RAFD** Radboud Faces Database
- **RMS** Root-mean-squared
- **UI** User Interface

Bibliography

- [1] Chu, W.S., de la Torre, F., Cohn, J. "Selective Transfer Machine for Personalized Facial Expression Analysis," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume:PP, Issue: 99.
- [2] Microsoft. (2015). *Microsoft Project Oxford Emotion API*. Available: <https://www.projectoxford.ai/emotion>. Last accessed 29th Mar 2016.
- [3] Jones, M. (2016). *Automatic emotion capture when viewing web-based media on a smartphone: Initial Plan*
- [4] Google Play. (2016). *Samsung LOOK AT ME - Android Apps on Google Play*. Available: <https://play.google.com/store/apps/details?id=com.samsung.lookatme>. Last accessed 22nd Apr 2016.
- [5] PR Newswire Association. (2014). *Affectiva expands emotion insights to TV and online programming, after signing multi-year agreement to standardize facial coding for Fortune 500 brands*. Available: <http://www.prnewswire.com/news-releases/affectiva-expands-emotion-insights-to-tv-and-online-programming-after-signing-multi-year-agreement-to-standardize-facial-coding-for-fortune-500-brands-222818881.html>. Last accessed 22nd Apr 2016.
- [6] Luxury Daily. (2015). *Bentley pushes Bentley with emotional recognition and innovative app*. Available: <http://www.luxurydaily.com/bentley-pushes-bentley-with-emotional-recognition-and-innovative-app/>. Last accessed 22nd Apr 2016.
- [7] Jones, M. (2015). *Face tracking and emotion classification iOS app*. Available: <https://github.com/mrhysjones/cuop>. Last accessed 29th Mar 2016.
- [8] Harley, T. (2013). *Video Realistic Facial Modelling and Synthesis - BSc Thesis*. Available: <https://www.cs.cf.ac.uk/PATS2/>
- [9] Saragih, J. M., Lucey, S., Cohn, J F. (2011). "Deformable Model Fitting by Regularized Landmark MeanShift" in *International Journal of Computer Vision*. 91(2) pp.200-215

- [10] McDonald, K. (2015). *Real time deformable face tracking in C++ with OpenCV 2*. Available: <https://github.com/kylemcdonald/FaceTracker>. Last accessed 31st Mar 2016.
- [11] Cootes, T.F., Taylor, C.J., D.H. Cooper, D.H., Graham, J.(1995). *Active shape models - their training and application*". *Computer Vision and Image Understanding*, 61, 38-59.
- [12] Sonka, M., Hlavac, V., Boyle, R. (2008). *Image Processing, Analysis, and Machine Vision*. Toronto: Thomson Learning.
- [13] MULTIPLE.ORG. (2009). *The CMU Multi-PIE Face Database*. Available: <http://www.multiple.org/>. Last accessed 1st Apr 2016.
- [14] Ekman, P. (1969). Pan-cultural elements in facial displays of emotions. *Science*, 164, 86-88.
- [15] Ekman, P. (1999). Basic Emotions. In Dalglish, T. & Power, M. J. (Eds.), *Handbook of Cognition and Emotion*, 45-60.
- [16] Ekman, P., Friesen, W. (1978). *Facial Action Coding System: A Technique for the Measurement of Facial Movement*. Consulting Psychologists Press, Palo Alto.
- [17] DataFace. (2003). *Facial Action Coding System Affect Interpretation Dictionary (FACSAID)*. Available: <http://www.face-and-emotion.com/dataface/facsaid/description.jsp>. Last accessed 22nd Apr 2016.
- [18] Thuseethan, S., Kuhanesan, S. (2014). *Eigenface Based Recognition of Emotion Variant Faces*. Available: <http://iiste.org/Journals/index.php/CEIS/article/view/14100>. Last accessed 28th Apr 2016.
- [19] Hsu, C. W., Chang, C. C., & Lin, C. J. (2003). *A practical guide to support vector classification*.
- [20] Jin, Chih-Jen., Chang, Chih-Chung. (2015). *LIBSVM – A Library for Support Vector Machines*. Available: <https://www.csie.ntu.edu.tw/~cjlin/libsvm/>. Last accessed 7th Apr 2016.
- [21] Lucey, P., Cohn, J. F., Kanade, T., Saragih, J., Ambadar, Z., Matthews, I. (2010). *The Extended Cohn-Kande Dataset (CK+): A complete facial expression dataset for action unit and emotion-specified expression*. Paper presented at the Third IEEE Workshop on CVPR for Human Communicative Behavior Analysis (CVPR4HB 2010)
- [22] Lundqvist, D., Flykt, A., & Ohman, A. (1998). *The Karolinska Directed Emotional Faces - KDEF*, CD ROM from Department of Clinical Neuroscience, Psychology section, Karolinska Institutet, ISBN 91-630-7164-9.

- [23] Kyushu University. (1998). *The Japanese Female Facial Expression (JAFPE) Database*. Available: <http://www.kasrl.org/jaffe.html>. Last accessed 30th Mar 2016.
- [24] Radboud University Nijmegen. (2010). *Radboud Faces Database - Overview*. Available: <http://www.socsci.ru.nl:8180/RaFD2/RaFD?p=overview>. Last accessed 30th Mar 2016.
- [25] SSPNET. (2011). *GEMEP-FERA*. Available: <http://sspnet.eu/2011/05/gemep-fera/>. Last accessed 30th Mar 2016.
- [26] Eyeris. (2015). *Welcome to the Eyeris EmoVu SDK Documentation*. Available: http://emovu.com/docs/html/getting_started.htm. Last accessed 7th Apr 2016.
- [27] Eyeris. (2015). *EmoVu - Try Demo*. Available: <http://emovu.com/w/1a2d>. Last accessed 7th Apr 2016.
- [28] FacioMetrics LLC. (2015). *IntraFace on the App Store*. Available: <https://itunes.apple.com/us/app/intraface/id937424937?mt=8>. Last accessed 7th Apr 2016.
- [29] Microsoft. (2016). *Microsoft Cognitive Services - Face API*. Available: <https://www.microsoft.com/cognitive-services/en-us/face-api>. Last accessed 22nd Apr 2016.
- [30] Microsoft. (2016). *Microsoft Cognitive Services - Emotion API*. Available: <https://www.microsoft.com/cognitive-services/en-us/emotion-api>. Last accessed 22nd Apr 2016.
- [31] Microsoft. (2016). *Microsoft Cognitive Services - Computer Vision API*. Available: <https://www.microsoft.com/cognitive-services/en-us/computer-vision-api>. Last accessed 22nd Apr 2016.
- [32] Microsoft. (2015). *Happy? Sad? Angry? This Microsoft tool recognizes emotions in pictures - Next at Microsoft*. Available: <http://blogs.microsoft.com/next/2015/11/11/happy-sad-angry-this-microsoft-tool-recognizes-emotions-in-pictures/>. Last accessed 22nd Apr 2016.
- [33] Affectiva. (2015). *Emotion Recognition Software and Analysis - Affectiva*. Available: <http://www.affectiva.com/>. Last accessed 22nd Apr 2016.
- [34] Creately. (2016). *Online Diagram Software to draw Flowcharts, UML & more | Creately*. Available: <http://creately.com/>. Last accessed 17th Apr 2016.
- [35] Balsamiq. (2016). *Balsamiq Mockups*. Available: <https://balsamiq.com/products/mockups/>. Last accessed 10th Apr 2016

- [36] Bootstrap. (2016). *Bootstrap -The world's most popular mobile-first and responsive front-end framework*. Available: <http://getbootstrap.com/>. Last accessed 10th Apr 2016.
- [37] MEAN.IO. (2016). *MongoDB, Express, Angularjs Node.js powered fullstack web framework*. Available: <http://mean.io/#!/>. Last accessed 10th Apr 2016.
- [38] mongoDB. (2016). *NoSQL Databases Explained*. Available: <https://www.mongodb.com/nosql-explained>. Last accessed 23rd Apr 2016.
- [39] Node.js Foundation. (2016). *Express - Node.js web application framework*. Available: <http://expressjs.com/>. Last accessed 23rd Apr 2016.
- [40] Google. (2015). *AngularJS - Superheroic JavaScript MVW Framework*. Available: <https://angularjs.org>. Last accessed 23rd Apr 2016.
- [41] Arase, K. (2015). *QR Code Generator implementation in ActionScript3, Java, JavaScript and more*. Available: <https://github.com/kazuhiroarase/qrcode-generator>. Last accessed 10th Apr 2016.
- [42] Ehrlich, C. (2015). *CEMovieMaker: CEMovieMaker is a quick and dirty way to create a movie from and array of UIImages*. Available: <https://github.com/cameronehrlich/CEMovieMaker>. Last accessed 10th Apr 2016.
- [43] Google Developers. (2016). *YouTube IFrame Player API*. Available: https://developers.google.com/youtube/v3/guides/ios_youtube_helper. Last accessed 5th Apr 2016.
- [44] Twitter. (2016). *Fabric - Twitter's Mobile Development Platform*. Available: <https://get.fabric.io/ios?locale=en-us>. Last accessed 5th Apr 2016.
- [45] Skipor, T. (2016). *Angular-nvD3*. Available: <http://krispo.github.io/angular-nvd3/#/>. Last accessed 10th Apr 2016.
- [46] OpenCV. (2015). *OpenCV*. Available: <http://opencv.org/>. Last accessed 10th Apr 2016.
- [47] matplotlib. (2016). *matplotlib: python plotting*. Available: <http://matplotlib.org/>. Last accessed 17th Apr 2016.
- [48] Apple. (2014). *Navigation Controllers*. Available: <https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/NavigationControllers.html>. Last accessed 17th Apr 2016.

- [49] Apple. (2012). *Dispatch Queues*. Available: <https://developer.apple.com/library/ios/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html>. Last accessed 17th Apr 2016.
- [50] TensorFlow. (2016). *TensorFlow – an Open Source Software Library for Machine Intelligence*. Available: <https://www.tensorflow.org/>. Last accessed 22nd Apr 2016.
- [51] D3.js. (2016). *Data driven documents*. Available: <https://d3js.org/>. Last accessed 22nd Apr 2016.
- [52] Flickr. (2016). *Flickr - Photo Sharing!* Available: <https://www.flickr.com/>. Last accessed 22nd Apr 2016.
- [53] Casserly, D. (2015). *FlickrKit - An iOS Flickr Framework, written in Objective-C*. Available: <https://github.com/devedup/FlickrKit>. Last accessed 22nd Apr 2016.
- [54] SoundCloud. (2016). *SoundCloud - Hear the world's sounds*. Available: <https://soundcloud.com/>. Last accessed 22nd Apr 2016.
- [55] Embedly. (2016). *SoundCloud Embedly Provider*. Available: <http://embed.ly/provider/soundcloud>. Last accessed 22nd Apr 2016.
- [56] Apple. (2012). *Paging Back and Forward*. Available: <https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/DisplayWebContent/Tasks/BackForwardList.html>. Last accessed 22nd Apr 2016.