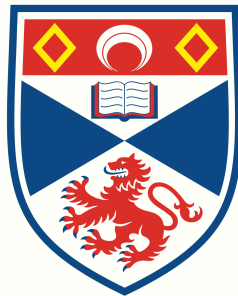


# Affective Mirror: Automated emotion detection through photoplethysmography & facial expression analysis

Melissa Mozifian



University of  
St Andrews

This dissertation is submitted in partial fulfilment for the degree of  
*Bachelor of Science*  
at the University of St Andrews

April 2014



# Abstract

Facial expressions are some of the most powerful channels of nonverbal human communication. Emotions can be considered as a feedback mechanism that reveals our thoughts, feelings, expectations, etc. Even though determining how a person is feeling based on their expressions is trivial for us, it is quite a challenging task for computers. Current research has proposed various techniques such as machine learning to address this issue. On the other hand, computers are capable of analysing many other forms of signals that humans cannot observe with the naked eye. For instance, emotions such as excitement give rise to an increased heartbeat rate.

The goal of this project is to develop a system that can detect emotions such as happiness and excitement by analysing a video feed received from a regular webcam. Emotion detection is performed in real-time by simultaneously feeding the same video stream to a facial expression analysis algorithm and a photoplethysmography algorithm that estimates the rate of heartbeat. Both their outputs are visualised as a whole to determine the type of emotion shown by the subject. As the analysis is performed only using the webcam, the user is not constrained by the need to wear any devices. Such a capability, can help build systems that are more effective of extracting patterns useful for detecting a multitude of emotions. We argue that this technology has the potential to bring new perspectives to the development of medical and psychological applications. Finally, we evaluate the effectiveness of the proposed system with the help of trained models for detecting emotions on human participants.





# Acknowledgements

I wish to thank my supervisor Dr Per Ola Kristensson who helped and guided me throughout this dissertation project. You always guided me towards the right path, and pushed me to further challenge my skills. Your encouragement and brilliant ideas were invaluable for this project.

My gratitude goes towards all the lecturers who were always happy to be approached at any time and provided full support.

I would like to thank the technicians for bearing with me and providing support when I needed equipments such as hardware and software.



# **Declaration**

## **Candidate's Declarations**

I hereby certify that this dissertation, which is 14,735 words in length, has been composed by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree. Credit is explicitly given to others by citation or acknowledgement. This project was conducted by me at The University of St Andrews from Sep/2013 to Apr/2014 towards fulfilment of the requirements of the University of St Andrews for the degree of BSc under the supervision of Dr. Per Ola Kristensson. In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work

Melissa Farinaz Mozifian



# CONTENTS

<b>Contents</b>	<b>i</b>
<b>List of Figures</b>	<b>v</b>
<b>Code Listings</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Overview . . . . .	1
1.1.1 Terminology . . . . .	2
1.2 Problem Definition . . . . .	2
1.3 Project Objectives . . . . .	4
1.3.1 Primary Objectives . . . . .	4
1.3.2 Secondary Objectives . . . . .	4
1.3.3 Tertiary Objectives . . . . .	4
1.4 Main Contributions . . . . .	4
1.5 Software Development Methodology . . . . .	5
1.5.1 Source Control Management . . . . .	5
1.5.2 Agile Development Approach . . . . .	5
1.5.3 Test-Driven Development . . . . .	5
1.6 Ethics . . . . .	5
<b>2 Context Survey</b>	<b>7</b>
2.1 Introduction . . . . .	7
2.2 Affective Computing . . . . .	7
2.2.1 Two-Factor Theory of Emotion . . . . .	8
2.2.2 Physical vs Cognitive . . . . .	8
2.2.3 Sentic Modulation . . . . .	8
2.2.4 Facial Expression . . . . .	9
2.2.5 Social Signal Processing . . . . .	10
2.2.6 Honest Signals . . . . .	11
2.2.7 Contagious Mood Signals . . . . .	11
2.2.8 Emotion Recognition & Emotional Response . . . . .	12
2.3 Techniques for Facial Expression & Emotion Analysis . . . . .	12
2.3.1 Overview . . . . .	12
2.3.2 Existing Tools & Libraries . . . . .	12
2.4 Summary . . . . .	14

<b>3</b>	<b>Conceptual Framework</b>	<b>17</b>
3.1	Facial Expression Analysis . . . . .	17
3.1.1	Kanako Library Overview . . . . .	17
3.2	Pulse Detection . . . . .	21
3.2.1	Face & Forehead Detection . . . . .	21
3.2.2	PPG Overview . . . . .	21
3.3	Emotion Visualisation . . . . .	22
3.3.1	Data Communication . . . . .	22
3.3.2	Visualised Emotions . . . . .	23
3.3.3	Pulse Graphs . . . . .	23
<b>4</b>	<b>System Specification &amp; Architecture</b>	<b>25</b>
4.1	User Requirements . . . . .	25
4.2	Architecture & High Level Design . . . . .	25
4.3	Data Model . . . . .	27
4.4	System Integration . . . . .	29
4.4.1	Integration with Kanako . . . . .	29
4.4.2	External Library Integration . . . . .	30
<b>5</b>	<b>Implementation</b>	<b>33</b>
5.1	External Dependencies . . . . .	33
5.1.1	OpenCV . . . . .	33
5.1.2	Boost . . . . .	33
5.1.3	GSL . . . . .	33
5.1.4	CUDA 5.0 or above . . . . .	34
5.2	Face Detection . . . . .	34
5.2.1	<i>detectMultiScale</i> API . . . . .	34
5.2.2	Isolating the Forehead Region . . . . .	36
5.3	Pulse Detection . . . . .	36
5.3.1	Strategy for Porting Code . . . . .	37
5.3.2	Implementation . . . . .	37
5.3.3	Problems Encountered . . . . .	42
5.4	Facial Expression Analysis . . . . .	43
5.4.1	Kanako Library Integration . . . . .	43
5.5	Visualisation . . . . .	43
5.5.1	Emotion Intensity Bars . . . . .	43
5.5.2	BPM Graph . . . . .	44
5.5.3	Limitations & Further Improvements . . . . .	45
<b>6</b>	<b>Evaluation</b>	<b>47</b>
6.1	Pulse Detection Evaluation . . . . .	47
6.1.1	C++ vs Python Implementation Comparison . . . . .	48
6.1.2	Experiments Conducted on Human Participants . . . . .	49
6.1.3	Further Analysis . . . . .	51
6.1.4	Future Evaluation Work . . . . .	54

<b>7</b>	<b>Critical Assessment</b>	<b>55</b>
7.1	Challenges . . . . .	55
7.1.1	Learning a New Language & Platforms . . . . .	55
7.1.2	Challenges of Using Kanako . . . . .	56
7.1.3	Porting an Existing Algorithm . . . . .	57
7.2	Limitations . . . . .	57
7.2.1	Pulse Detector Algorithm Sensitivity to Motion . . . . .	57
7.2.2	Sensitivity of Algorithms to Sources of Illumination . . . . .	58
7.2.3	Deficinecies . . . . .	58
7.3	Future Work . . . . .	59
7.3.1	Improved Facial Analysis Library . . . . .	59
7.3.2	Improving Pulse Detection Algorithm . . . . .	59
7.3.3	Cognitive & Physiological Mapping Based on Further Evaluation . . . . .	59
7.3.4	Eulerian Video Magnification for Revealing Subtle Changes . . . . .	59
7.3.5	Multiple Face Analysis . . . . .	60
7.3.6	Stronger Face Detection Algorithm . . . . .	60
7.3.7	Adjustable Forehead Region . . . . .	60
7.3.8	User-tailored System . . . . .	60
7.3.9	Voice Processing . . . . .	60
7.3.10	Further Testing . . . . .	61
7.4	Application . . . . .	61
7.4.1	Teaching App . . . . .	61
7.4.2	Angry Driver Recognition . . . . .	62
7.4.3	Effect of Art on Emotions . . . . .	62
7.4.4	Lie Detector . . . . .	62
7.4.5	User-tailored Recommendation App . . . . .	63
<b>8</b>	<b>Conclusions</b>	<b>65</b>
8.1	Summary . . . . .	65
8.2	Current Status . . . . .	65
8.3	Further Complications . . . . .	66
	<b>Appendix A Testing</b>	<b>67</b>
A.1	Algorithm Correctness . . . . .	67
A.1.1	Unit Tests . . . . .	67
A.2	Screen-shot Samples of Running Program . . . . .	72
	<b>Appendix B Pulse Detection Algorithm</b>	<b>77</b>
	<b>Appendix C Experimental Result</b>	<b>79</b>
	<b>Appendix D Changes to the Original Specification</b>	<b>81</b>
D.1	Implementing a Face Analysis library from scratch . . . . .	81
D.2	Eulerian Video Magnification . . . . .	81
	<b>Appendix E Project Plan</b>	<b>83</b>

<b>Appendix F Software Listing</b>	<b>85</b>
<b>References</b>	<b>87</b>



# LIST OF FIGURES

3.1	Two set of 2D-points belonging to two different classes [1]. . . . .	19
4.1	Basic architecture overview. . . . .	26
4.2	Use-case diagram displaying interaction with the user and components. . . . .	27
4.3	Data flow between the components. . . . .	27
4.4	Component interaction and trained classifiers requirements. . . . .	28
4.5	Top-level component decomposition view of the Affective Mirror system along with interactions among the components. . . . .	28
4.6	Initial design of system before applying the complete integration. . . . .	29
4.7	Tight integration with Kanako. . . . .	30
4.8	Hierarchy layer architecture of system. . . . .	30
4.9	External libraries used by the two main components of Affective Mirror. . . . .	31
5.1	Process of capturing frame and pulse detection algorithm. . . . .	38
6.1	Python vs C++ implementation for a 30 seconds pre-recorded video. . . . .	49
6.2	Python vs C++ implementation running live for about 1 minute while wearing a heart monitor device. . . . .	50
6.3	Three different adjustments applied to the algorithm separately. . . . .	51
6.4	Increased forehead region to cover the entire forehead. . . . .	52
6.5	Averaging over RGB vs Green channel. . . . .	53
6.6	RGB vs Greyscale Frame. . . . .	54
A.1	Detecting happiness. . . . .	73
A.2	Detecting surprise. . . . .	73
A.3	Detecting sadness. . . . .	74
A.4	Detecting happiness and slight excitement. . . . .	74
A.5	Detecting excitement. . . . .	75
A.6	Detecting neutral. . . . .	75
A.7	Detecting fear. . . . .	76
C.1	Experiments Result. . . . .	80
E.1	Planning & Weeks Spent. . . . .	84

# CODE LISTINGS

5.1	Face Detection . . . . .	34
5.2	Mean calculation . . . . .	37
5.3	Interpolate Function . . . . .	39
5.4	Hamming Window Calculation . . . . .	40
5.5	FFT Function . . . . .	41
5.6	Calculate angles of raw FFT coefficients . . . . .	41
5.7	Calculate absolute value of given complex numbers. . . . .	42
5.8	Mean vs Time Graph Implementation . . . . .	44
5.9	FFT Graph Implementation with Stretching Effect . . . . .	45
7.1	Initial bug in OpenCV examples included within the framework . . . . .	56
7.2	Fixing the OpenCV example code bug . . . . .	56
A.1	C++ Test Unit . . . . .	67
A.2	Python Test Unit . . . . .	69
A.3	C++ Test Result . . . . .	71
A.4	Python Test Result . . . . .	71
B.1	Estimating Pulse . . . . .	77

# INTRODUCTION

## 1.1 Overview

Emotions are one important aspect of human life. Emotions define us, allow us to express ourselves, and react to the outside world's events. They are fundamental to the human experience, influencing our everyday tasks such as learning, communicating and rational decision-making. Happiness, anger, excitement, surprise, among other emotions motivate certain actions and enrich human experience [2].

We refer to the system as "Affective Mirror" since it is considered as an agent that interacts with a person to provide feedback at a deeper level of how an individual appears in various situations. For instance, consider the situation of being in the most important interview of your life. The interviewer is watching you and judging your performance. Hence you might practice beforehand in front of an "Affective Mirror". The ability to try out that important conversation in front of your computer, has certain convenience and privacy that brings a sense of comfort. With a camera, microphone and various other sensors, a computer is far more advanced than a normal mirror most people use for practicing. With the right software a computer can be turned into an intelligent mirror with the ability to understand the users facial expression, which can also sense changes in physiological parameters such as heart rate.

With significant achievements and advancements in imaging hardware and software such as Open Computer Vision (OpenCV)[44] the past decade has witnessed a rapid growth in literature pertaining to human face emotion analysis. However humans have very deeply complex emotions. When it comes to human-computer interaction, emotions have the potential to improve these interactions. To further understand emotions, consider an intelligent adult with a full range of emotions who is capable of effectively managing and using these emotions to aid in important decision making scenarios. Adult emotional intelligence consists of the abilities to recognise,

express and have emotions coupled with the ability to control a situation and skilfully handle the emotions of others [3]. When it comes to computers, not all of these affective abilities will be needed at all times, and depending on the context of the application only a subset of the abilities might be required. Adapting certain aspects of human emotions for computers, gives them advantages such as more flexibility and rational decision-making. These abilities could enable a computer to be of more assistance to the user [3].

The aim of this project is to examine how emotions can be incorporated into models of intelligence and in particular, into computers and their interactions with humans. It is entirely possible that a computer with such capability, can outperform humans at recognising emotions.

We will look at emotions from two perspectives. Those that are expressed publicly, or those that are expressed via close contact. Public emotions are communicated through facial expression, vocal inflection, and body language. These are the visible cues. The other form which we have less control, is sensing physiological signals. For instance if somebody holds your hand, they may also feel your pulse racing and sense clammy hands [3]. Hence the more “in contact” we are with a person, the more accurate we can analyse their emotions. Nowadays people have frequent physical contact with computers which places computers in a unique position to effectively take advantage of the readily available personal data to better understand and analyse their users.

### 1.1.1 Terminology

**Emotional state** refers to internal dynamics when an emotion is developed. This includes both aspects of mental and physical states.

**Emotional experience** refers to consciously perceiving our own emotional state. For example right now I am feeling good and happy.

**Emotional expression** relates to what other people perceive by examining the face [3].

**BPM** refers to heart rate which is expressed as beats per minute. We will refer to bpm and pulse interchangeably.

## 1.2 Problem Definition

Although a few emotion detections systems [4] [5] [6] have been developed and used within industry, the focus of their algorithms has been based on the effectiveness of facial expression recognition algorithms. Also the majority of the current emotion recognition applications have also been targeted at market and hence the code is not available to the research community nor

have they exposed their approach and algorithms. While research is gearing towards improving pattern matching algorithms for an effective human facial expression analysis, additional algorithms can be combined to make measures other than analysing the facial expressions. Computers are much more capable than humans at performing many tasks. But when it comes to dealing with human emotions, we tend to draw the boundaries and exclude machines from such capabilities. We rely on humans more to observe, understand and deduce emotions based on our expressions, gestures and the context of situation. However there are measurements that can be sensed by a computer but not visually detected by a human, such as heart rate and temperature. Another obstacle for expression detection in general is the typical challenges present in the field of computer vision. Some of the main key challenges are further elaborated below:

- Main challenges in the field of Computer Vision where large set of data is required to train the system. Furthermore the system could be limited to work at its best under the same conditions and on the set of data that was trained on. This could also include all the lighting conditions in the surrounding environment.
- Training then gets more expensive as the dimensionality<sup>1</sup> of data increases, hence more powerful algorithms with optimisations that enable running on GPU is required.
- Reading the expression on the face is certainly useful but does not capture the complete emotion. Human body has many ways of communicating. Further analysis is enabled by taking into account other measures such as pulse, blood flow, voice and temperature<sup>2</sup>.

In addition to the challenges described above, it is our understanding that current emotion detection systems do not apply any physiological measures, in combination with facial expression analysis algorithms.

The overall goal of this project is to consider the limitations of the human visual system and bring in various technologies which enable computers to analyse emotions by studying the user's face. We will also examine open questions and challenges related to distinguishing emotions when going beyond analysing facial expressions, but to also integrate other means of measures. We highlight recent work from measuring pulse using image analysis, to magnifying video frames. Our aim is to integrate such existing algorithms to enhance the effectiveness of a system detecting emotion by analysing the human face and changes in the heart rate.

---

<sup>1</sup>This concept is explained in chapter 3.

<sup>2</sup>For instance if the user is pressing the keys too hard, the measure of pressure could be a factor to determine their frustration.

## 1.3 Project Objectives

### 1.3.1 Primary Objectives

1. Face detection capability
2. Basic facial Action Units [13] detection, such as smile, raised eyebrow and open mouth AUs.

### 1.3.2 Secondary Objectives

1. Basic emotion detection such as happiness, sadness, surprise, fear, anger, disgust and excitement.
2. Means of measuring physiological measures such as pulse detection capability.
3. Integration of the two technologies.

### 1.3.3 Tertiary Objectives

1. Real-time face analysis
2. Graph Visualisation
3. Evaluation and further testing

## 1.4 Main Contributions

The presented system has been built on top of existing libraries and open-source projects. Inspired by reviewing recent work on Eulerian Video Magnification (EVM) [7], with the initial motivation to integrate this algorithm with the system, the presented challenges called for seeking another approach for detecting the pulse. The challenges introduced by this method are further discussed in Chapter 7.

This work describes, implements and evaluates a novel methodology for recovering the cardiac pulse rate from video frames of the human face, integrated with a face analysis library to detect human emotion. An implementation using a simple webcam with ambient daylight providing illumination is presented. It is not clear what methodology existing Emotion Detection systems use in order to detect emotion since the code is not available. However majority focus on facial expression analysis, including the third party library used for this project. This work

demonstrates the effect of integrating different existing approaches and together yield a more powerful detection. Given the low-cost and ease of use, this technology is promising for future extend, improvements and deployment on various applications across the range of medical and psychological fields.

## **1.5 Software Development Methodology**

### **1.5.1 Source Control Management**

Github<sup>3</sup> was used as means of version control. All the code developed for this project is available from the github repository. However since the project makes use of an external private library, we did not have permission to make the repository public. The code including Kanako code (the modified version that is integrated with this project) has been included within the submission folder.

### **1.5.2 Agile Development Approach**

Agile-driven approach was taken where the focus was on iterative and incremental development. The design of system evolved over time since the first prototype of the system served the purpose of passing a feasibility test. This approach helped with testing methodology as well as re-designing the whole system for a better structure [8].

### **1.5.3 Test-Driven Development**

Unit-testing was performed while porting the Python code to C++. This involved working in short cycles and writing small tests at each stage of re-implementing the algorithm. Dummy-data was fed to both Python and C++ version of the code for comparative analysis of the resulting values<sup>4</sup>.

## **1.6 Ethics**

The preliminary ethics self-assessment form was submitted for this project. The project did not require full ethics as no personal information was gathered, only crowd sourcing the ground truth effectively and no "additional risk" was involved using the heat rate monitors.

---

<sup>3</sup><https://github.com/>

<sup>4</sup>Please refer to the Evaluation chapter for further detail on analysis





# CONTEXT SURVEY

This chapter discusses the background of the topics considered important for this project. It specifically highlights topics related to Human Computer Interaction (HCI) and Affective Computing.

## **2.1 Introduction**

Current research in HCI aims to contribute towards new means of communication with a computer by seeking opportunities to leverage computer devices to study, understand, and positively affect human behaviour. Work has been done to show how computers can be more emotionally intelligent, especially responding to a person's frustration in a way that reduces the negative feeling built up, when the user is interacting with the computer. The computer which we regularly interact with can get to know us and our preferences. A computer with emotional intelligence, will be able to detect how the user is feeling and respond accordingly. However many questions arise when considering how computers might recognise and express emotions. If a computer is trying to recognise or understand our emotion, it should be able to gather information by analysing the face, listening to our voice, noticing gesture, and appraising the situation we are in in order to successfully evaluate our emotion [3].

## **2.2 Affective Computing**

Affective computing looks at techniques which enable us to build computers that can reason about human emotions as well as expressing them. The term "Affective" refers to the reliance of a computer on affective capabilities that already exist in people. Imagine you are having a really bad day, and you jump on the bus and meet the cheerful driver who manages to change

your mood immediately. Or when an enthusiastic friend walks by, his enthusiasm is reflected back in your own interactions. In other words, moods tend to be contagious. Your friend might even try to cheer you up by saying something funny. A computer can do something similar if it can detect the user's sadness and perhaps try to make the user laugh. If a computer knows how its user is feeling, it can be more of assistant and deal with the user's needs. Moods and emotions are also powerful motivators. If you really enjoy interacting with a person, you seek any opportunity to do so [3].

### **2.2.1 Two-Factor Theory of Emotion**

Schachter and Singer [9] developed the two-factor theory of emotions. The two-factor theory suggests that emotion are based from a combination of a state of arousal and a cognition that makes best sense of the situation the person is in [9]. Hence there are two ways of looking at emotions:

1. Emotions are cognitive.
2. Emotions are physical, heart rate is an example.

Research on the cognitive aspect focuses on understand the context within which we derive the arising emotion, for example "That was an important goal that I attained, hence I am happy" [3].

### **2.2.2 Physical vs Cognitive**

The development of affective computing require understanding of both physical and cognitive components of emotion. In fact emotion theorists still do not agree even on a definition of emotion and facial expression theories [3]. Hence further complication arises from mapping and relating between the two.

### **2.2.3 Sentic Modulation**

Sentic modulation (septic comes from the Latin sentire, the root of the words sentiment and sensation), such as voice inflection, facial expression, and posture is the physical means by which an emotional state is typically expressed [10]. Expressing emotional state through sentic modulation is natural and usually subconscious. Today we know that both brain and body interact in the generation of emotion and its experience. Thoughts lead to emotions and also emotions can occur without any cognitive evaluation such as reactions to changes in bodily chemistry [3].

### 2.2.4 Facial Expression

Facial expression is the most widely acknowledged forms of septic modulation. Interest in facial expression has a rich history dating back to 1960s. The pioneering work in the field of "emotion recognition" was conducted by a team of scientists led by the american psychologist Paul Ekman, the foremost authority on facial expressions. Ekman followed up the theory proposed by Charles Darwin. Darwin believed people worldwide must manifest emotions the same way. Hence he deduced that human emotions are universally recognised and easily interpreted through facial expressions regardless of culture or language [11]. Ekman travelled around the world, studying people's emotions among different cultures, and the expressions always matched. Ekman called it "basic emotion model" and suggested that this model could be used to even distinguish a liar. His basic model maps emotions to a unique set of facial muscle movement patterns [3]. Their system, known as the "Facial Action Coding System" (FACS) maps between muscles and an emotion space, and was developed based on their theory of this basic model [12]. FACS are used as an index of facial expressions, and Action Units (AUs) are the fundamental actions of individual muscles or groups of muscles [13]. Ekman suggested that by distinguishing between actual expression and microexpressions, it is possible to expose a deceiver attempting to mimic without having actual feelings behind them. Ekman consulted the CIA to help interrogators in the 1970s. Ekman also deduced that humans have 6 facial expressions: happiness, anger, sadness, fear, surprise and disgust. In Ekman's study and various other research studies where the study was replicated in order to prove correct, research subjects were asked to look at photographs of facial expressions such as smiling, scowling and match them to a limited set of emotions or to stories with phrases like, "Her husband recently died". Over the following decades, this method of studying emotions has been replicated and has been widely accepted since debut 1969. However recent studies flags up flaws with this model [14]. There is currently debate among psychologist that emotions are not universally recognised as proposed by Charles Darwin, and that facial movements might be evolved behaviours for expressing emotion. New research published in the journal *Current Biology* by scientist at the University of Glasgow have proposed only four basic emotions [15]. They believe Action Units involved in signalling different emotions as well as time-frame over which muscle was activated, are common between fear and surprise. In both fear and surprise, we tend to widen the eyes to gather more visualise information to assess situation and they believe that everything else is a consequence of cultural evolution.

In another study published in the journal *Emotion* [16], subjects were not given any clues. Instead they were asked to freely describe the emotion on a face. In this study subject's performance plummeted. In particular they argue that providing subjects with a preselected set of emotion words, leads to biased result where answers are hinted and hence skewing the results.

Can we detect someone's emotional state by just looking at his face? We often "read" what someone is feeling with a quick glance. Scientific studies support the idea that the face is signalling the full array of human sentiments, from fear and anger to joy and surprise. If faces do not "speak for themselves", then the question is how to read other people or even develop algorithms to do so. Face is not telling the whole story. This has proven to be harder than one might expect. We do not passively recognise emotions but actively perceive them, relying on a wide variety of contextual clues such as vocalisation, body gesture and eye movements[17]. What is it about our face, our voice and other mannerisms that communicate our feelings to the outside world? Are physiological measurements such as pulse and skin conductivity, sources of reliable and universal indicators of emotion? [3] There are many questions that arises when studying human emotions. There is an ongoing research and this project aims to open up new opportunities for more advanced means of research with the aid of computers.

### **2.2.5 Social Signal Processing**

Humans label perceiving, learning and adapting to the world as intelligent behaviour. There is now growing interest in cognitive science which is an interdisciplinary study of cognition and neural mechanisms that give rise to biological intelligence and behaviour [18].

Social intelligence is an aspect of human intelligence which has been considered as the most important in order to succeed in life. But how do we measure success? Is IQ a good measure of human intelligence? The research is growing towards various fields such as affective, cognitive, linguistic, perceptual and social systems.

Social intelligence is the ability to express and recognise social signals and behaviours such as agreement, politeness and empathy coupled with knowledge of interaction styles and strategies that help a person achieve their objectives [19] [20]. Social signals and social behaviours are the expression of ones awareness of situation and the social dynamics that govern them. These expressions manifest themselves though non-verbal behavioural cues such as facial expressions, body postures and gestures, and vocal outburst like laughter [20]. When it comes to computers, however, they are socially ignorant [21]. Not all computer applications will need social intelligence. The current computational power works well enough for context-independent tasks such as performing computation-intensive tasks or making plane reservations. However such means of computing is inappropriate for applications that require deeper interactions with humans. Understanding emotions can alter the way we interact with machines - such capability can open up boundless opportunities.

Visionaries such as Mark Weiser predicted a future of embedded sensors inside everything,

making computers so exciting, so wonderful and interesting that we never want to live without them [22]. "Ubiquitous Computing is fundamentally characterised by the connection of things in the world with computation" (M. Weiser). Its highest focus is to make a far-reaching vision of computation embedded in real world. At a very high level, it aims to make a computer invisible by making it embedded and so naturally fit with anything we interact with, on a daily basis [22]. Mundane everyday artefacts could become augmented as soft media, able to enter into a dynamic digital world [23]. The closer we get to realising a yet more efficient and comfortable lifestyle, one would expect for these computers to learn our routines, preferences and assist us in many possible ways.

Most of current pervasive systems facilitate various aspects of research including sensor technology, software technology, artificial intelligence and human-computer interaction. Researchers working in artificial intelligence and context-aware systems study sensor data to design algorithms for activity recognition, providing a higher level of support. Majority of these technologies have the common goal of improving the quality of end user experience. They aim to make computers more personal device, tailored to ever-changing interests of user. The long-term goal is to enable computers to sense agreement, inattention or frustration and be able to adopt and respond to these social signals, in the most appropriate manner [20]. The study of social interactions and developing automated assessing of human social behaviour, will result in valuable systems that could revolutionise basic research in cognitive and social sciences [20].

### **2.2.6 Honest Signals**

Honest signals emerges from a new field called "Network Science" [24], which instead of understanding people individually, studies them in the context of their social networks. Honest signals refer to the subtle patterns in how an individual interacts with other people. Such patterns can communicate the person's attitude, for instance one can detect when someone is bluffing, paying attention or genuinely interested. The analysis of honest signals gives us the power to predict outcomes of situations whether it's a job interview or a first date [25].

### **2.2.7 Contagious Mood Signals**

Studies on twitter and Facebook has recently revealed how quickly emotions such as pride, happiness, disappointment can spread throughout the online community. A new study<sup>1</sup> shows that emotions on social networks can be contagious. Social activities such as status updates, emotions expressed by a person can cause emotional changes in another person [26]. If one's

---

<sup>1</sup>This study examined status updates posted by 100 million users in the 100 most popular U.S. cities between 2009 and 2012.

emotions or moods can be altered through interactions via social media, we may wonder whether such capability can be applied to a personal computer.

### **2.2.8 Emotion Recognition & Emotional Response**

The implication of teaching a computer to understand feelings, is that we are explicitly defining the emotions, by telling the computer what an angry face looks like. Even humans often fail to recognise the true underlying emotion. For instance a human might not be able to distinguish the difference between contempt and anger. Instead we do this by learning. A personalised system where the system knows its user and it does the detection customised for that user only, could prove to be more effective. In this case, the system can be trained on the user. Understanding emotions is only one side of the coin, knowing how to show sympathy is another challenge specially for a computer. An important question is how well can we generalise the emotions detected. Because if we are only looking at the facial expression through the eyes of a computer (camera) how can we accurately say how the user is feeling? The good news is that computers are capable of analysing many other means of data such as temporal, blood pressure and skin conductivity. This can be considered as an advantages that computers have over an expert psychologist in the field of human emotion analysis.

## **2.3 Techniques for Facial Expression & Emotion Analysis**

### **2.3.1 Overview**

Majority of existing systems use Paul Ekman's model as the basis of facial expression detection<sup>2</sup>. Current research's focus has been on improving the existing machine learning algorithms which lead to more effective training. A well trained system is expected to perform well, given a new set of data.

### **2.3.2 Existing Tools & Libraries**

Current tools and existing systems have been discussed in this section. Some of these tools were studied as a use case for this project as well as the potential feasibility of integration with our system.

---

<sup>2</sup>Including the third party library used for this project.

### 2.3.2.1 Social Data Analysis

The most basic means of analysing user's emotion is by collection of user-related data. Typically approaches such as self-report strategy or collecting surveys are too restrictive. However a more automated approach, could consist of monitoring user based on their location and habits. Ongoing research is being carried out in order to make inferences about people's well-being and behaviour based on smartphone data collection and feedback mechanism to be delivered to the interface [27].

### 2.3.2.2 iMotions

iMotions[4] is a company that produces biometric research software platform. They offer a FACET (Facial Action Coding System) SDK which provides the ability to track and analyse the emotional responses of users in real-time, detecting and tracking eight facial expressions of primary emotions. Initially we made a query regarding gaining access to their SDK, however due to high academic pricing<sup>3</sup>, this option was abandoned.

### 2.3.2.3 SHORE

Sophisticated High-speed Object Recognition Engine (SHORE)[28] offers a basic C++ API with features such as :

- Position of the face, the eyes, nose and mouth
- Information whether the eyes or the mouth are open or closed
- Gender classification
- Recognition of facial expressions ("Happy", "Surprised", "Angry" and "Sad").

We also approached their team since they do collaborations with academia, however they were unable to offer us license to use their software.

### 2.3.2.4 Emotient

Emotient [5] is formerly known as Machine Perception, makers of CERT<sup>4</sup>. This startup firm has recently received additional \$6 million in funding to further develop its facial-recognition emotion-sensing technology. As with other companies, their aim is help retailers better understand how customers feel about products while they are shopping. They have also

---

<sup>3</sup>\$4900 USD for a short limited time.

<sup>4</sup>Computer Expression Recognition Toolbox

announced the development of a Google Glass glassware app, designed to perform on-the-spot sentiment analysis [29].

### **2.3.2.5 Vocal Intonation**

Vocal intonation is the second widely acknowledged form of sentic modulation. Interestingly emotions in speech can be understood by young children before they can understand what is being said [30].

Most emphasis on interaction with computers via speech has focused on teaching computers to understand what is said. More recently, researchers have focused on teaching the computer to recognise who is speaking [3]. Furthermore, vocal inflection is also important in applications where people and computers depend upon the use of synthetic voices [3]. One example of an existing project is "openEAR" [31] which is an Open-Source Emotion and Affect Recognition Toolkit. This library provides audio feature extraction algorithms and classifiers. Another existing library is "openSMILE" [32] which performs audio, speech and music feature extraction.

### **2.3.2.6 Physiological Measures**

There are many physiological responses that can be combined to assist in recognition of emotional states. These include heart rate, diastolic and systolic blood pressure, pulse, pupillary dilation, respiration, skin conductance and colour, as well as temperature[3]. However physiological influences of emotions have many implications. For instance sweaty palms and a rapid heart could indicate fear or perhaps it's a sign of arousal. Our brain must appraise the situation we are in before it can label the state with an emotion [9]. This task is however more difficult for a computer which would require further information about the environment and the context of the current state of its user.

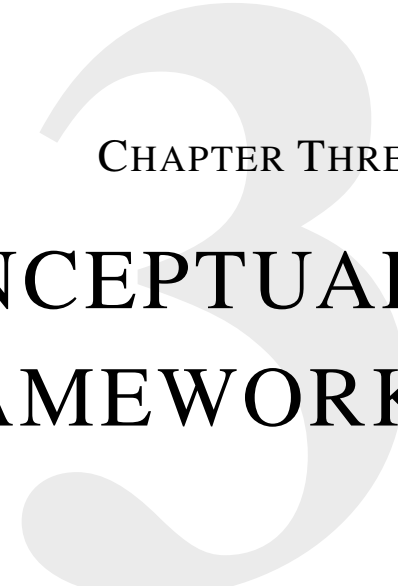
## **2.4 Summary**

This chapter has provided basic knowledge about human emotion theory, emphasising on the two aspects of emotions such as physical and cognitive. We highlighted that affective systems equipped with camera, microphones, physiological sensors and sophisticated facial expression recognition could begin to recognise physiological components of emotion. As computer devices become more embedded and more powerful, further capabilities can be realised through implementation of sophisticated algorithms which can make use of such computational power. Additionally, a computer can observe and learn human behaviour, and analyse their emotion by not only studying the user but also analysing the current situation. Once these capabilities



are in place, a thorough cognitive reasoning can be derived which can reason about the present emotions [3].





CHAPTER THREE

# CONCEPTUAL FRAMEWORK

This chapter presents a detailed discussion of the conceptual underpinnings that form the system as well as strategies used to design the system. The employed strategy consists of three distinct technologies. These three technologies are explained in detail with an emphasis on their interaction with each other within the overall conceptual framework.

## **3.1 Facial Expression Analysis**

### **3.1.1 Kanako Library Overview**

As argued in Chapter 2, existing libraries that perform facial expression analysis are not readily available due to two reasons. Firstly, there are currently only a few existing tools, and the existing ones are not publicly shared by companies and hence not free to use. Therefore, an underlying objective of this project was to devise a basic facial expression detection library using OpenCV Machine Learning library. However this proved to be quite challenging, since the focus of this project has been mainly on the integration of existing algorithms. Algorithms and APIs offered by OpenCV are quite powerful however in terms of effectiveness and optimisation, they will need further support. Hence we decided to turn to academia to gain access to existing libraries performing basic facial expression analysis. “Kanako” is a private face analysis library, used for this project. Kanako is still under development and hence it’s not quite a fully implemented API yet. However it does have a fairly high capability of fast simultaneous detection of multiple AUs in real time. Although the algorithms still need improvement, the accuracy and capability of this library does serve the purpose of this project. Also modifications to the source code of Kanako was required in order to integrate it with our system using an appropriate design.

### 3.1.1.1 Kanako Components

- lib : contains the source code as well as Unix Makefile.
- models : Contains a number of SVM models for different FACS Action Units and Paul Ekman's Basic Emotions along with face and eye detector models.
- tools :
  - training : Intended to train AU models to be used with the library. However we were not given access to train the system and the author of code gave us the pre-trained data where the training was performed on a large face database.
  - realTime tool : Intended to detect AUs in real time using frames captured from a camera. The application displays SVM decision values at the terminal window and estimated intensity.
  - video tool : Intended to detect AUs offline using frames captured from a video feed.
- scripts : Running scripts used to set up all required environmental variables like DYLIB\_LIBRARY\_PATH where applicable and set all of the input arguments.

### 3.1.1.2 Machine Learning Concepts

In order to briefly explain the algorithms used by Kanako, we will clarify the underlying general concept. This section aims to give machine learning a practical definition. We first define machine learning as obtaining knowledge through experience<sup>1</sup> or receiving initial training by an external source. This implies purposeful learning since learning without purpose is not really training. In the context of computers, we define learning as recognising structure and patterns from data. Machine learning techniques have been adopted due to their high level of performance<sup>2</sup>[33]. In this context, by performance we refer to the accuracy of the underlying algorithms. The learning process aims to change the behaviour of the system in a way that improves performance in the future.

### 3.1.1.3 Algorithms

There are two main types of learning : Supervised vs unsupervised. From a theoretical point of view, the two approaches differ in how training is performed on the model that represents

---

<sup>1</sup>Where gathering experience might occur from actual situations the algorithm encounters.

<sup>2</sup>Machine learning techniques have led to many successful applications such as Bioinformatics, Image Recognition, Natural Language Processing and Search Engines.

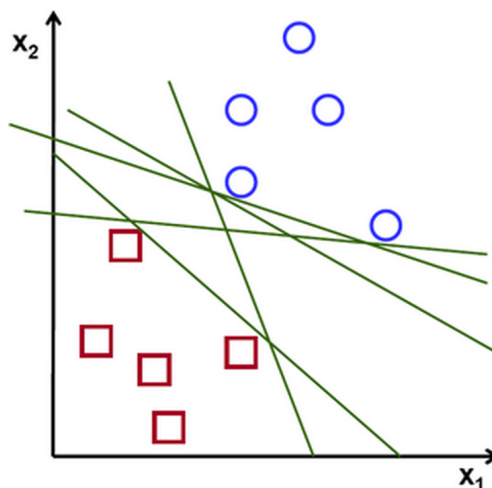
the problem. With unsupervised learning, the correct results are not given to the model during training<sup>3</sup>. Whereas in supervised learning, training data includes both the input (i.e. set of labelled training data) and the desired results. In this case, the correct results are the target and are given as input to the model during the learning process[34].

Support Vector Machine (SVM)[1] is a class of supervised learning models which perform classification by constructing a  $N$ -dimensional hyperplane which separates the data into categories. The algorithm outputs an optimal hyperplane which categorises new examples.

To better understand the algorithm, let's consider the following simplified problem, dealing with lines and points in the Cartesian plane:

- For a linearly separable set of 2D-points which belong to one of two classes, find a separating straight line.

**Figure 3.1:** Two set of 2D-points belonging to two different classes [1].



However, typically we are dealing with hyperplanes and vectors in high dimensional space<sup>4</sup>. Figure 3.1 shows multiple lines offering solution to the problem. The operation of the SVM algorithms is then based on finding the line which distinguishes the categories the best [1].

<sup>3</sup>There are two approaches to unsupervised learning. The first approach is to use a rewarding system to indicate success rather than explicitly stating the categorisations. A second type is clustering where the goal is to find similarities in the training data [34].

<sup>4</sup>The dimensionality can be very large and this also makes it quite hard to be presented explicitly in the memory.

### 3.1.1.4 Kanako Algorithms & Optimisations

Kanako uses a Radial Basis Function kernel (RBF kernel)[35] which is a popular kernel function used in SVM classification<sup>5</sup>. Kanako uses a third party library, GTSVM (A GPU-Tailored Approach for Training Kernelized SVMs)[36]. This library is used for an optimal and efficient training using SVM on a Graphics Processing Units(GPU)<sup>6</sup>.

Given the fragment of video frame (i.e. the region of the detected face), for every block of 5 frames, the algorithm calculates a prediction for AUs using SVM. These predictions are distances from hyperplane, hence they tend to get larger when a particular AU is active. The API returns intensity for every AU. The rationale behind using intensities is that they are more meaningful since they can determine how active a certain AU is as well as determining intermediate stages. For instance when smile is detected, the particular AU corresponding to smile will increase. If neutral all AUs will be small and negative, so ideally when AU is active, corresponding value should be positive. Essentially there must always be a clear distinction between active and inactive stage. Although currently it performs a naïve computation of intensities by using the prediction values and fitting them into a range from 0 ... 1. It treats all predictions below -0.5 as 0 and all predictions above 0.5 as 1.

The library provides two sets of APIs. One API returning AUs corresponding to basic emotions such as happiness, surprise, sadness, anger, disgust and fear. The second API for high level emotions is still under development. Thus after performing evaluation of system, the former proved to be more effective.

### 3.1.1.5 Kanako Issues & Limitations

- In terms of hardware the library ideally demands a powerful machine with a quad-core processor a GPU with high computational capability<sup>7</sup>.
- The GTSVM library used does not support linear kernel which is more ideal for facial expression recognition. Hence it is currently limited by the RBF kernel SVM.
- Intensity estimation is rather naïve.

<sup>5</sup>Kernel functions enable operating in a high-dimensional data space. Other functions operating in high dimensions are polynomial and sigmoid.

<sup>6</sup>This is a CUDA based SVM [37] .

<sup>7</sup>As the number of AUs increases GPU allows to keep classification time low whereas CPU requires exponentially more time. Even though its faster, a typical modern CPU has only 4-6 cores and up to 16 for expensive CPUs whereas relatively cheap desktop GPU has about 1500 cores [38].

- Since the system is still under developments, a few major bugs were discovered while using the library.
- Lastly, not all basic emotions are detected as expected. In particular, happiness, sadness and surprise work the best but anger, fear and disgust are harder to detect. Hence the algorithms need further improvements and more training is required.

## 3.2 Pulse Detection

The algorithm is divided into two steps, face detection and pulse estimation.

### 3.2.1 Face & Forehead Detection

The adopted algorithm uses the OpenCV API to detect faces by performing automatic facial tracking. The system implements this in a way that the frames are passed to both components, Kanako and the pulse detection component. Essentially they both share the same stream of captured frames and apply their own individual algorithms on these frames. Face detection has the purpose of detecting the location of the face within the whole frame. Once we detect the face we then send the rectangular portion containing only the face to the pulse detector algorithm. Note that the whole frame is passed to Kanako since it performs its own face and eye detection to analyse the expression. The pulse detector algorithm then calculates the position of forehead. It then isolates the forehead region to which it applies a so-called image Photoplethysmography (PPG) [39] algorithm. The PPG process forms the basis for pulse detection.

### 3.2.2 PPG Overview

Photoplethysmography is a novel video-based methodology for non-contact, automated cardiac pulse measurements. PPG was first described in the 1930s [39], as a simple low-cost optical technique to measure the blood volume changes. Detection of the cardiovascular pulse waves traveling through the body is referred to as Plethysmographic<sup>8</sup> [40]. This method has the ability to sense the cardiovascular pulse wave i.e. blood volume pulse through variations in transmitted or reflected surrounding light [41]. This webcam-based technique uses normal ambient light as a source of illumination for remote acquisition of PPG signals and a simple digital camera. This method takes into account the volumetric changes in the facial blood vessels during the cardiac cycle which modify the path length of the incident ambient light in a way that the subsequent changes in amount of reflected light indicate the timing of cardiovascular events i.e. the pulse

---

<sup>8</sup>Plethysmos means increase in Greek.

[41]. The RGB colour channel contains a mixture of the reflected plethysmographic signal. Since the hemoglobin absorptivity differs across the visible and near-infrared spectral range [42], each colour sensor records a mixture of the original source signals with slightly different weights. The observed signals from the red, green and blue channels are amplitudes of these signals (i.e. averages of all pixels in the forehead region) [41].

### **3.2.2.1 RGB channels vs Green Channel**

It has been reported that the green component features the strongest plethysmographic signal than the red and blue channels [41]. Although red and blue channels also contain plethysmography information, they tend to be more noisy. While it is unclear why this is the case, one might expect it to be related to the green channel and the fact that hemoglobin absorptivity is highest in green/yellow light [41]. However this may not necessarily be always the case. Hence this theory has been tested in the evaluation section, where we tested the effectiveness of algorithm, by averaging RGB channels vs green channel.

### **3.2.2.2 Limitations**

This method is prone to motion-induced signal corruption. Another challenge this method presents is the effectiveness when under the influence of variations in ambient light intensity which can affect the accuracy of physiological assessment. The success of the algorithm relies on good lighting and minimal noise due to motion and under these conditions, a stable heartbeat should be detectable in about 15 seconds.

## **3.3 Emotion Visualisation**

The third component brings the results of the analysis of the two components, and provides intuitive data visualisation. This component also contains basic logic to define mapping from cognitive to physiological analysis of the face. That is for instance, detecting happiness and a high heart rate at the same time, indicates excitement.

### **3.3.1 Data Communication**

A good design must enable ease of data communication within the system. i.e. data flow. The system has been designed in such a manner that each individual components perform their task and make the data available for the visualisation component.



### 3.3.2 Visualised Emotions

The emotion bars indicate the intensity of AUs detected. High intensity results in the green bar increasing. If the pulse detection is also enabled, the resulting *bpm* (beats per minute) is analysed with AUs such as happiness and fear. Note that further research is required to perform an effective mapping from cognitive to physiological signals. Currently the system makes basic assumptions, such as if the person is happy and the *bpm* is high, the person is excited. However high pulse in conjunction with fear could also indicate anxiety. A high pulse rate may also be due to increased physical activity, which is the case in our experiments since we ask the participants to perform simple exercises in order to monitor bpm variations. Hence these mappings must be done by careful analysis of AUs as well as considering the context. Since under normal circumstances, we expect a stable heart rate from a user sitting in front of the computer, we make the assumption that a high *bpm* is related to the individual's emotions.

### 3.3.3 Pulse Graphs

The pulse graph is a separate *graph window* from the emotions focusing only on the *bpms*. Three graphs are drawn :

- Means
- FFTs<sup>9</sup>
- BPMs

---

<sup>9</sup>Pruned to filter out occurring lows and highs caused by the noise



# SYSTEM SPECIFICATION & ARCHITECTURE

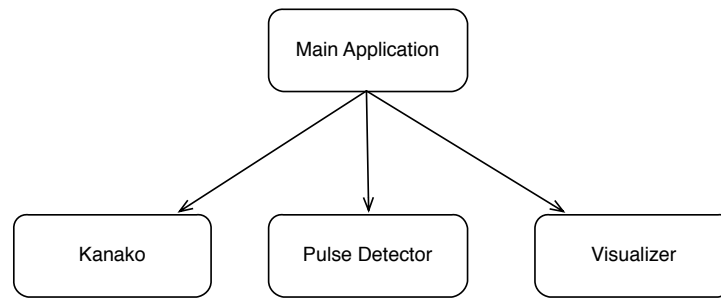
This chapter presents high level view of the architecture adopted by the system and the core requirements of the system.

## 4.1 User Requirements

- The ability to identify basic emotions of the user through a video stream captured through a standard webcam. The list of basic emotions are: happiness, sadness, surprise, fear, anger and disgust.
- Ability to detect users heart rate through the video stream captured from a standard webcam under good light conditions.
- Forms of visualisation to inform the user of the detected emotion.

## 4.2 Architecture & High Level Design

The high-level architectural view of the system is shown in Figure 4.1 which illustrates the main components of the application.

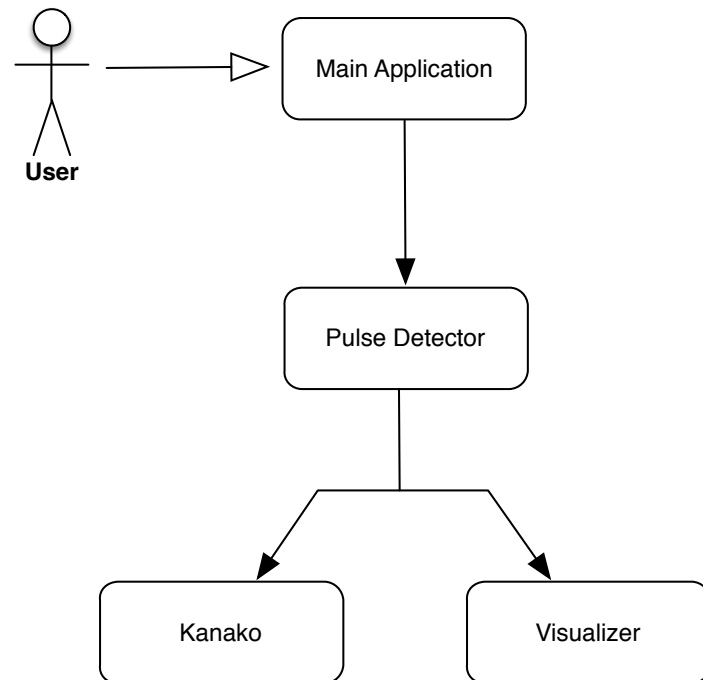


**Figure 4.1:** Basic architecture overview.

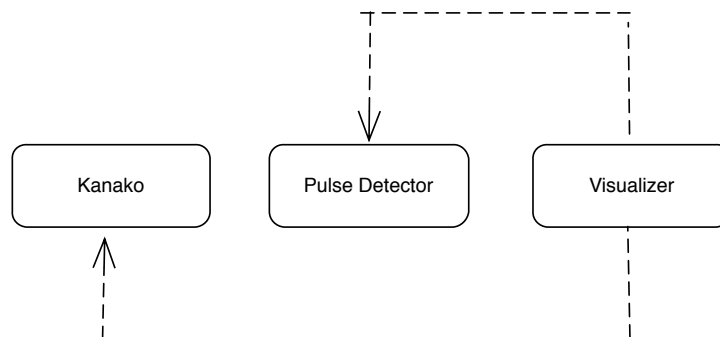
Figure 4.2 illustrates the interactions between the user and the system, and among the components of the system. The main application runs the pulse detector component, which performs the following logic :

- Using OpenCV API, connect to the webcam and start reading video frames.
- Initialise Kanako, make a copy of the video frame and pass it on to Kanako.
- Initialise the visualiser component.
- Initialise the graph window display.
- Perform face detection.
- Run the pulse detection algorithm on the detected face.
- Show the resulting graph.

Figure 4.3 further illustrates how the visualizer component uses the data from the other two components to perform the visualisation.



**Figure 4.2:** Use-case diagram displaying interaction with the user and components.



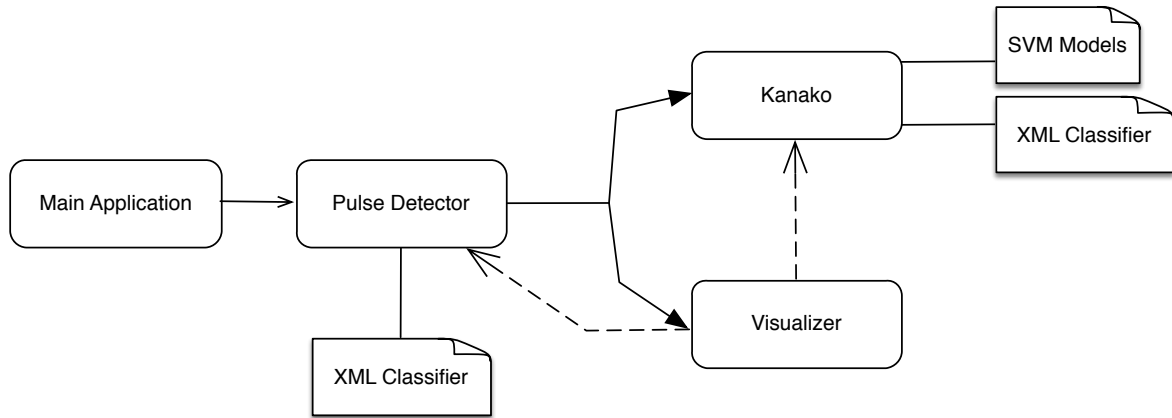
**Figure 4.3:** Data flow between the components.

## 4.3 Data Model

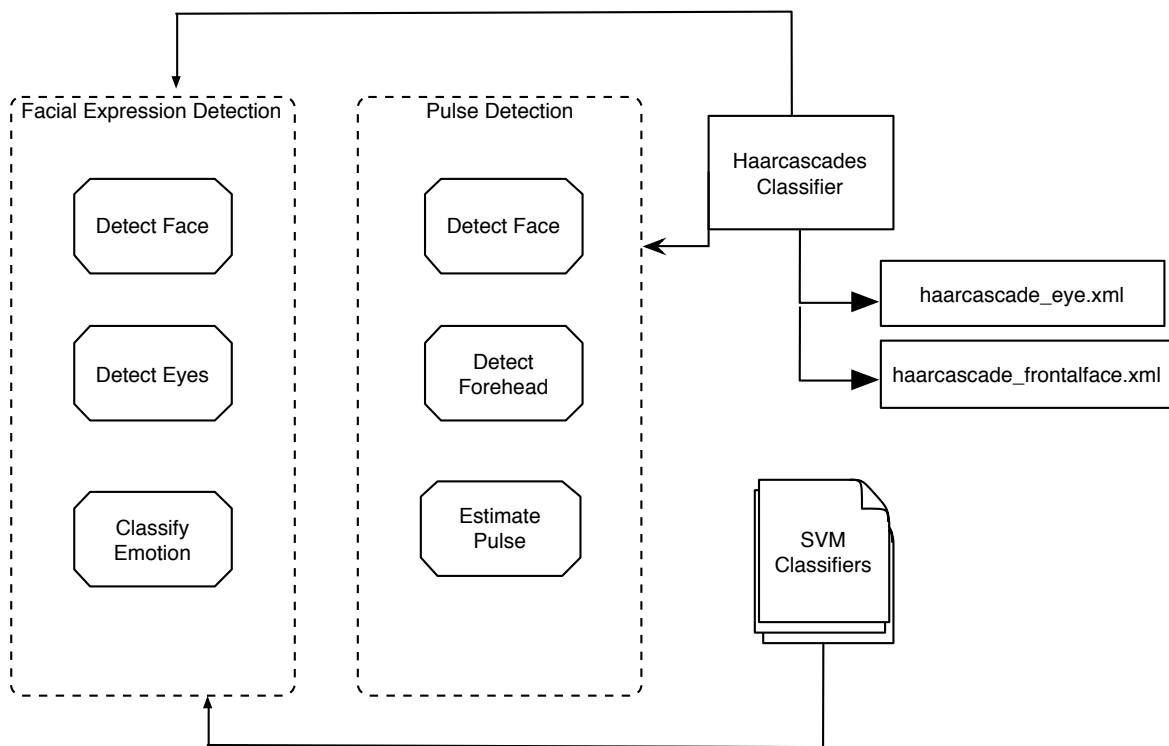
The two components, including Kanako and pulse detector need pre-trained data such as classifiers and SVM models. The haarcascades classifiers obtained from OpenCV [43] are common to both components. In addition, Kanako also uses its own internal trained models to analyse high level emotions<sup>1</sup>. Further component interaction and data loading is illustrated in Figure 4.4. Note that in this diagram, the dotted arrows indicate the component *uses* the resulting

<sup>1</sup>Note that the XML face classifiers used for both components are identical.

calculated data from the the other two components.



**Figure 4.4:** Component interaction and trained classifiers requirements.



**Figure 4.5:** Top-level component decomposition view of the Affective Mirror system along with interactions among the components.

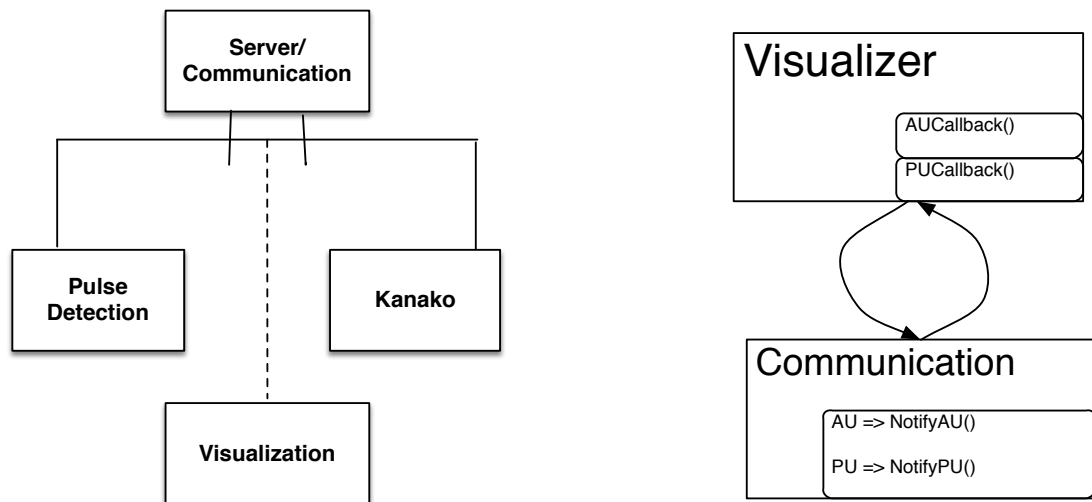


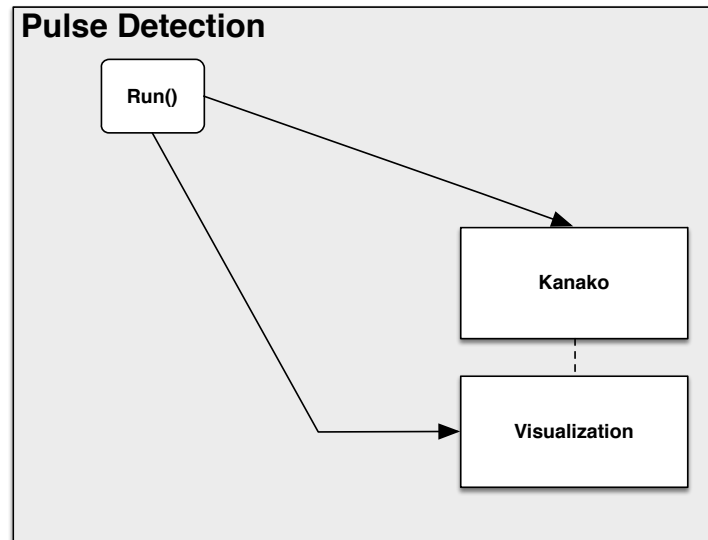
Figure 4.6: Initial design of system before applying the complete integration.

## 4.4 System Integration

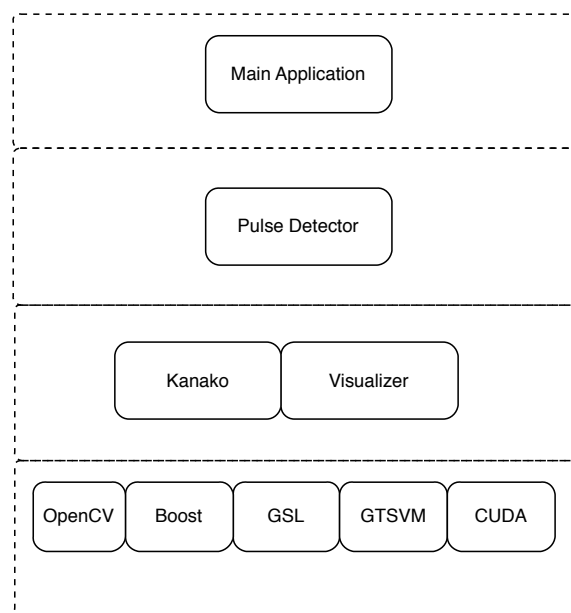
Figure 4.6 illustrates the initial architecture of the system. In this design, the components are loosely-coupled, communicating via the server. This design provides an illusion of integration, even though the components were completely separate. An improvement to this is to perform a tight integration which results in a stand-alone application.

### 4.4.1 Integration with Kanako

Designing a stand-alone application, has many advantage as the main component will have control over the rest of the system components. A tight-integration architecture was chosen for this application, since ideally we want our system to be in charge. This gives power to the main system which in turn uses sub-systems and libraries to perform the required task. Figure 4.8 attempts to illustrate the system as a layered architecture where the main application starts an instance of pulse detector. The pulse detector then uses the libraries from the layers below, such as Kanako and OpenCV to first, make a call to Kanako to start the expression analysis task, and perform its own logic to detect the pulse.



**Figure 4.7:** Tight integration with Kanako.

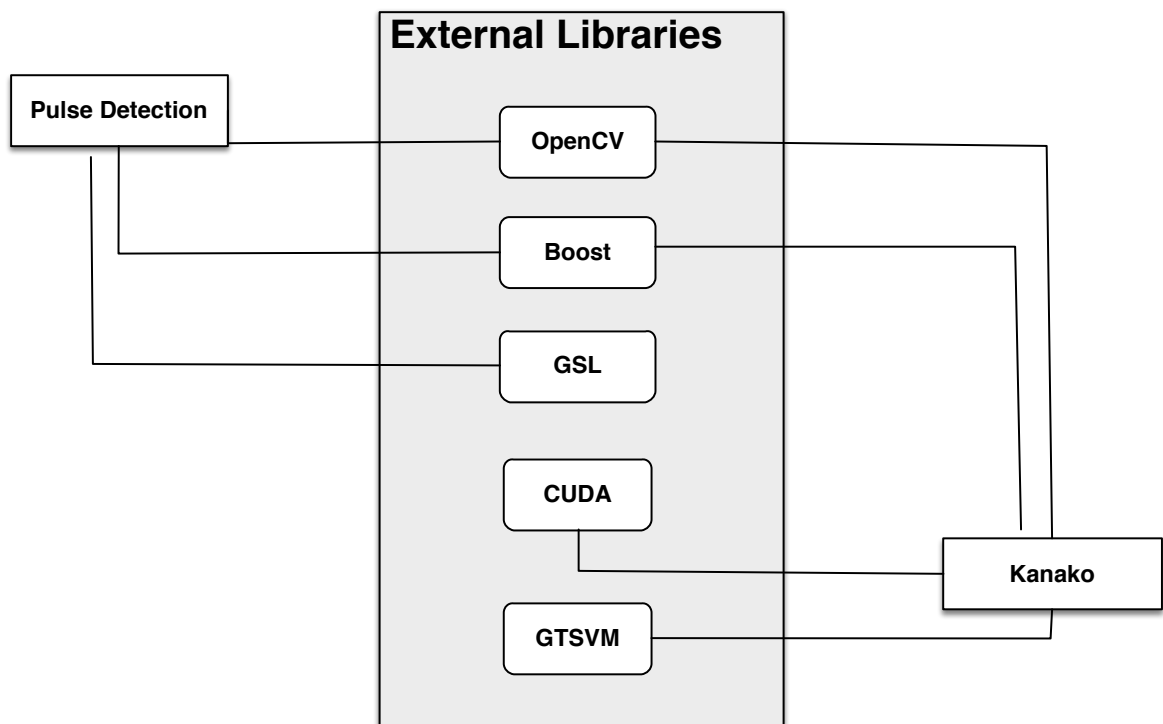


**Figure 4.8:** Hierarchy layer architecture of system.

#### 4.4.2 External Library Integration

Finally, Figure 4.9 shows how the interactions of the main two components with the external libraries.





**Figure 4.9:** External libraries used by the two main components of Affective Mirror.



# IMPLEMENTATION

This chapter presents approaches employed for implementing Affective Mirror that follow the design discussed in the previous chapter. The chapter also highlights interesting aspects of the code and challenges encountered in the implementation.

## 5.1 External Dependencies

### 5.1.1 OpenCV

OpenCV [44] was chosen for this project since its a de-factor library for building computer vision applications and related research. OpenCV is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. OpenCV offers C,C++ and Python APIs. This projects makes use of the C++ API, in order to enable the process of integration with Kanako. Required headers and libraries for OS X are precompiled and included within the project.

### 5.1.2 Boost

Boost [45] is a set of C++ libraries that provide support for linear algebra, multithreading, image processing and more. The pre-compiled boost libraries (version 1.54) have also been included.

### 5.1.3 GSL

GNU Scientific Library (GSL) [46] must be installed since its a prerequisite for the pulse detection component. GSL is a numerical library for C and C++. This library was used as equivalent of SciPy and numpy function to re-implement the algorithm which was originally developed for python.

### 5.1.4 CUDA 5.0 or above

This is the requirement for Kanako. With CUDA installed, and the third-party libraries included, installing/compiling OpenCV and Boost will not be required since the pre-compiled header files have been included. Compiler NVCC 5.5 with GCC 4.6.4 (Visual Studio 2010 compiler for Windows) as a host compiler for C/C++ code are recommended <sup>1</sup>.

## 5.2 Face Detection

This project implements face detection using OpenCV APIs. OpenCV provides a large collection of useful functionality such as classifiers to detect objects in video stream. The OpenCV face detection algorithm is based on work by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" [47]. This is a machine learning based approach where a cascade of boosted classifier, using 14 Haar-like digital image features, are trained with a lot of positive and negative examples. The word "cascade" in the classifier name means there are several stages that are applied subsequently to a Region of Interest (ROI) unit. At some stage either the candidate is rejected or all the stages are passed. Internally it applies a support vector machine (SVM) classifier to each image patch [48]. These pre-trained Haar Feature-based Cascade classifiers <sup>2</sup> [43] are available with OpenCV 2.4+. Specifically, the frontal face classifier was used to detect the face.

### 5.2.1 *detectMultiScale* API

The API detects objects of different sizes in the input video frame. It then populates a vector of `cv::Rect`, where `cv::Rect` is just a data structure which stores coordinates of a rectangle. If faces are found, these coordinates are the positions of detected faces, returned as `Rect(x,y,w,h)`.

Code snippet to read camera frames and detect face is shown in Listing 5.1.

---

```
1
2 void PulseDetector::run() {
3     cv::CascadeClassifier faceClassifier;
4     if (!faceClassifier.load("haarcascade_frontalface_alt.xml")) {
5         cerr << "Unable to load face classifier.\n";
6         exit(1);
7     }
8     // Open camera device
```

---

<sup>1</sup>For this project CUDA 5.0 was used, however upgraded versions should also work fine.

<sup>2</sup>The classifiers are in XML format

```

9   cv::VideoCapture camera;
10  if (!camera.open(0)) {
11      cerr << "Unable to initialise camera.\n";
12      exit(1);
13  }
14  while (processing) {
15      // Read frame
16      cv::Mat frameOriginal, frameGreyscale;
17      camera.read(frameOriginal);
18      if (!frameOriginal.empty()) {
19          cv::Scalar color(255, 255, 0, 0);
20          // Convert image to gray scale and equalize
21          cv::flip(frameOriginal, frameOriginal, 1); // Mirror effect
22          cv::cvtColor(frameOriginal, frameGreyscale, CV_BGR2GRAY);
23          cv::equalizeHist(frameGreyscale, frameGreyscale);
24          // Detect faces
25          vector<cv::Rect> faces;
26          faceClassifier.detectMultiScale(frameGreyscale, faces, 1.1, 3,
27              CV_HAAR_FIND_BIGGEST_OBJECT | CV_HAAR_SCALE_IMAGE,
28              cv::Size(30, 30));
29          if (!faces.empty()) {
30              // Do something...
31          }
32      }

```

---

**Listing 5.1:** Face Detection

Here we set the `CV_HAAR_FIND_BIGGEST_OBJECT` flag to enable tracking the most prominent face in the image [48]. Note that parameters need to be configured to reduce detection of false positives.

We also set the following parameters :

- `scaleFactor` - Parameter specifying how much the image size is reduced at each image scale. OpenCV uses this factor to do multiple passes over the image to scan for feature-hints. So increasing the scale factor might give faster results, but might fail to find a face. 1.1 is a good value for this since it uses a small step for resizing.
- `minNeighbours` - Parameter specifying how many neighbours each candidate rectangle should have to retain it. This parameter determines how much “neighbourhood” is required to pass it as a face rectangle. For instance if it is in neighbourhood of other rectangles, then it’s ok, pass it further. Hence the parameter will affect the quality of the detected faces. Higher value results in less detections but with higher quality. 3-6 is a typical good value for.
- `minSize` - Minimum possible object size. Objects smaller than that are ignored. This parameter determine how small size we want to detect. Usually, [30, 30] is a good start for

face detections.

### 5.2.2 Isolating the Forehead Region

The forehead region is required for the pulse detection algorithm. The algorithm needs a single patch of plain skin on the individual. This region ideally should be easy to track with a consistent shape across individuals. Hence the forehead region could be considered best for this. Also according to the author of the original code, cheeks actually seem to have better SNR<sup>3</sup> with respect to estimating a heart rate<sup>4</sup>, but they vary from person to person, hence they are hard to isolate.

The simplest approach to isolate the forehead, is to estimate which region of the face the forehead will be i.e. by simply calculating the region. In the original implementation, the forehead is locked since the algorithm operates by averaging over time, hence needs a minimum for 15 seconds to stabilise in order to estimate the correct result.

Initially we thought this was inconvenient since if the individual moves slightly, the analysed frames in the region will be inaccurate. It's possible to keep tracking the forehead as the subject moves. This way, the detected forehead region also moves rather than being static in which case the user is forced not to move. However this approach increased the measurement errors since that the algorithm is indeed sensitive to noise introduced by motion and noise is further enhanced if the entire tracked region is constantly being moved. Therefore the locking was brought back to the implementation.

## 5.3 Pulse Detection

To integrate this algorithm, there were three approaches :

- Run the applications separately and enable communication through server. This was the stage of the system in the initial prototype. There was no integration applied in this approach.
- Embedding Python in C++. This approach proved to be non-elegant as well as introducing further dependencies on the system. Although this approach is not impossible but could also further complicate the integration process.
- Port the Python implementation into C++. This allows for a complete integration with the Kanako library and the rest of the components such as visualisation. This also makes the

---

<sup>3</sup>Signal to Noise Ratio

<sup>4</sup>This was also tested during evaluation which seemed to be true on some individuals face.

design much neater and more elegant.

### 5.3.1 Strategy for Porting Code

First an understanding of the overall algorithm is required. Another set of problems arises from the lack of scientific computing packages such as Numpy[49] and Scipy[50]. Furthermore Python allows more compact and easy means of expressing code which is not the case in C++. Research was required to first target existing external libraries that could be used as a substitute to the Python libraries. GSL was chosen since it is free software, written in C. It thus has a C-like allocation and way of programming. GSL was used to perform computations such as Fast Fourier Transform. Note that OpenCV also provides API to find the Fourier Transform of images. However there are other computations performed throughout the process, hence once GSL was integrated, we made use of most of the available functionalities.

Figure A.2 better illustrates the strategy used to port the code. This code snippet shows how the main steps of the algorithm were targeted, and the required function at each single step was implemented as a C++ helper function. Python implementation is available from [51].

Figure A.7 illustrates the overall process of algorithm and the process of capturing frames.

### 5.3.2 Implementation

#### 5.3.2.1 Calculating Mean

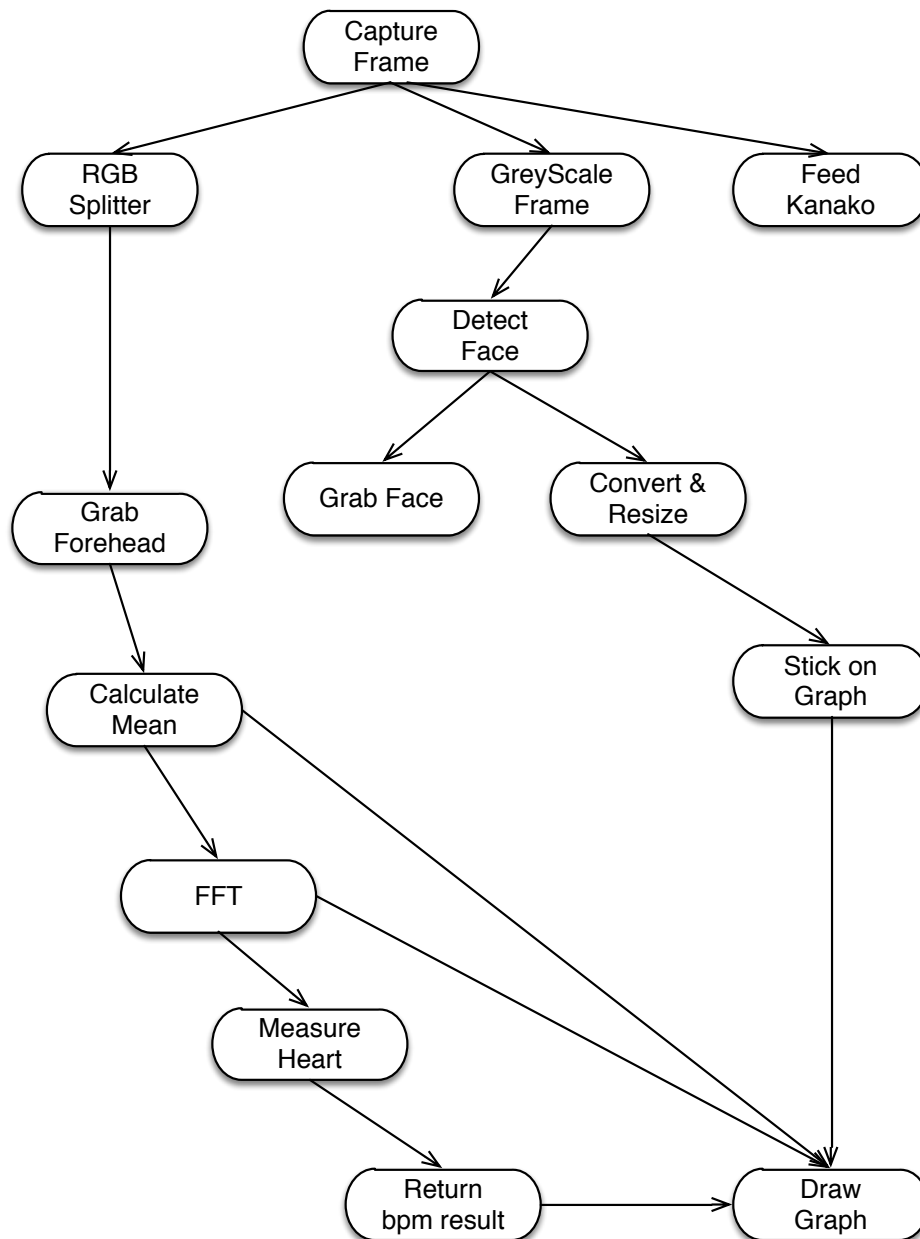
Having isolated the forehead region, the first step is decompose the region into the RGB channels and spatially averaging them in order to obtain the raw RGB traces. For the original implementation, we use all the three components including red(R), green (G) and blue (B). OpenCV actually returns the video frames in BGR format i.e. different colour-space format ordering. Since we are averaging over all of them, we don't care about the order. Perhaps only the fact that the green component is in the middle since later on we will experiment with the green component individually. The API `cv::mean` calculates the average of array elements i.e. all the pixels in that ROI.

---

```
double PulseDetector::calculate_mean(const cv::Mat& image) {
    cv::Scalar means = cv::mean(image);
    return (means.val[0] + means.val[1] + means.val[2]) / 3;
}
```

---

**Listing 5.2:** Mean calculation



**Figure 5.1:** Process of capturing frame and pulse detection algorithm.

### 5.3.2.2 Timestamps

We also store time stamps. Even though the Fourier Transform has the ability to provide fundamental physiological information easily, it makes the assumptions that the signals are in steady-state. But in reality physiological signals are often transient in nature [39]. Hence combination of both time and frequency domain information can prove to be a more reliable



physiological assessment. Boost.Chrono library [52] was used to implement timestamps<sup>5</sup>.

### 5.3.2.3 List Operations

Since the data collection is a continuous process, we start trimming the earliest. Most of these operations are provided by the Python libraries however these functions had to be implemented for our application. Example of these functions are basic list operations such as *slice*, *linspace*. The function *linspace* adopted from Python implementation, returns evenly space numbers over a specified interval. So this function takes the first time sample and last, evenly spaced by sample size. Note that sample size is the size of samples being gathered, once it reaches the maximum threshold (currently set to 250) the lists will get trimmed. Next we calculate the frames per second (fps) which is required later on for frequency calculations.

### 5.3.2.4 Interpolate

This is a mathematical function to construct new data points within the range of a discrete set of known data points. Given a set of data points  $(x_1, y_1) \dots (x_n, y_n)$ , the routine computes a continuous interpolating function  $y(x)$  such that  $y(x_i) = y_i$ . The GSL library provides a variety of interpolation methods, including Cubic splines and Akima splines. The python interpolation function uses a linear interpolation. Hence we define the type of interpolation for our *interp* function as *gsl\_interp\_linear*. This computation is required for the preparation before applying *FFT*.

Code snippet in listing 5.3 illustrates how to use the interpolate function in GSL.

---

```

1  vector<double> PulseDetector::interp(vector<double> interp_x, vector<double>
    data_x, vector<double> data_y) {
2      assert (data_x.size() == data_y.size());
3      vector<double> interp_y(interp_x.size());
4      vector<double> interpRes;
5      // GSL function expects an array
6      double data_y_array[data_y.size()];
7      double data_x_array[data_x.size()];
8      copy (data_y.begin(), data_y.end(), data_y_array);
9      copy (data_x.begin(), data_x.end(), data_x_array);
10
11     double yi;
12     int L = interp_x.size();
13     gsl_interp_accel *acc = gsl_interp_accel_alloc ();
14     gsl_spline *spline = gsl_spline_alloc (gsl_interp_linear, L);
15     gsl_spline_init (spline, data_x_array, data_y_array, L);

```

---

<sup>5</sup>Note that Chrono is used as an external library even though it is now part of the C++ standard library, however the system uses gcc 4.6 which was compatible with the CUDA version used by Kanako.

```

16 // Iterate through given x-interpolation range
17 for(int xi = 0; xi < interp_x.size(); xi++)
18 {
19     // For each value in x-range, get interpolate y value
20     yi = gsl_spline_eval (spline, interp_x[xi], acc);
21     interpRes.push_back(yi);
22 }
23 gsl_spline_free (spline);
24 gsl_interp_accel_free (acc);
25 return interpRes;
26 }

```

---

**Listing 5.3:** Interpolate Function

### 5.3.2.5 Hamming Window

The Hamming function comes from the signal processing literature, where it is typically used for smoothing the truncated autocovariance function in the time domain [53]. However we need to implement this in C++. The result of *interpolated* values are multiplied by this function.

The Hamming Window is defined as :

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{M-1}\right) \quad 0 \leq n \leq M-1$$

This can be calculated simply, as illustrated in the code snippet below:

```

vector<double> PulseDetector::hammingWindow(int M) {
    vector<double> window(M);
    if (M == 1) {
        window[0] = 1.0;
    } else {
        for (int n = 0; n < M; ++n) {
            window[n] = 0.54 - 0.46 * cos((2 * M_PI * n) / (M - 1));
        }
    }
    return window;
}

```

---

**Listing 5.4:** Hamming Window Calculation

### 5.3.2.6 FFT

GSL provides a FFT (Fast Fourier Transform) library. Fast Fourier Transforms are efficient algorithms for calculating the discrete Fourier transform (DFT). DFTs are useful because they reveal periodicities in data as well as relative strengths of any periodic components [54]. The function “*gsl\_fft\_real\_workspace\_alloc*” allocates a workspace for a real transform. Workspace structs are typically used to handle required memory allocations. The function “*gsl\_fft\_real\_wavetable*” prepares a trigonometric lookup table for an FFT of size  $n$  real elements. Both functions return a pointer to the newly allocated structs. The function “*gsl\_fft\_real\_transform*” is then called with the assumption that the given data is an array of time-ordered real data. It then returns a *halfcomplex\_coefficient*, an array of half-complex coefficients, which can be converted into an ordinary complex array i.e. *complex\_coefficient* by calling “*gsl\_fft\_halfcomplex\_unpack*”.

The implementation is illustrated below :

---

```

1 vector<gsl_complex> PulseDetector::fft_transform(vector<double>& samples) {
2     int size = samples.size();
3     double data[size];
4     copy(samples.begin(), samples.end(), data);
5     // Transform to FFT
6     gsl_fft_real_workspace* work = gsl_fft_real_workspace_alloc(size);
7     gsl_fft_real_wavetable* real = gsl_fft_real_wavetable_alloc(size);
8     gsl_fft_real_transform(data, 1, size, real, work);
9     gsl_fft_real_wavetable_free(real);
10    gsl_fft_real_workspace_free(work);
11    // Unpack complex numbers
12    gsl_complex unpacked[size];
13    gsl_fft_halfcomplex_unpack(data, (double *) unpacked, 1, size);
14    // Copy to a vector
15    int unpacked_size = (size / 2) + 1;
16    vector<gsl_complex> output(unpacked, unpacked + unpacked_size);
17    return output;
18 }

```

---

**Listing 5.5:** FFT Function

We also need to calculate the angle of the resulting raw FFT coefficients returned by “*gsl\_fft\_real\_transform*”.

---

```

1 vector<double> PulseDetector::calculate_complex_angle(vector<gsl_complex>
2     cvalues) {
3     // Get angles for a given complex number
4     vector<double> output(cvalues.size());
5     for (int i = 0; i < cvalues.size(); i++) {

```

```

5     double angle = atan2(GSL_IMAG(cvalues[i]), GSL_REAL(cvalues[i]));
6     output[i] = angle;
7 }
8 return output;
9 }

```

---

**Listing 5.6:** Calculate angles of raw FFT coefficients

Finally we obtain the absolute values of FFT coefficients as shown below:

```

1 vector<double> PulseDetector::calculate_complex_abs(vector<gsl_complex> cvalues)
2 {
3     // Calculate absolute value of a given complex number
4     vector<double> output(cvalues.size());
5     for (int i =0; i < cvalues.size(); i++) {
6         output[i] = gsl_complex_abs(cvalues[i]);
7     }
8     return output;
9 }

```

---

**Listing 5.7:** Calculate absolute value of given complex numbers.

### 5.3.2.7 Max FFT Index

The index of the largest value in the *filtered FFT* values correspond to the targeted frequency. Note that we filter out the frequencies less than 50 and greater than 180 since the normal human heart range is between 60 - 100 bpm<sup>6</sup>.

## 5.3.3 Problems Encountered

### 5.3.3.1 Code Structure & Debugging

One problem encountered was due to lack of expertise in implementing a relatively large algorithm which requires complex computations under the hood. Re-starting the process and breaking down the steps into smaller functions, helped significantly. More importantly it is essential to test the result at each stage of computation. This also includes testing the Python-equivalent list operations, which might seem fairly simple, however, if not tested individually, could prove to be error-prone. The testing strategy has been discussed in the testing section and further illustrated in listings A.1 & A.2.

---

<sup>6</sup>During one of the stages of testing and evaluation, one of the participants had a low heart rate, as low as 37. Initially the filtering was altered to take into account lower frequencies however this was an exceptional case. In order to comply with the original implementation, the filter range was kept the same.

### 5.3.3.2 Lack of Simple List Operations in C++

Python provides great facility for a lot of list and string operations. Unfortunately this is not the case in C++, hence implementing a number of helper function was required.

## 5.4 Facial Expression Analysis

Kanako was used as a third party library to perform face analysis. The source code was kindly shared by the author, who is a PhD student currently developing this API. The latest stage of the source code was obtained and alterations to the source code had to be made in order to perform a tightly-coupled integration with our main system.

### 5.4.1 Kanako Library Integration

The code was refactored in Kanako in a way that control was given to our main system. Both components relied on the main video capture loop. Calling Kanako from the main system gives more flexibility and the captured frames can be supplied to Kanako which in turn runs it's own face detection and classification logic. This also abstracts away all the complexity involved with the way Kanako operates. We can then simply call `KanakoLive::Run(kl, kanakoFrame)`. Note that here we make a copy of the captured frame and pass it on to Kanako. The function was further altered to return the data (AUs intensities), and this data is then passed on to the visualisation component to perform the visualisation.

## 5.5 Visualisation

Two forms of visualisation are applied. Emotion intensity bars and bpm graphs. The graphs only use the data from the pulse detector component. The data obtained from Kanako i.e. intensities and pulse detector i.e. pulse data, are analysed together and results are shown in the form of emotion intensity bars.

### 5.5.1 Emotion Intensity Bars

The intensity of bars are triggered depending on the "activeness" of a certain emotion. This data is collected from Kanako. We could straight away use the intensity to map to the intensity bar. The pulse is also used for certain emotions. Although the cognitive and physiological mapping requires further background research and evaluation, the system makes a few of basic assumptions. For instance if the smile AU is triggered and pulse is high, it deduces excitement.

Combination of surprised and high pulse could indicate fear. In addition to this, fear and low heart rate could indicate anxiety.

### 5.5.2 BPM Graph

For this part, we grab the detected face and resize it to place it on the top corner of the graph window. Note that even though the visualizer component is responsible to perform the visualisation, its main purpose is to visualise the integrated data of the two components. The graph is a form of visualisation, but using only the pulse detection data throughout the computation. Since the data is readily available within the same component, and the drawing logic is simple (using OpenCV highGUI API), this form of visualisation has been included within the same component, to reduce the overhead of passing data to the visualizer.

Three graphs displaying the result of computation are also drawn:

- Means : Represents variation of mean values over time.
- FFTs: Represents variation of frequencies over time.
- Bpms: Represents variation of bpms over time.

The data for this part is gathered throughout the computation stage of the algorithm. The code snippet in 5.8 illustrated the graph drawing methodology:

---

```

if (_means.size() >= 2) {
    // Clear the canvas
    graph = cv::Mat::zeros(BPM_GRAPH_HEIGHT, BPM_GRAPH_WIDTH, CV_8UC3);
    int gapFactor;

    double gap = BPM_TRACE_SIZE / (MAX_SAMPLES);
    double x = 0;
    for (int i = 0; i < _means.size() - 1; ++i) {
        cv::Point p1((int) x, (int) _means[i] + 50);
        cv::Point p2((int) (x+gap), (int) _means[i+1]+ 50);
        cv::line(graph, p1, p2, cv::Scalar(0, 255, 0, 0));
        x = x + gap;
    }
}

```

---

**Listing 5.8:** Mean vs Time Graph Implementation

For the graph the *gap* is calculated by taking the *width* of the graph window region, and max number of samples (250 in this case). For each sample data, it adds this factor between the first

point and the next i.e.  $i + 1$ . Also note that it needs a minimum of two points to start drawing. To further apply a nice visual effect, we could also apply a “stretching” effect to the graph. This is displayed where initially not enough data is gathered. To implement this, we use the *linspace* function which calculates the intervals between the existing points and the width of the graph. These values will be the  $x - axis$  values. Having determined the position on the  $x - axis$ , it then places the frequency points on the  $y - axis$ .

---

```

if (_fftabs.size() >= 2) {
    vector<double> xvals = linspace(0, BPM_GRAPH_WIDTH - 1, _fftabs.size());
    for (int i = 0; i < _fftabs.size() - 1 ; ++i) {
        cv::Point p1((int)xvals[i], BPM_GRAPH_HEIGHT - (int) _fftabs[i]*4-150);
        cv::Point p2 ((int)xvals[i+1], BPM_GRAPH_HEIGHT - (int) _fftabs[i+1]*4-150);
        cv::line(graph, p1, p2, cv::Scalar(0,255,255,0));
    }
}

```

---

**Listing 5.9:** FFT Graph Implementation with Stretching Effect

### 5.5.2.1 Problems Encountered

Initially due to rounding errors, the graph was not being drawn correctly. The fact that we need to calculate the gap is important initially it was trying to fit it in the window however there were not enough points hence the rounding errors and it lead to random drawing. explanation goes here

## 5.5.3 Limitations & Further Improvements

### 5.5.3.1 Forehead Region Adjustment

Once the face is locked, the processing starts and the window of the face will remain fixed because its taking average etc The current method for detecting forehead is rather naive. The algorithm uses calculated region of forehead with respect to the face. The system can let the user to adjust the position of forehead that fits their face best.

### 5.5.3.2 More Sophisticated Cognitive & Physiological Mappings

Not all the mappings were applied since not all the AUs are picked up accurately at all times, except for happiness, surprise and sadness. Future work could deploy the improved version of the library as well as performing more sophisticated mapping among the data. Furthermore more measures could be taken to make a more informed decision regarding the individual’s emotions.

For instance angry face, as well as examining the muscle movements in the face, it can also be combined using high pulse and the change of skin colour i.e. the person face turns red.



# CHAPTER SIX

# EVALUATION

This section describes the evaluation performed on both components of the system individually as well as the overall system. The main focus of this evaluation has been on the pulse detector algorithm's correctness and accuracy.

## 6.1 Pulse Detection Evaluation

All the experiments were conducted using a basic webcam embedded in a laptop (built-in iSight camera on a Macbook Pro system) to capture live video feed for analysis. The captured video frames were in colour (24-bit RGB having 8 bits/channel), processed at 15 frames per seconds (*fps*) with a pixel resolution of 640 x 480. Experiments were conducted indoors and with varying amount of sun light and artificial light as source of illumination.

The evaluation was performed in three parts :

- The implementation of the pulse detection algorithm was evaluated against the original implementation using :
  - Feeding both programs a pre-recorded video stream that simulates a webcam. This was to ensure both programs have produced the same output given the same input.
  - Running both applications simultaneously side-by-side with the same webcam and comparing the results.
- The accuracy of the algorithm was measured on human participants wearing a wrist blood pressure monitor device.

- Finally further analysis was performed to alter certain parameters in the algorithm to test its effectiveness, accuracy and behaviour. This test was carried out using a recorded video of myself.

## 6.1.1 C++ vs Python Implementation Comparison

### 6.1.1.1 Pre-processed Video

Throughout the process of porting the algorithm, both versions of the code, the Python and C++ code, were modified in order to read a pre-recorded video frame lasting around 30 seconds. The resulting values from both algorithms were then compared in order to ensure that the computations in the ported implementation is consistent with those of the original implementation, at each step. Finally both programs were run on the same pre-recorded video (i.e. same input).

Figure 6.1 illustrates a comparative analysis of the Python implementation of algorithm vs the C++ implementation. The sample video was recorded on myself, while sitting on a chair with a stable heart rate. Note that in order to carry out this experiment, an OpenCV program was used to record the video due to specific frames formats required by the OpenCV API in order to read video files. This program is included as part of the project submission.

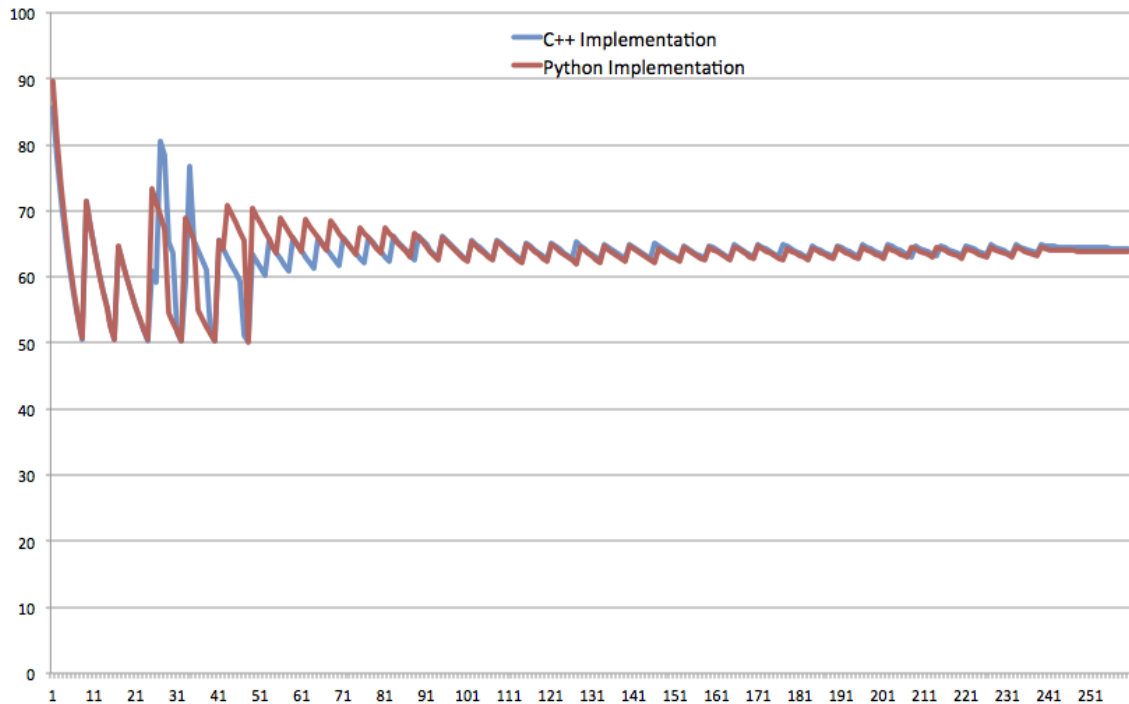
It is important to note that our claim is that both algorithms are equivalent, though both systems are not identical. Hence the data result varies slightly, due to different frame rate processing of each implementation. This is due to using two different languages and two different versions of OpenCV APIs<sup>1</sup>.

### 6.1.1.2 Running Live

For this experiment, the two programs were tested while processing captured frames from the webcam. A Polar RS300X heart rate monitor device was worn by myself. Although the results seemed consistent with each other as well as the result from the heart monitor device, the correlation between the measured data is not perfect. Note that both implementations do have slight variation in terms of speed due to using two different versions of OpenCV video processing APIs. Python implements seems to read/process frames slightly faster. Hence we expect the results to match after the computed heart beat rates have reached a stable state i.e. when there is almost no movement of the participant or other forms of sampling noise. The reading on the heart monitor device while performing this experiment was between the range of 65 and 72 bpm. Unfortunately it was not possible to store the continuous heart measures of

---

<sup>1</sup>When using OpenCV *read* function, given a recorded video, the *fps*(frames processed per second) also increases, hence at each step of reading a frame, we call *cv::wait(50)* to introduce slight delay since the *fps* affects the algorithm.



**Figure 6.1:** Python vs C++ implementation for a 30 seconds pre-recorded video.

the device, however, obtaining access to a set of continuous measures and plotting them, could further enhance the evaluation of this experiment. Therefore there is no direct synchronisation between the three measures, hence it's an estimate.

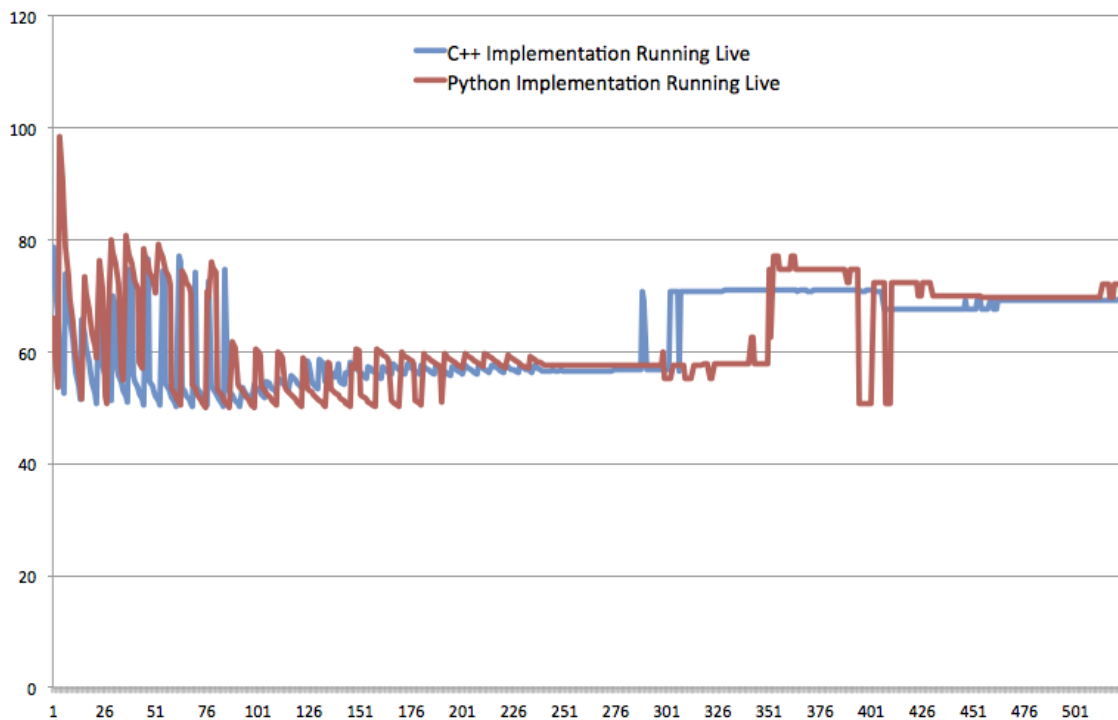
## 6.1.2 Experiments Conducted on Human Participants

### 6.1.2.1 Set up

An Omron R2 Wrist Blood Pressure Monitor was used to take *one-off* measures. This device has a one-button operation, once pressed it starts immediate measurements and it takes roughly around 30 seconds to display the result.

### 6.1.2.2 Methodology

Five participants (including 4 male and 1 female) were asked to sit as still as possible in front a computer with an embedded web-cam, to minimise noise due to any movements. This experiment was conducted during daylight on a rainy day, hence the illumination source was slightly lower than a typical sunny day. The source of illumination also included artificial light in the lab environment. The experiment involved two stages. First stages, measurements were taken while the participant sat still and their heart rate was stable. They were asked to position their arm so



**Figure 6.2:** Python vs C++ implementation running live for about 1 minute while wearing a heart monitor device.

that the wrist unit was at the same level as their heart. Three measurements were collected while the application was estimating a continuous measure of the participant's bpm (after the values got stabilised) as well as one-off measures from the wrist heart monitor device. A second set of measurements were taken from participants undergoing a short set of exercises. They were then asked to perform 10 jumping-jacks or push-ups. Immediately after the exercise ended, the participant sat on a chair and was asked to hold his/her head still without any support. Meanwhile the wrist blood pressure device was put back on their wrist. Again three measures were taken of the elevated heartbeat rates resulting from the exercises.

### 6.1.2.3 Experimental Result

Please refer to C.1 for the result of this experiment.

### 6.1.2.4 Challenges & Flaws

It typically takes a minimum of 10-15 seconds for the pulse results returned by the algorithm to stabilise. The wrist device also takes also half a minute to return the result which corresponded to that moment of time the measure was taken, hence it was not continuous. Therefore the correlation between the two sets of data was not 100% perfect. Typically 15-20 seconds were

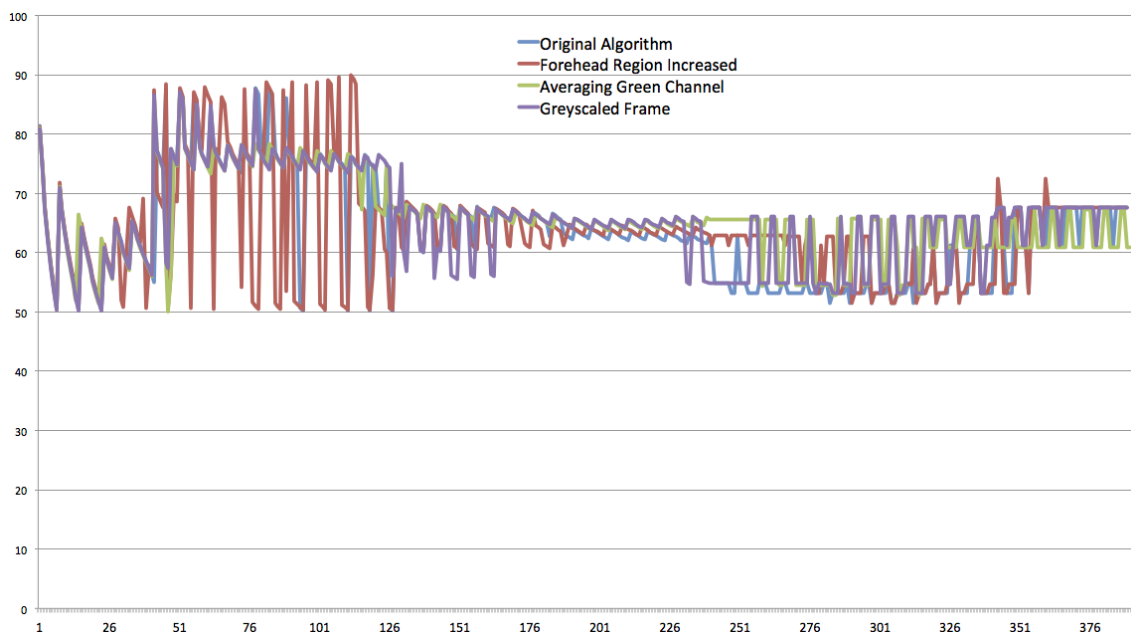
given until for the pulse results to stabilise and the wrist unit button was pressed in advance in order to capture the results occurring at roughly the same time.

It was also difficult for the participants to stay still specially after the exercise due to panting, where further noise as a result of their movements was introduced during the experiment. Such movements could potentially decrease the accuracy.

Another challenge for the heart beat monitor algorithm was the difference between the shape of their forehead and tone of skin colour. We found that for some people, the algorithm worked much better on the cheeks i.e. less SNR<sup>2</sup>. However this was harder to capture for some other people and they preferred forehead since it was more comfortable for them to expose the forehead during the experiment. For reasons of consistency in the experimental data, all the measurements included with this experiment were gathered from participants foreheads. Further experimentation and analysis is required to confirm the effectiveness of the algorithm on different regions of the face as well as different tones of skin.

### 6.1.3 Further Analysis

The following configuration changes were made to the algorithm and measurements were taken using the same pre-recorded video used previously.



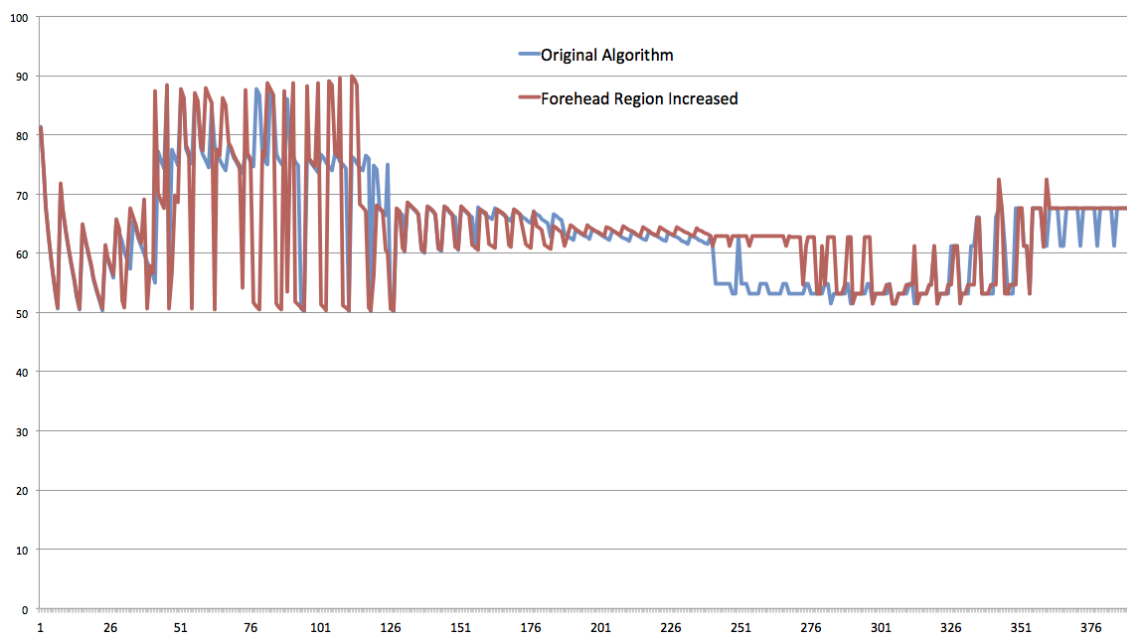
**Figure 6.3:** Three different adjustments applied to the algorithm separately.

<sup>2</sup>Signal to Noise Ratio

### 6.1.3.1 Increasing Forehead Region

The algorithm requires any patch of skin in the facial area to operate on. Hence increasing the region has the potential of improving the algorithm. However this could also introduced more noise since unwanted areas such as hair and eyebrows might also be captured. Also the forehead region of every individual is different. Figure 6.4 illustrates the effect and indeed some peaks i.e. noise was introduced as a result. However the overall average of both analysis were similar

Original Alg Ave	Forehead Region Increased Ave	Wrist Heart Monitor Measure
63.68	63.72	63

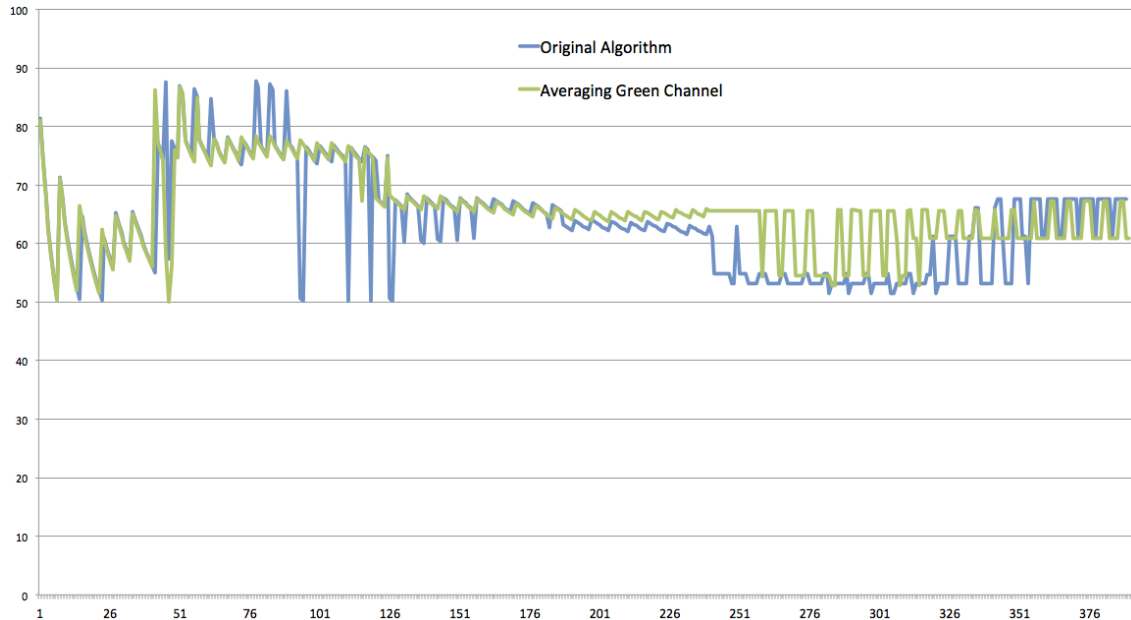


**Figure 6.4:** Increased forehead region to cover the entire forehead.

### 6.1.3.2 Switching From RGB to Single Green Channel

Due to current debate on whether the algorithm performs better with green channel rather than RGB, we put this into test. Using the same video, the accuracy of algorithm degraded slightly. Hence we deduce that RGB works better in this case. Such claim requires further analysis and experimentation on participants. The effect is illustrated in figure 6.5.

Original Alg Ave	Green Channel Ave	Wrist Heart Monitor Measure
63.68	65.73	63

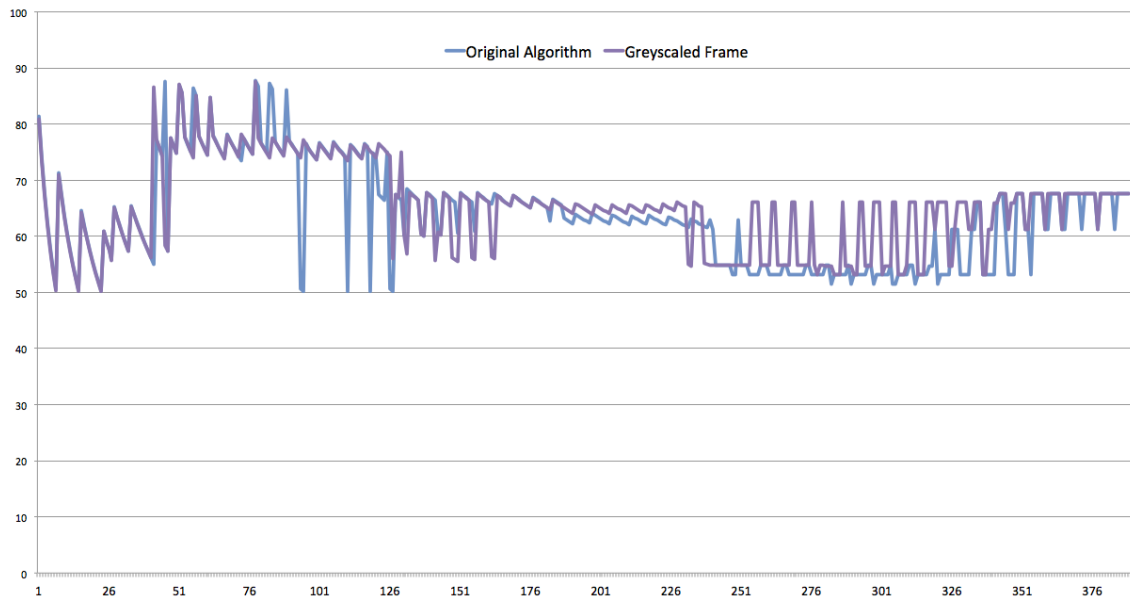


**Figure 6.5:** Averaging over RGB vs Green channel.

### 6.1.3.3 RGB vs Greyscale Frame

Since the algorithm operates through variations in reflected light, using the three components of RGB channel, we were under the impression that the conversion from colour to grey scale might cause the algorithm not to work well at all. Although there was slight decreased in accuracy, it was still able to pick up the frequency components. The effect is illustrated in figure 6.6.

Original Alg Ave	Greyscale Frame Ave	Wrist Heart Monitor Measure
63.68	65.23	63



**Figure 6.6:** RGB vs Greyscale Frame.

#### 6.1.4 Future Evaluation Work

The recording time for this present work was relatively short and future work needs to extend the time window to enable long-term, continuous measurements. Future work can also be carried out to evaluate how different skin tones affect the accuracy of this technique. According to our observations during the experiments, bpm's were harder to detect for participants with pale skin tone compared to participants with darker skin. Such observations require further testing and evaluations. The effect of the adjustments made to the algorithm also requires further analysis.



# CRITICAL ASSESSMENT

This chapter reflects views on the overall system and challenges involved throughout the project.

## 7.1 Challenges

This was a research-based project, requiring extensive amount of reading on current progress of existing systems in the field. The requirements of the system evolved through incremental research and feasibility tests. Various techniques and algorithms were considered and then abandoned due to lack of feasibility to be integrated with the system. The following sections cover the main challenges involved in this project.

### 7.1.1 Learning a New Language & Platforms

C++ is a very disciplined language. In a sense, C++ forces a programmer towards proficiency in terms of writing elegant and well-structured code and designing a neat architecture. Hence the learning experience was most certainly an enjoyable one.

With regards to the library and platforms used for this project, having OpenCV up and running for the first time could be tricky. OpenCV is an open source, cross-platform library, which is quite powerful in terms of the capabilities it offers. However despite the latest improvements in documentations and support, problems might arise, in which case debugging could be difficult. The very first problem that was encountered throughout this project was due to lack of defensive programming in almost all of the examples available with the library. This was due to the fact that OpenCV reads the video frames captured from the camera in a *while* loop. However some cameras on mac machines take a few seconds to start sending the frames to the application trying to connect to the embedded camera. Hence this caused the application to fail without giving any

useful errors. This issue can be easily fixed by altering the logic while looping to read camera frames.

---

```
if( capturedFrame ) {
    // do the logic
}

else {
    // Error
    return -1;
}
```

---

**Listing 7.1:** Initial bug in OpenCV examples included within the framework

---

```
while (true) {
    // Read frame
    cv::Mat capturedFrame;
    camera.read(frameOriginal);
    if (! capturedFrame.empty()) {
        // do the logic
    }
    // If this fails, keep trying to read frames, perhaps camera is too slow to
    // initialise
}
```

---

**Listing 7.2:** Fixing the OpenCV example code bug

This code will run in an infinite loop until frames are captured, in which case it will carry on the work.

## 7.1.2 Challenges of Using Kanako

In terms of compatibility, this library relies on an advanced hardware. It uses CUDA which is parallel computing platform which enables using GPUs for general purpose processing. Another problem is that since this library is at its early stages of development (our system uses the 0.1 version of the library), hence throughout the development of the system, there were some updates to the version of the library in use. Also the library does not allow training to be carried out, rather provides pre-trained models as classifiers.

### 7.1.2.1 Memory Leak Bug Fix

In the latest update of the code, there was a memory leak issue which was reported to the author. Even though the code would compile and run, it caused crashes on every run. This was a new version of the code and the effectiveness of the algorithm had been improved significantly. Hence effort was put in order to find the fault in order to be able to use this version of the code for the project since the result was much more satisfactory. Since the library failed to free resources, having examined the code, the resources were being freed. However *print statements* can highlight at which instance the cleaning logic was not actually being executed. In this case the problem was due to having inheritance inside the structure, where the parent class destructor was never called and this led to memory leak issues. Instead it needed an explicit clean up and this fixed the issue.

### 7.1.3 Porting an Existing Algorithm

Porting the python code required background knowledge of the algorithm. But the major challenge was the lack of supporting libraries for the computations. Hence further research was conducted to find the most suitable numerical libraries and some functionalities such as simple python functions were implemented from scratch due to lack of these functionalities in C++ standard libraries. Boost [45] also proved to be useful in terms of providing numerical library support.

Another challenge was targeting the algorithm itself. Since the system is composed of various components, at first glance, the underlying algorithm was not obvious. Hence the components that were not required, such as interface, capturing video frames, face detection etc were eliminated first. The rest of the code was further studied to highlight and further reading material was covered to gain a better understanding of the algorithm.

## 7.2 Limitations

### 7.2.1 Pulse Detector Algorithm Sensitivity to Motion

Even though the accuracy of the algorithm has proven to be quite impressive, this is only the case if in a very stable position. Any slight movement leads to inaccurate results which needs a few seconds to pick up the correct values from the frames again since it's averaging the captured over time. Another observation made was differences in individuals skin colour and facial shape. It is indeed the case where certain skin patches on an individual's face are better processed by the algorithm. During evaluations, participant's cheeks were examined off the record. Due to lack of

time the difference in measures were not collected, however this requires further evaluation. But with this knowledge in mind, user is able to find the best region of their face which works the best, as well as taking into account the degrees of illumination. Lighting is of course important since it's the amount of reflected light that enables such processing in the first place.

## **7.2.2 Sensitivity of Algorithms to Sources of Illumination**

As with many other existing computer vision algorithms, this has always been the main challenge. The algorithms effectiveness degrades if the lighting condition is not suitable. An ideal lighting condition would be indoor with artificial light sources. Better performance was achieved when the system was tested indoor with additional source of natural light such as during sunny days.

## **7.2.3 Deficiencies**

### **7.2.3.1 Kanako Memory Leak Issues**

Even though the previous bug fix prevented the crashing of system, there are other existing bugs which cause memory leak issues. These issues have yet to be resolved by the author. Currently if the application runs over long period, eventually will cause the system to run out of memory and hence crashes. This is due to the fact that the training data is quite large<sup>1</sup>. Since the classification stage would require loading all this data to the memory, if the application is run and terminated constantly, it leads to shortage of memory. This fault was discovered during the implementation of the system.

### **7.2.3.2 Code Maintainability**

During the stages of designing the system, the aim was to make the code as maintainable as possible. However one challenge with this was the fact that Kanako is not quite an API yet. Hence in order to be integrated with our system, modifications had to be made to the code. This lead to a tightly-coupled integration of Kanako with our system. However the integration strategy is quite simple and therefore not too problematic to further alter the system to break this integration.

### **7.2.3.3 Facial Expression SVM Models**

Current pre-trained models for happiness, surprised and sadness seem to perform really well. However fear, anger and disgust do not work very well. Hence Kanako requires further improvements in the algorithms.

---

<sup>1</sup>Some training models are as large as 197.7 MB

## **7.3 Future Work**

### **7.3.1 Improved Facial Analysis Library**

As mentioned before, Kanako is still under development and in the near future it will be further improved.

### **7.3.2 Improving Pulse Detection Algorithm**

As suggested in the paper [41], further signal processing techniques can be used in other to reduce sources of fluctuations in light due to artefacts such as motion and changes in ambient light conditions.

### **7.3.3 Cognitive & Physiological Mapping Based on Further Evaluation**

Currently the system makes two simple assumptions about human emotions. If the person is happy and the pulse is high, deduce excitement. This assumption excludes high pulse due to exercise since we did not expect our participants to look happy but rather tired! However if the person looks distressed or sad and pulse is high, this could indicate fear. Further evaluation of the system and current algorithms can make a stronger case while making such assumptions. As discussed in chapter 2, the system is nowhere near examining the context of situation. Such evaluation could also prove that perhaps current measures are still not enough. There are still other means of measures that could be taken such as temperature (since the user is touching the computer and keyboard) or skin conductivity which might be easier to apply in the case of pulse detector algorithm. Furthermore, measures such as the amount of pressure applied to the keyboard can also be measured as an indication of frustration or distress.

### **7.3.4 Eulerian Video Magnification for Revealing Subtle Changes**

Even though Eulerian Video Magnification (EVM) was the source of inspiration for this project and extensive amount of effort and time was spent in examining this algorithm and the code, after integration with the system, it reduced the effectiveness of our facial analysis algorithm. Initial assumption was that this could potentially further improve it, by amplifying regions of the face to reveal subtle changes in video. It was an assumption that this would enable us to collect pulse data and measure blood flow. In other words, video amplification could reveal cues that are not visible by the naked eye. However one major limitation with this algorithm was the high-level noise it introduces to the frames. It also performed well on stable videos which lacked motion, even lack of making facial expressions. We did consider using EVM to reveal pulse initially by

amplifying the frames to be able to visually see the pulse. This would not however give us any measures as well as reducing the performance of facial analysis algorithms. Therefore another approach was required and the current pulse detection algorithm was found after doing further research.

### **7.3.5 Multiple Face Analysis**

Currently the application only assumes a single user. Future work can enable multi-user emotion detection.

### **7.3.6 Stronger Face Detection Algorithm**

More advanced face detection algorithm uses skin colour. This leads to a more accurate face detection. Currently there is a possibility when the user is not present in front of the camera, the surrounding objects in the environment might be picked up, mistaken as a face.

### **7.3.7 Adjustable Forehead Region**

The current pre-calculated forehead region is restrictive since it does not perform an explicit *forehead* detection. Rather face is detected, and the forehead is calculated however people's facial form differs from person to person. Enabling the user to make such adjustments could improve the algorithm to work best for different individuals. Such adjustment could be applied to the region of the face which has the best coverage as well as smoothness of the patch of skin<sup>2</sup>.

### **7.3.8 User-tailored System**

Perhaps the most challenge of machine learning and data classification, is the generalisation and how well it performs when presented unseen data. However the system could also be trained on the user themselves as well as the main training which requires a large database of data.

### **7.3.9 Voice Processing**

For this project, voice processing was not studied in detail. However as mentioned in Chapter 2, there are existing tools and algorithms for voice classification. Future work can further analyse the existing systems and seek further opportunities to extend the existing system.

---

<sup>2</sup>This might be due to the fact that the light is better reflected from a smooth region, or regions with more blood flow such as cheeks.

### 7.3.10 Further Testing

Further evaluation can be performed on the current system to test the effectiveness of the algorithms under low lighting conditions and different illuminations. The system can be tested under the influence of ambient light intensity, and varying the light intensity. A more reliable testing should also be performed in a different environment i.e. an environment where the system has the potential to be deployed, rather than in the laboratory.

## 7.4 Application

There are various possible applications for an emotion analysis system, solving various problems whether it's medical or psychological, learning, entertainment, or social development. Some of the applications mentioned in this section already exist. We will further examine how they can be improved and propose other applications.

### 7.4.1 Teaching App

An effective learning highly depends on emotions. Curiosity and fascination sparks many learning episodes. As the level of difficulty increases however, one might feel confused, frustrated or stressed. Due to these negative feelings, learning might even be abandoned. In education, pupil's social signals inform the teacher whether there is a need to adjust the instructional message. Successful human teachers acknowledge this and work with it [20]. When a student finds a fun tutor, they are more likely want to interact with them and are further encouraged to learn. Many of us have been greatly influence by a special teacher who engaged us with creative and clear explanations who by his own enthusiasm transmitted a love for a topic [3].

Computer that can also guide a student towards a successful learning experience, responding intelligently when he or she is frustrated. Learning could be assisted by not only prompting (explicitly designating what user likes or does not like) the user for their preferences, but also by watching their expressions and behaviour. Such system can support a pupil to build their confidence gradually. For instance it could provide clues when the student seems puzzled or worried. Or change the difficulty level depending on their reaction to a given problem. The goal of the system will be defined as how to redirect or motivate the pupil at that the moment that matters the most. Once the obstacles are passed, progress will be made.

An old example of this application was built at IBM Almaden Research Laboratory, called "COACH" [55] by Ted Selker. This system gives adaptive feedback while the user tries to learn the programming language LISP. Selker's system watches user actions to build an adaptive user

model which is capable of selecting appropriate advice. The system chooses to use description, example, syntax, timing, topic etc depending on the level of experience and skills they user has demonstrated. This system consider the user's opinion of the learning environment, opinion of the topics read, comfort with the topic learned etc [3]. Such existing system could use more cues such as if the user is pleased or frustrated for instance. Experienced teachers say it is important to detect the level of engagement and frustration of a student. Offering help when the user is already engaged, could be distracting. Hence the system can effectively provide feedback when it's needed the most.

### **7.4.2 Angry Driver Recognition**

Another use-case scenario of our application is as an embedded automobile interface, capable of detecting stress or anger. Most accidents are caused by people who are angry or upset. The aim is to develop a system which can detect stress level which may or may not indicate a serious medical condition [3]. A team of researchers in Switzerland have developed a facial recognition application, with embedded cameras into steering wheels. The application detects angry or upset drivers who tend to be more aggressive and less attentive. The team claims, assessing emotional state of drivers could improve safety [56].

### **7.4.3 Effect of Art on Emotions**

Some of the greatest fluidity in expressions are shown in actors and musicians, where expressing emotions is part of their profession [3]. But what is it about art or music that gives rise to certain human emotional feelings? What is it about a piece of painting or a piece of music that brings us joy and happiness or saddens our emotions? An application capable of analysing the user's reaction to certain piece of art, can perhaps learn more about the art itself than one might expect.

### **7.4.4 Lie Detector**

An application that can reach the accuracy of facial expression analysis to the extend that is able to distinguish between expressions and microexpressions in the face in order to expose deceivers. These microexpressions are considered as the level of control involved in perfecting one's poker face to hide emotions. One might expect for such application to perform even more accurately than people.



### **7.4.5 User-tailored Recommendation App**

A personal recommendation app which can be used as a monitoring application running in the background. During certain times of day it could analyse the user and for instance if the user looks angry or tired, it might suggest taking a cup of a tea! A more sophisticated application, could be based on patterns learned by user habits. A system with more knowledge of its user, can also be more of assistant.



# CONCLUSIONS

## 8.1 Summary

This project has defined key issues in developing an intelligent system capable of detecting basic human emotions. It also touched on how this capability can change the way they work and what applications it will enable. Computers can currently discriminate about six different facial expressions. Furthermore they can analyse our voice, determine the blood flow by analysing the skin, measure our temperature etc. As computers become more lightweight and wearable, they can measure emotional responses wherever and whenever they occur. We argued that a system with this capability, can get to know its user, recognise their emotion and respond accordingly. Existing human emotion tools were studied, and to our knowledge, none of the existing systems combine measuring physiological signals with their expression analysis algorithms. Research was pursued to further improve the existing algorithms and capabilities which lead to the development of Affective Mirror.

## 8.2 Current Status

Affective Mirror fulfils all the requirements specified for this project. Its current capabilities are detecting six facial expressions and measuring the heartbeat. The facial expression analysis library in use, is still under development. Further updates to the existing version of the library will be required in the future. The pulse detection algorithm implements the core algorithm and it has the limitation being sensitive to rapid movement. The two systems have been integrated and the most basic mapping between the expression such as happy and heartbeat has been performed to deduce additional emotions such as excitement. A third component performs the visualisation of the data gathered during face analysis stage.

### 8.3 Further Complications

Both people and computers would have a relatively easy time recognising emotions, if they were always displayed consistently. Most emotions do not map to a fixed form of septic modulation. Another complication is the rise of physiological responses similar to those in a particular emotional state without corresponding to an emotion. For instance heart rate increases during exercise. Such responses must be analysed within the context of the situation [3]. Another factor further complicating the emotion recognition is the fact that different individuals express emotions differently. Such patterns also depend on gender, context and social and cultural expectations [3]. But how can we expect a computer to recognise emotions if everyone expresses them differently? Although solving this problem would be a terrific accomplishment but perhaps unnecessary. Nicholas Negroponte pointed out an alternative solution is for your personal computer to understand you and your language [57], rather than everyone's emotions, physiological cues or facial expression patterns. The individual's personal computer will respond best if it is able to perceive context. Such affect recognition problem can be posed as a computer learning and recognising pattern in order to determine which features are the best predictors for each individual [3]. The required computational power and algorithms already exist. Therefore in the near future, computers will have the capability to get to know us and adopt to our moods accordingly. Our personal computer will provide sympathy when we need it most, and perhaps gives us the support when we need it most.

# TESTING

## A.1 Algorithm Correctness

### A.1.1 Unit Tests

Each step of calculation in the algorithm, contains small unit tests using fixed, hard-coded data fed to both programs, to ensure correctness of implementation. Please note that the algorithm steps for C++ are abstracted away from the test units. However test written for python was injected inside the implementation. Inside the algorithm, the frame data to be analysed is overwritten by the dummy-data to make a consistent comparative analysis between the implementation of the two algorithms. The unit tests written for both programs are illustrated in A.1 and A.2

testing strategies such as unit testing

---

```
1
2 void testingAlgo(){
3     vector<double> _fftabsTesting;
4     vector<double> _frequenciesTesting;
5     // Frames per second frame hard-coded for both programs
6     double _fps = 11.1111111111;
7     double m[10] = {55,56,57,57,55,54,53,55,57,56};
8     double t[10] = {1.1,1.2,1.25,1.4,1.5,1.6,1.75,1.8,1.9,2.0};
9     vector<double> test_means, test_times, test_even;
10    test_means.insert(test_means.end(), m, m+10);
11    test_times.insert(test_times.end(), t, t+10);
12    test_even = fd.linspace(test_times[0], test_times.back(), 10);
13
14    int sampleSize = test_means.size();
15    vector<double> test_interp = fd.interp(test_even, test_times, test_means);
16    fd.dump("Test even", test_even);
17    fd.dump("Test interpolated", test_interp);
18
```

```

19 vector<double> test_hamming = fd.hammingWindow(sampleSize);
20 printf("-----\n");
21 fd.dump("Hamming Window : ", test_hamming);
22
23 fd.list_multiply_vector(test_interp, test_hamming);
24 fd.dump("Multiply interp * hamming\n", test_interp);
25
26 printf("-----\n");
27 double allMeans = fd.list_mean(test_interp);
28 printf("Mean %lf : ", allMeans);
29 printf("----- \n");
30
31 fd.list_subtract(test_interp, allMeans);
32 fd.dump("Subtracted interpolate\n", test_interp);
33
34 printf("-----\n");
35 vector<gsl_complex> fftraw = fd.fft_transform(test_interp);
36 fd.dump_complex("one dimensional discrete fft\n", fftraw);
37
38 printf("-----\n");
39 vector<double> angles = fd.calculate_complex_angle(fftraw);
40 fd.dump("angles\n" , angles);
41
42 printf("-----\n");
43 _fftabsTesting = fd.calculate_complex_abs(fftraw);
44 fd.dump("FFT Absolute\n", _fftabsTesting);
45
46 printf("-----\n");
47 _frequenciesTesting = fd.arange((sampleSize / 2) + 1 );
48 fd.dump("frequency spaced interval : ", _frequenciesTesting);
49
50 printf("%lf -----\n", _fps);
51 printf("%lf", sampleSize);
52
53 fd.list_multiply(_frequenciesTesting, (_fps / sampleSize));
54 fd.dump("frequency multiplied : ", _frequenciesTesting);
55
56 printf("-----\n");
57
58 vector<double> freqs(_frequenciesTesting);
59 fd.list_multiply(freqs, 60.0);
60 fd.dump("Multiplied frequencies : ", freqs);
61
62 printf("-----\n");
63 vector<double> fitered_indices = fd.list_filter(freqs, 50, 180);
64 fd.dump("Filtered frequencies : ", fitered_indices);
65
66 printf("-----\n");
67 _fftabsTesting = fd.list_pruned(_fftabsTesting, fitered_indices);
68 fd.dump("Pruned FFT abs : ", _fftabsTesting);
69
70 printf("-----");
71 freqs = fd.list_pruned(freqs, fitered_indices);
72 fd.dump("Pruned frequencies : ", freqs);

```

```

73
74     printf("-----");
75     angles = fd.list_pruned(angles, filtered_indices);
76     fd.dump("Pruned angles : ", angles);
77
78     printf("-----");
79     int max = fd.list_argmax(_fftabsTesting);
80     printf("%lf\n", max);
81
82     double bpm = freqs[max];
83     printf("%lf\n", bpm);
84 }

```

---

### Listing A.1: C++ Test Unit

---

```

1 # ----- START OF BPM CALC
2 # face rect is constant at this point, therefore forehead is locked
3     forehead1 = self.get_subface_coord(0.5, 0.18, 0.25, 0.15)
4     self.draw_rect(forehead1)
5 # calc mean of RGB values in the forehead rect area in the input frame
6     vals = self.get_subface_means(forehead1)
7 # append mean in array of means
8     self.data_buffer.append(vals)
9 # **** test code - This will overwrite frame data
10    print "-----"
11    self.data_buffer = [55,56,57,57,55,54,53,55,57,56];
12    self.times = [1.1,1.2,1.25,1.4,1.5,1.6,1.75,1.8,1.9,2.0]
13    print "Means: ", self.data_buffer
14    print "Times: ", self.times
15 # ****
16 # IF no. of mean values in array > 250
17     L = len(self.data_buffer)
18     if L > self.buffer_size:
19         # reload array with the last (i.e. latest) 250 values
20         self.data_buffer = self.data_buffer[-self.buffer_size:]
21         # reload timestamp array with last (i.e. latest) 250 values
22         self.times = self.times[-self.buffer_size:]
23         # set L = 250
24         L = self.buffer_size
25 # create a numpy array using array of means
26     processed = np.array(self.data_buffer)
27     self.samples = processed
28 # If no. of mean values > 10
29     if L >= 10:
30         # get dimensions of numpy array
31         # *** NOT USED **
32         self.output_dim = processed.shape[0]
33         # calc frames-per-second = L / (times[max] - times[0])
34         self.fps = float(L) / (self.times[-1] - self.times[0])
35         print "FPS: ", self.fps
36         # get evenly spaced time values between times[0] and times[max] for

```

```

        L number of items
37     even_times = np.linspace(self.times[0], self.times[-1], L)
38     print "Event times: ", even_times
39     # interpolate mean values over evenly distributed times (???)
40     interpolated = np.interp(even_times, self.times, processed)
41     print "Interpolated: ", interpolated
42     # apply Hamming window to interpolated mean values
43     interpolated = np.hamming(L) * interpolated
44     print "Interpolated-hamming: ", interpolated
45     # calc mean of interpolated means, subtract it from interpolated
        means
46     interpolated = interpolated - np.mean(interpolated)
47     print "Interpolated-means: ", interpolated
48     # calc one dimensional discrete fft of processed interpolated values
49     raw = np.fft.rfft(interpolated)
50     print "FFT-raw: ", raw
51     # get angle of raw fft data
52     phase = np.angle(raw)
53     print "Angles: ", phase
54     # get absolute values
55     self.fft = np.abs(raw)
56     print "FFT Absolutes: ", self.fft
57     # cal frequencies - using spaced values within interval between 0 -
        L/2+1
58     self.freqs = float(self.fps) / L * np.arange(L / 2 + 1)
59     print "Frequencies: ", self.freqs
60     # Get indices of frequencies > 50 and less than 180
61     freqs = 60. * self.freqs
62     print "Frequencies * 60: ", freqs
63     idx = np.where((freqs >= 50) & (freqs <= 180))
64     print "Filtered indices: ", idx
65     # Used filtered indices to get corresponding fft values, angles, and
        frequencies
66     pruned = self.fft[idx]
67     print "Pruned FFT abs: ", pruned
68     pfreq = freqs[idx]
69     print "Pruned frequencies: ", pfreq
70     phase = phase[idx]
71     print "Pruned angles: ", phase
72     self.freqs = pfreq
73     self.fft = pruned
74     idx2 = np.argmax(pruned)
75     print "Argmax: ", idx2
76
77     t = (np.sin(phase[idx2]) + 1.) / 2.
78     t = 0.9 * t + 0.1
79     alpha = t
80     beta = 1 - t
81
82     self.bpm = self.freqs[idx2]
83     self.idx += 1
84     print self.bpm
85     print "-----"
86

```



```
87 # ----- END OF BPM CALC
```

---

### Listing A.2: Python Test Unit

#### A.1.1.1 Results of Running Both tests

---

```
Test even[1.100000] [1.200000] [1.300000] [1.400000] [1.500000] [1.600000] [1.700000]
[1.800000] [1.900000] [2.000000]

Test interpolated[55.000000] [56.000000] [57.000000] [57.000000] [55.000000]
[54.000000] [53.333333] [55.000000] [57.000000] [56.000000]
-----
-----
-----
-----
one dimentional discrete fft
[-0.000000 : 0.000000] [-122.436320 : -43.281558] [4.896616 : 7.861698] [2.593014 :
2.520825] [1.084540 : 0.849218] [-0.823211 : 0.000000]
-----
angles
[3.141593] [-2.801801] [1.013748] [0.771283] [0.664302] [3.141593]
-----
FFT Absolute
[0.000000] [129.861256] [9.261919] [3.616390] [1.377461] [0.823211]
-----
11.111111 -----
11.111111frequency multiplied :
[0.000000] [1.111111] [2.222222] [3.333333] [4.444444] [5.555556]
-----
Multiplied frequencies :
[0.000000] [66.666667] [133.333333] [200.000000] [266.666667] [333.333333]
-----
Filtered frequencies : [1.000000] [2.000000]
-----
Pruned FFT abs : [129.861256] [9.261919]
-----Pruned frequencies : [66.666667] [133.333333]
-----Pruned angles : [-2.801801] [1.013748]
-----0.000000
66.666667
```

---

### Listing A.3: C++ Test Result

---

```
Means: [55, 56, 57, 57, 55, 54, 53, 55, 57, 56]
Times: [1.1, 1.2, 1.25, 1.4, 1.5, 1.6, 1.75, 1.8, 1.9, 2.0]
FPS: 11.1111111111
Event times: [ 1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2. ]
```

```

Interpolated: [ 55.  56.  57.  57.  55.  54.  53.33333333  55.  57.  56. ]
Interpolated-hamming: [ 4.4          10.50669515  26.22694478  43.89
53.47422331
52.5019647  41.06666667  25.30670111  10.6943147  4.48      ]
Interpolated-means: [-22.85475104 -16.7480559  -1.02780626  16.63524896
26.21947227
25.24721366  13.81191563  -1.94804994 -16.56043634 -22.77475104]
FFT-raw: [ -1.42108547e-14 +0.j          -1.22436320e+02-43.2815579j
4.89661633e+00 +7.86169804j  2.59301425e+00 +2.52082464j
1.08454022e+00 +0.84921783j  -8.23211495e-01 +0.j          ]
Angles: [ 3.14159265 -2.80180089  1.01374845  0.77128258  0.66430184
3.14159265]
FFT Absolutes: [ 1.42108547e-14  1.29861256e+02  9.26191922e+00  3.61639043
e+00
1.37746085e+00  8.23211495e-01]
fps 11.1111111111
Frequencies: [ 0.  1.11111111  2.22222222  3.33333333  4.44444444  5.55555556]
Frequencies * 60: [ 0.  66.66666667  133.33333333  200.  266.66666667
333.33333333]
Filtered indices: (array([1, 2]),)
Pruned FFT abs: [ 129.86125585  9.26191922]
Pruned frequencies: [ 66.66666667  133.33333333]
Pruned angles: [-2.80180089  1.01374845]
Argmax: 0
66.6666666667

```

---

**Listing A.4:** Python Test Result

## A.2 Screen-shot Samples of Running Program

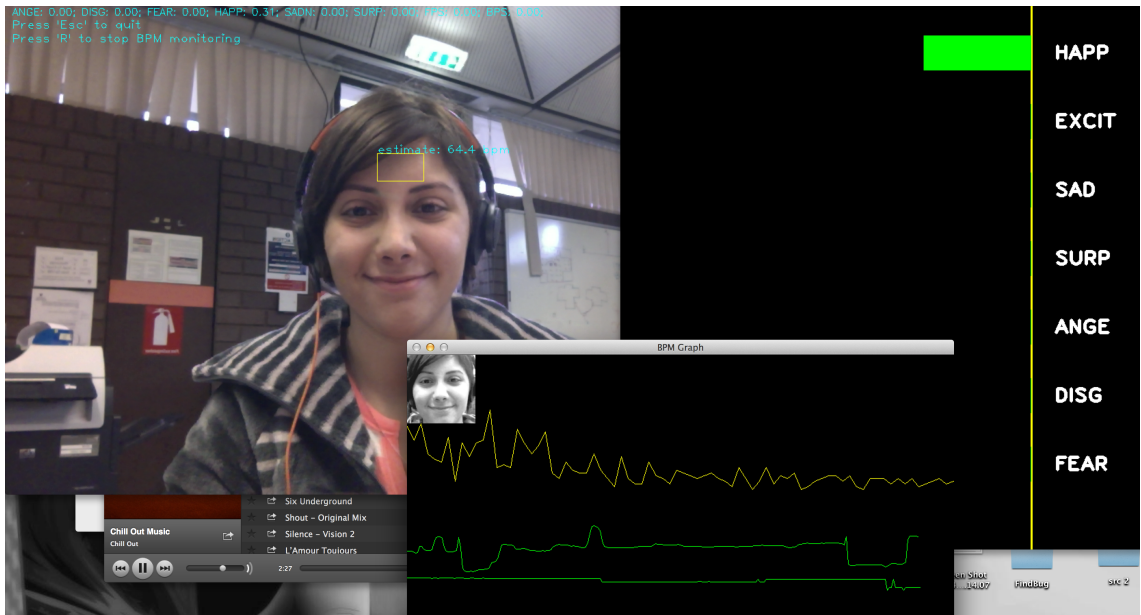
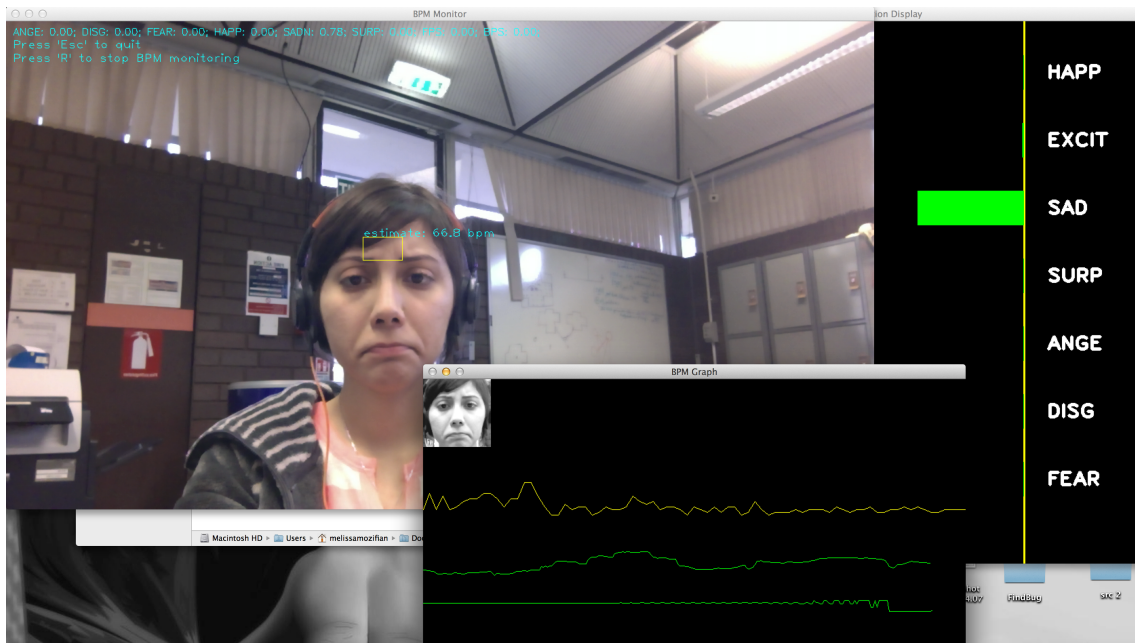


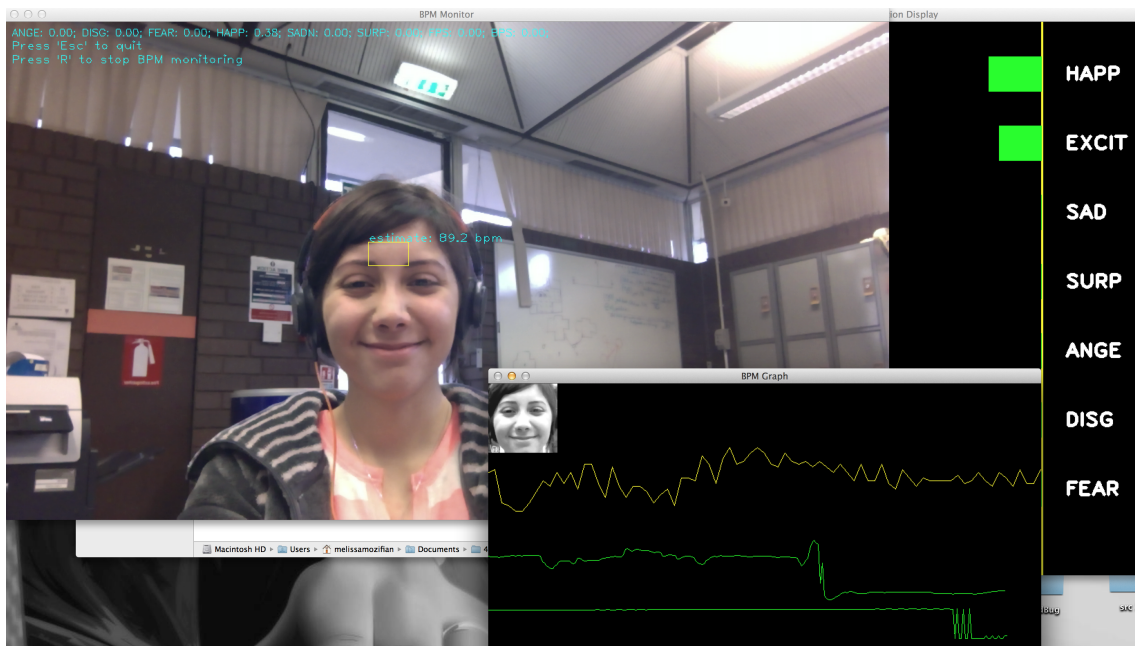
Figure A.1: Detecting happiness.



Figure A.2: Detecting surprise.



**Figure A.3:** Detecting sadness.



**Figure A.4:** Detecting happiness and slight excitement.



Figure A.5: Detecting excitement.



Figure A.6: Detecting neutral.

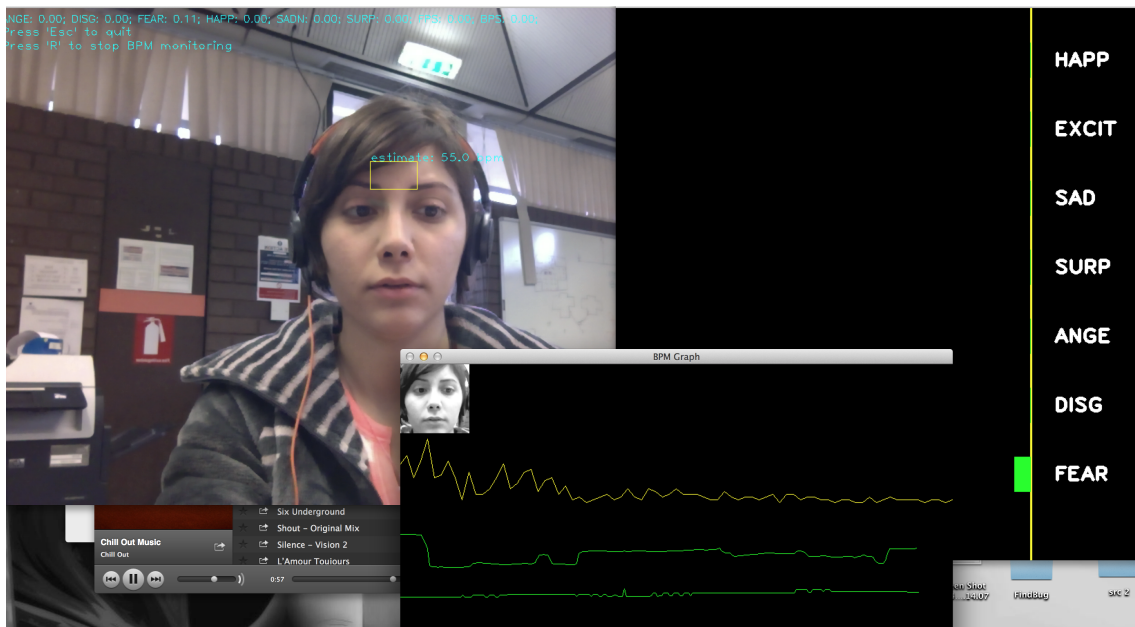


Figure A.7: Detecting fear.



# PULSE DETECTION ALGORITHM

Code snippet of the pulse detection algorithm is shown in Listing B.1.

---

```
1  PU PulseDetector::estimateBPM(const cv::Mat& skin) {
2    _means.push_back(calculate_mean(skin));
3    _times.push_back(timestamp());
4
5    PU pdata;
6    int sampleSize = _means.size();
7    // Check Point
8    assert (_times.size() == sampleSize);
9    // If there are no efficient samples, dont proceed
10   if (sampleSize <= MIN_SAMPLES) {
11       return pdata;
12   }
13   // If there are more samples than required, trim oldest
14   if (sampleSize > MAX_SAMPLES) {
15       list_trimfront(_means, MAX_SAMPLES);
16       list_trimfront(_times, MAX_SAMPLES);
17       list_trimfront(_bpms, MAX_SAMPLES);
18       sampleSize = MAX_SAMPLES;
19   }
20   // FPS
21   _fps = sampleSize / (_times.back() - _times.front());
22   vector<double> even_times = linspace(_times.front(), _times.back(), sampleSize
23       );
24   vector<double> interpolated = interp(even_times, _times, _means);
25   vector<double> hamming = hammingWindow(sampleSize);
26
27   list_multiply_vector(interpolated, hamming);
28   double totalMean = list_mean(interpolated);
29   list_subtract(interpolated, totalMean);
```

```
29
30 // One dimensional Discrete FFT
31 vector<gsl_complex> fftraw = fft_transform(interpolated);
32 vector<double> angles = calculate_complex_angle(fftraw);
33 // Get absolute values of FFT coefficients
34 _fftabs = calculate_complex_abs(fftraw);
35 // Frequencies using spaced values within interval 0 - L/2+1
36 _frequencies = arange((sampleSize / 2) + 1);
37 list_multiply(_frequencies, _fps / sampleSize);
38 // Get indices of frequencies that are less than 50 and greater than 180
39 vector<double> freqs(_frequencies);
40 list_multiply(freqs, 60.0);
41 // Filter out frequencies less than 50 and greater than 180
42 vector<double> fitered_indices = list_filter(freqs, BPM_FILTER_LOW,
43     BPM_FILTER_HIGH);
44 // Used filtered indices to get corresponding FFT values, angles, and
45 // frequencies
46 _fftabs = list_pruned(_fftabs, fitered_indices);
47
48 freqs = list_pruned(freqs, fitered_indices);
49
50 angles = list_pruned(angles, fitered_indices);
51
52 int max = list_argmax(_fftabs);
53
54 _bpm = freqs[max];
55 pdata.bpm = _bpm;
56
57 return pdata;
58 }
```

---

**Listing B.1:** Estimating Pulse



APPENDIX C

# EXPERIMENTAL RESULT

Participant	state	Sex	Age	Weight(kg)	Device Measure(BPM)	Algorithm(BPM)	Error	Sqrt Error
1	Stable	Male	23	66.5	81	82.9	1.9	3.61
					82	83.2	1.2	1.44
					76	76.6	0.6	0.36
	After Exercise	98	82	-16	256			
		87	87.6	0.6	0.36			
		87	87.2	0.2	0.04			
2	Stable	Male	22	85	83	83.8	0.8	0.64
					79	78.4	-0.6	0.36
					81	81.3	0.3	0.09
	After Exercise	102	117	15	225			
		93	93.7	0.7	0.49			
		94	93.7	-0.3	0.09			
3	Stable	Male	21	80	85	74.9	-10.1	102.01
					74	74.3	0.3	0.09
					81	82	1	1
	After Exercise	111	100	-11	121			
		100	101.8	1.8	3.24			
		89	87.7	-1.3	1.69			
4	Stable	Female	23	65	78	76	-2	4
					78	74.1	-3.9	15.21
					80	78.8	-1.2	1.44
	After Exercise	89	78.4	-10.6	112.36			
		86	87	1	1			
		82	82	0	0			
5	Stable	Male	23	82	51	53.9	2.9	8.41
					58	61.8	3.8	14.44
					51	48.6	-2.4	5.76
	After Exercise	63	63.9	0.9	0.81			
		69	57.8	-11.2	125.44			
		63	63.72	0.72	0.5184			
Mean	SD	RMSE	Standard Error					
79.804	14.5701503	5.79338243	2.660133333					

Figure C.1: Experiments Result.

# CHANGES TO THE ORIGINAL SPECIFICATION

## **D.1 Implementing a Face Analysis library from scratch**

This was the main original problem throughout the project. Implementing a sophisticated facial expression analysis, given the time constraint for this project was not possible. Due to unavailability of API for existing software, or significantly high prices for limited use of another set of APIs, it was decided to write a basic library using OpenCV API. We also did take into consideration the possibility of the system not achieving all of its requirements. Fortunately, collaborations with the researchers at the University of Nottingham, enabled us to deploy their system in exchange for extensive testing and bug reports. It was also decided to provide additional support to document the existing code.

## **D.2 Eulerian Video Magnification**

The initial specification, required for EVM [7] to be integrated with our system. However the question was raised whether this could contribute towards further improving the existing algorithms. Having applied EVM and our facial expression analysis on pre-recorded video, caused certain amount of degrade in the accuracy of the AUs. Also further challenges existed in order to make the integration of EVM and our system possible. The EVM implementation is written in Matlab [58]. The integration of Matlab and C++ code was performed using the Matlab Engine API, however it proved to be challenging. Re-implementation of the EVM algorithm in C++ was also considered as an option. Since there was no confirmation whether this could

contribute positively towards our system, the use of this algorithm was abandoned. Instead similar methodology and algorithms taking similar approach was found in the place of EVM.

APPENDIX E

# PROJECT PLAN



# SOFTWARE LISTING

===== Structure of the files =====

The folder "Kanak-Melissa" contains all the code, including the third-party libraries. This folder also contains the private library code used for this project. Please do not distribute this code without the permission of the author. The "Getting started guide.pdf" contains instructions on how to compile and run Kanako's libraries. Since during the development of this project, Kanako was still under development, the latest update was obtained. In order to integrate Kanako with our system, modifications to the code were made.

The modifications were made to the following files :

- lib/KanakoLive.cpp
- include/KanakoLive.h

There are parts of the code which have been commented out. For instance, `KanakoLive::Run(KanakoLive *kl, cv::Mat &frame)` has been further modified to return the data. In the future updates of Kanako, modifications should not be required since it will be a fully implemented API.

The main code developed for this project is included within the folder :

Kanak-Melissa/tools/krealtimetool

- main.cpp : Main entry of the program, this creates an instance of pulse detector component.
- PulseDetector.cpp : This contains most of the code and logic, including OpenCV API to capture video frames, detecting face, calling Kanako, pulse detection algorithm plus graph visualisation for the pulse.

- visualizeEmotion.cpp : Visualisation based on bar intensity, this uses common data between the two components.
- common.h : This includes common global variables and functions among all the other components.
- pulseDetector.h : Pulse detector header files
- visualizeEmotion.h Visualizer header files
- osx/ : Directory containing makefile and running script.

For further instructions and external dependencies refer to the "README.md" file.



# REFERENCES

- [1] OpenCV, “Introduction to support vector machines.” [http://docs.opencv.org/doc/tutorials/ml/introduction\\_to\\_svm/introduction\\_to\\_svm.html](http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html). Accessed March 2014.
- [2] S. Brave and C. Nass, “The human-computer interaction handbook,” in *The human-computer interaction handbook* (J. A. Jacko and A. Sears, eds.), ch. Emotion in Human-computer Interaction, pp. 81–96, Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 2003.
- [3] R. W. Picard, *Affective Computing*. Cambridge, MA, USA: MIT Press, 2000.
- [4] B. R. Platform, “Attention tool D facet module: Facial expression analysis.” <http://imotionsglobal.com/software/add-on-modules/attention-tool-facet-module-facial-action-coding-system-facs/>. Accessed March 2014.
- [5] Emotient, “Facial expression recognition and analysis technologies.” <http://www.emotient.com/>. Accessed March 2014.
- [6] Affectiva, “Affdex facial coding.” <http://www.affdex.com/>. Accessed March 2014.
- [7] H.-Y. Wu, M. Rubinstein, E. Shih, J. Guttag, F. Durand, and W. Freeman, “Eulerian video magnification for revealing subtle changes in the world,” *ACM Trans. Graph.*, vol. 31, pp. 65:1–65:8, July 2012.
- [8] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*. Upper Saddle River, NJ, USA: Pearson, 2011.
- [9] S. S. Jerome Singer, “The schachter and singer page.” <http://www.holah.karoo.net/schachter.htm>. Accessed March 2014.
- [10] R. W. Picard, “Response to sloman’s review of affective computing,” *AI Magazine*, vol. 20, 1999.

- [11] C. Darwin, *The Expression of the Emotions in Man and Animals*. San Francisco, CA, USA: John Murray, 1916.
- [12] D. S. Ann M. Kring, “The facial expression coding system (faces): A users guide.” <http://ist-socrates.berkeley.edu/~akring/FACES%20manual.pdf>.
- [13] Paul Ekman, Wallace V, Friesen Joseph , Hager, “Facial action coding system: The manual.” <http://face-and-emotion.com/dataface/facs/manual/TitlePage.html>, 2002.
- [14] Nedkov, Svetoslav and Dimov, Dimo, “Emotion recognition by face dynamics,” in *Proceedings of the 14th International Conference on Computer Systems and Technologies, CompSysTech '13*, (New York, NY, USA), pp. 128–136, ACM, 2013.
- [15] P. G. S. Rachael E. Jack, Oliver G.B. Garrod, “Dynamic facial expressions of emotion transmit an evolving hierarchy of signals over time,” *Current Biology*, vol. 24, pp. 187–192, 2014.
- [16] Kristen A, Lisa Feldman, Bliss Moreau, James A, “Language and the perception of emotion,” in *Trends in Cognitive Sciences*, vol. 6(1), pp. 125–138, American Psychological Association, 2006.
- [17] P. L. F. Barrett, “What faces can’t tell us.” [http://www.nytimes.com/2014/03/02/opinion/sunday/what-faces-cant-tell-us.html?smid=tw-share&\\_r=1](http://www.nytimes.com/2014/03/02/opinion/sunday/what-faces-cant-tell-us.html?smid=tw-share&_r=1). Accessed March 2014.
- [18] Carnegie Mellon University, “Center for the neural basis of cognition.” Accessed March 2014.
- [19] D. K. Albrecht, “Social intelligence theory.” <https://www.karlalbrecht.com/siprofile/siprofiletheory.htm>, 2004. Accessed March 2014.
- [20] A. Vinciarelli, M. Pantic, and H. Bourlard, “Social signal processing: Survey of an emerging domain,” *Image Vision Comput.*, vol. 27, pp. 1743–1759, Nov. 2009.
- [21] A. S. Pentland, “Socially aware computation and communication,” *Computer*, vol. 38, no. 3, pp. 33–40, 2005.
- [22] M. Weiser, “Ubiquitous computing.” <http://www.ubiq.com/hypertext/weiser/UbiHome.html>, 1996. Accessed March 2014.
- [23] “The smart-its project.” <http://www.smart-its.org/>. Accessed March 2014.

- [24] A Cambridge University Press Journal, "Network science." <http://www.indiana.edu/~netsci/index.html>. Accessed March 2014.
- [25] A. Pentland, *Honest Signals: How They Shape Our World*. Cambridge, MA, USA: MIT Press; New edition edition, 2010.
- [26] R. L. HOTZ, "Emotions vented online are contagious, study finds." <http://online.wsj.com/news/articles/SB10001424052702303546204579435550002436002?mg=reno64-wsj&url=http%3A%2F%2Fonline.wsj.com%2Farticle%2FSB10001424052702303546204579435550002436002.html>. Accessed March 2014.
- [27] Neal Lathia and Veljko Pejovic and Kiran Rachuri and Cecilia Mascolo and Mirco Musolesi and Peter Jason Rentfrow, "Smartphones for large-scale behaviour change interventions.," *IEEE Pervasive Computing*, vol. 12, July-September 2013.
- [28] Fraunhofer, "Shore - sophisticated high-speed object recognition engine." <http://www.iis.fraunhofer.de/en/bf/bsy/produkte/shore.html>. Accessed March 2014.
- [29] H. Mandel, "A new mobile app may read your feelings." <http://www.examiner.com/article/a-new-mobile-app-may-read-your-feelings>. Accessed March 2014.
- [30] R. van Bezooijen, *Characteristics and Recognizability of Vocal Expressions of Emotion*. Netherlands: Mouton de Gruyter, 1984.
- [31] "openear." <http://sourceforge.net/projects/openart/>. Accessed March 2014.
- [32] openSMILE, "The munich versatile and fast open-source audio feature extractor." <http://opensmile.sourceforge.net/>. Accessed March 2014.
- [33] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.
- [34] AI Horizon , "Machine learning, part i: Supervised and unsupervised learning." [http://www.aihorizon.com/essays/generalai/supervised\\_unsupervised\\_machine\\_learning.htm](http://www.aihorizon.com/essays/generalai/supervised_unsupervised_machine_learning.htm). Accessed March 2014.
- [35] G. Shakhnarovich, "Introduction to statistical machine learning." <http://ttic.uchicago.edu/~gregory/courses/wis-ml2011/lect6ho.pdf>. Accessed March 2014.
- [36] GTSVM, "A gpu-tailored approach for training kernelized svms." <http://ttic.uchicago.edu/~cotter/projects/gtsvm/>. Accessed March 2014.

- [37] NVIDIA, “Cuda.” [http://www.nvidia.co.uk/object/cuda\\_home\\_new.html](http://www.nvidia.co.uk/object/cuda_home_new.html). Accessed March 2014.
- [38] NVIDIA, “What is gpu accelerated computing?.” <http://www.nvidia.com/object/what-is-gpu-computing.html>. Accessed March 2014.
- [39] Y. Sun, C. Papin, V. Azorin-Peris, R. Kalawsky, S. Greenwald, and S. Hu, “Use of ambient light in remote photoplethysmographic systems: comparison between a high-performance camera and a low-cost webcam,” *Journal of Biomedical Optics*, vol. 17, no. 3, pp. 037005–1–037005–10, 2012.
- [40] W. Verkruyse, L. O. Svaasand, and J. S. Nelson, “Remote plethysmographic imaging using ambient light.,” *Optics express*, vol. 16, pp. 21434–45, 2008 Dec 22 2008.
- [41] M.-Z. Poh, D. J. McDuff, and R. W. Picard, “Non-contact, automated cardiac pulse measurements using video imaging and blind source separation,” *Opt. Express*, vol. 18, pp. 10762–10774, May 2010.
- [42] W. G. Zijlstra, A. Buursma, and W. P. Meeuwse-van der Roest, “Absorption spectra of human fetal and adult oxyhemoglobin, de-oxyhemoglobin, carboxyhemoglobin, and methemoglobin.,” *Clin Chem*, vol. 37, no. 9, pp. 1633–8, 1991.
- [43] OpenCV, “haarcascades - trained classifiers.” <https://github.com/Itseez/opencv/tree/master/data/haarcascades>. Accessed March 2014.
- [44] Itseez, “Open source computer vision.” <http://opencv.org/>, 2014. Accessed March 2014.
- [45] C++ Standards Committee Library Working Group, “Boost.” <http://www.boost.org/>, 2014. Accessed March 2014.
- [46] T. G. O. System, “Gsl - gnu scientific library.” <http://www.gnu.org/software/gsl/>. Accessed March 2014.
- [47] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, pp. 511–518, 2001.
- [48] D. L. Baggio and S. Emami and D. M. Escriva and K. Ievgen and N. Mahmood and J. Saragih and R. Shilkrot, *Mastering OpenCV with Practical Computer Vision Projects*. Packt Publishing, Limited, 2012. recommended: advanced OpenCV project support/examples inc. iOS and Android examples.

- [49] N. developers, “Numpy.” <http://www.numpy.org/>. Accessed March 2014.
- [50] S. developers, “Scipy.” <http://www.scipy.org/>. Accessed March 2014.
- [51] Tristan H, “Webcam-pulse-detector.” <https://github.com/thearn/webcam-pulse-detector>. NASA Glenn Research Center.
- [52] Boost, “Chapter 5. boost.chrono 2.0.2.” [http://www.boost.org/doc/libs/1\\_55\\_0/doc/html/chrono.html](http://www.boost.org/doc/libs/1_55_0/doc/html/chrono.html). Accessed March 2014.
- [53] SciPy, “numpy.hamming.” <http://docs.scipy.org/doc/numpy/reference/generated/numpy.hamming.html>. Accessed March 2014.
- [54] Wolfram MathWorld, “Discrete fourier transform.” <http://mathworld.wolfram.com/DiscreteFourierTransform.html>. Accessed March 2014.
- [55] T. Selker, “Coach: A teaching agent that learns,” *Commun. ACM*, vol. 37, pp. 92–99, July 1994.
- [56] D. Lee, “Steering wheel camera detects angry drivers.” <http://www.bbc.co.uk/news/technology-26549273>. Accessed March 2014.
- [57] N. Negroponete, “Talking with computers.” <http://www.global-media.org/neome/docs/PDF's/02%20-%20other/alle%20artikelen%20van%20Negroponte.pdf>. Accessed March 2014.
- [58] Mathworks, “Matlab.” <http://www.mathworks.co.uk/products/matlab/>. Accessed March 2014.