

8-2014

MULTISENSORY EMOTION RECOGNITION WITH SPEECH AND FACIAL EXPRESSION

Jacob P. Roeland

Follow this and additional works at: http://aquila.usm.edu/honors_theses



Part of the [Software Engineering Commons](#)

Recommended Citation

Roeland, Jacob P., "MULTISENSORY EMOTION RECOGNITION WITH SPEECH AND FACIAL EXPRESSION" (2014).
Honors Theses. Paper 259.

This Honors College Thesis is brought to you for free and open access by the Honors College at The Aquila Digital Community. It has been accepted for inclusion in Honors Theses by an authorized administrator of The Aquila Digital Community. For more information, please contact Joshua.Cromwell@usm.edu.

The University of Southern Mississippi

MULTISENSORY EMOTION RECOGNITION WITH SPEECH AND FACIAL
EXPRESSION

by

Jacob Roeland

A Thesis
Submitted to the Honors College of
The University of Southern Mississippi
in Partial Fulfillment
of the Requirements for the Degree of
Bachelor of Science
in the School of Computing

August 2014

Approved by

Scott Neal, M.S., Thesis Adviser
Instructor of Information Technology

Andrew Sung, Ph.D., Director
School of Computing

Ellen Weinauer, Ph.D., Dean
Honors College

Abstract

Computers through both desktop and mobile devices are only becoming more important in our lives leading us to have more involved and longer interactions with them. Because of this our brains actually classify our involvement with them in a manner similar to our interactions with our fellow humans. This can lead to frustration and anxiety when our computers interrupt our work or pleasure with contextually inappropriate messages, much the same way it would if a friend or co-worker was pushy or rude.

A way to solve this issue is to give our machines emotional intelligence, or the ability to recognize and be aware of our emotions. While monitoring physiological symptoms such as skin conductivity and muscle tension is one of the most accurate ways of detecting emotions, it can also be done in a more physically and socially comfortable manner by way of visual and auditory clues.

This thesis will create a bimodal system where input is visual information via a still image and auditory information via a clip of human speech. The system will use two existing programs to identify the emotion found in each and, by using a weighted system, return the singular emotion felt.

Key Words: emotion recognition, affective computing, openSMILE, openEAR, Weka,
Human Emotion Detection from Image

Table of Contents

List of Figures	vi
List of Tables	vii
Chapter 1 – Introduction	1
Chapter 2 - Literature Review	4
Section 2.1: Affective Computing	4
Section 2.2: Detecting Emotion Using Physiological Symptoms	5
Section 2.3: Detecting Emotion Using Facial Expressions	8
Section 2.4: Detecting Emotion Using Facial Expressions	10
Chapter 3 – Methodology	12
Section 3.1: Introduction of Components Used	12
Section 3.2: Human Emotion Detection from Image	13
Subsection 3.2.1: Porting to Linux	13
Subsection 3.2.2: Removing GUI Elements	15
Subsection 3.2.3: Command-line	17
Subsection 3.2.4: Ramifications of Porting	17
Section 3.3: openSMILE/openEAR and Weka	19
Subsection 3.3.1: Setting up the Corpus	19
Subsection 3.3.2: Building the Model	20
Subsection 3.3.3: Classifying with Weka	22
Section 3.4: Merging the Databases	24
Chapter 4 - Conclusion	29
References	30

List of Figures

Figure 3.2.1.1: Start screen of Human Emotion Detection.	13
Figure 3.2.1.2: How the database was queried using Access.	14
Figure 3.2.1.3: How the database was queried after switching to MySQL.	14
Figure 3.2.1.4: Successfully detecting an emotion (or rather a lack of one).	15
Figure 3.2.3.1: Detecting an emotion using only the command-line.	17
Figure 3.3.1.1: Number of audio files for each emotion in the corpus.	20
Figure 3.3.2.1: Building the openSMILE model using the Berlin Database of Emotional Speech.	21
Figure 3.3.2.2.: Build complete and copying appropriate files to work directory.	21
Figure 3.3.3.1: Diff of changes made to config file.	22
Figure 3.3.3.2: Generating the <code>output.arff</code> file to be used by Weka.	23
Figure 3.3.3.3: Classifying the emotion using Weka.	23
Figure 3.3.3.4: Generating another <code>output.arff</code> and classifying.	24
Figure 3.4.1: Python script that runs both the Human Emotion Detection and openSMILE programs at once and outputs the results.	25
Figure 3.4.2: Output of Python script.	26
Figure 3.4.3: Final draft of Python script.	27
Figure 3.4.4: Output of Python script with debugging on.	28
Figure 3.4.5: Output of Python script when training with debugging on.	28

List of Tables

Table 2.2.1: How the subject in Picard et al's experiment described each emotion.	6
Table 2.2.2: How each emotion was classified by the system.	8
Table 3.2.4.1: Accuracy dropped after porting Human Emotion Detection to Linux.	18

Chapter 1 - Introduction

When we think of computers, we usually think of them as cold, calculating machines. We use them for tasks ranging from banking to sharing pictures of our meals with friends to landing a rover on Mars with a parachute 220 million miles away. Each day brings more and more opportunities for computers and artificial intelligence engines to better our lives. Yet for all the benefits computers bring us, they are dumb machines. They can only do what they have been told to do, no matter the circumstances. They ask to be restarted when the user is typing an important paper. They fail to complete a task done a thousand times before and give the user no understandable reason for the failure. They bombard us with software updates for applications that we have never used. All of these examples (and many, many more) obviously annoy us. They frustrate us with the seemingly sheer stupidity of their requests.

One way to make computers seem smarter is by giving them the ability to recognize emotions, or emotional intelligence. One may believe that emotions are strictly a human quality, an aspect of our humanity that really has no need to be programmed into our machines. And this would be true if we did not also interact with our machines as we do with humans. A theory from Stanford suggests that when interacting with something, be it man or machine, humans tend to still expect a human-to-human experience. If another human talks to you but never listens to you, that human is found to be annoying; it is similar with a computer (Reeves & Nass, 1996). Thus, it is desired for our machines to understand what we are feeling as they become more embedded in our lives.

Emotional intelligence can be defined as “the ability to recognize, express, and have emotions, coupled with the ability to regulate these emotions, harness them for constructive purposes, and skillfully handle the emotions of others ” (Picard, Vyzas, & Healey, 2001). It can be easily argued that machines may never require the ability to actually “have” or “regulate” emotions. Recognizing them, however, can be particularly useful. A computer could learn, for example, if Microsoft Word is open and the user is seen to be concentrating, then that user should not be interrupted for non-important reasons. How then could a computer learn when to interrupt and when not to? A human would learn over the course of their entire interactions with another person when to interrupt them and when not to. Someone failing to do this would be considered rude or arrogant. Similarly, a computer would learn this by watching the reaction from the user after the computer issues the interruption. If it is a negative response, the computer will note that that was a bad time. So a system must be created where a computer can monitor the signs of an emotion and then detect which emotion it is.

Emotion plays a role in nearly all human communication. According to Picard, et al., it affects “word choice, tone of voice, facial expression, gestural behaviors, posture, skin temperature and clamminess, respiration, muscle tension, and more.” (2001) We tend to think that studying faces is the best method of determining emotion as it is usually the most pronounced. However, these are also the easiest to fake. The most accurate detection would combine multiple sensors monitoring facial muscle tension, respiration rate, skin conductivity, etc. with computational reasoning and natural language processing. But using a computer with several electrodes gelled to one's face and a Hall effect respiration sensor strapped around one's diaphragm is not physically nor socially

comfortable. However, as technology improves with Moore's law, these sensors should become smaller and smaller, and affective computing will be much more convenient.

One may wonder why would it not be easier to simply use a camera to monitor facial features and a microphone to monitor speech and tone of voice as isn't that how humans do it? It's naïve to believe that humans do not also recognize other physiological emotional signs.

A stranger shaking your hand can feel its clamminess (related to skin conductivity); a friend leaning next to you may sense your heart pounding; students can hear changes in a professor's respiration that give clues to stress; ultimately, it is muscle tension in the face that gives rise to facial expressions. (Picard et al., 2001)

A system is therefore proposed that can use still images of a person's face and captured audio to detect what the user is currently feeling with a certain degree of accuracy. Both the openSMILE and "Human Emotion Detection from Image [*sic*]" databases will be set-up on a Linux machine with a Python script and MySQL database bridging the gap. The script will be given the file locations of an image and audio clip as arguments and give as output the detected emotion.

Chapter 2 - Literature Review

Section 2.1: Affective Computing

Affect recognition is a hard problem to solve. Even humans can at times misunderstand the emotion coming from another human, so it would be a mistake to assume that there is a perfect way (especially with arguments still being made about what emotions are exactly). To be effective, the algorithm only has to match a human's ability to recognize emotions. In non-emotive speech, humans are about 60% accurate at identifying an emotion, with computers matching or slightly beating that. This is dependent of course on the accuracy of the speech recognition itself; computers are now around 90% accurate on neutral speech and only 50-60% accurate on emotional speech. And in understanding what emotion is being expressed in speech, it is imperative that we “[recognize] what is said as well as how it was said.” (Picard et al., 2001)

In processing speech, sound files are fed into a “feature extraction script” which “extracts the features that represent global statistics.” It is then normalized to filter out the sensor noise and outliers. After that, the data is compared against mixture models to determine the emotion being expressed. The data and results are then used to train the model to become more accurate over time (Ramakrishnan, 2012).

Recognizing emotions via the face is easier for humans with 70-98% accuracy when choosing from six emotions. Computers detect expressions with an 80-98 degree accuracy when selecting from five to seven emotions. Some research has focused more on detecting so called “facial phenoms” or these minute facial movements that when combined form all human expression (Picard et al., 2001). A person's image is captured

using a video device and passed into a facial tracking system. This system looks for prominent facial features such as the eyebrows, eyes, lips, etc. Based on the locations of these organs, an emotion is determined using a model (Ramakrishnan, 2012).

The high percentages of detection can be deceiving. These machines are recognizing elements from exaggerated actions. The latest technology in these fields is comparable to where speech recognition was decades ago where a computer could detect the carefully articulated digits zero through nine with pauses separating each but could not detect them in natural speech. Emotional research is particularly difficult because defining emotion is hard. “[A]fter over a century of research, emotion theorists still do not agree upon what emotions are or how they are communicated.”(Picard et al., 2001)

There have been numerous studies and much research that attempted to do emotion processing using general affect data, that is features that occur in a large percentage of the population. Even then, actually defining a particular facial expression as a singular emotion is difficult. One person's set of phenoms that determine her “Romantic love” expression may be “Platonic love” in another. Or someone's “Hate” may share many qualities with another's “Anger”.

Section 2.2: Detecting Emotion Using Physiological Symptoms

Picard instead designed an experiment with only one person and that one person's data based on the definition problem. She also excluded any data collection from visual or auditory sources, instead opting for physiological symptoms. Each morning the subject, a graduate student, would arrive at her office and be outfitted with several sensors. These included an electromyogram that recorded facial tension, a blood pressure

monitor, a skin conductance sensor, and a respiration sensor. She also had a small pressure sensor to help with sustaining the emotion.

The researchers collected data for eight emotions: No emotion, Anger, Hate, Grief, Platonic love, Romantic love, Joy, and Reference. Before the experiment began, the subject recorded specific imagery, definitions, arousal levels, and valence level or how positive or negative the emotion is. The subject's table is shown below.

Emotion	Imagery	Description	Arousal	Valence
(N)o Emotion	blank paper, typewriter	boredom, vacancy	low	neutral
(A)nger	people who arouse rage	desire to fight	very high	very negative
(H)ate	injustice, cruelty	passive anger	low	negative
(G)rief	deformed child, loss of mother	loss, sadness	high	negative
(P)latonic love	Family, summer	happiness, peace	low	positive
Romantic (L)ove	Romantic encounters	excitement, lust	very high	positive
(J)oy	The music “Ode to Joy”	uplifting happiness	medium high	positive
(R)everence	church, prayer	calm, peace	very low	neutral

Table 2.2.1: How the subject in Picard et al's experiment described each emotion.

Each session lasted roughly half-an-hour giving the researchers 28-33 thousand samples per sensor. Of the 30 days they collected data, one or more of the sensors failed on about every third day. They formed two data sets. Data Set I, assembled before the 30 day experiment was completed, was formed as follows:

Data segments of 2,000 samples (100 seconds) in length were taken from each of the signals ... for each of the eight emotions, on each of the 19 days where there were no failures in these segments of data collection.

The 2,000 samples were taken from the end of each emotion segment to avoid the transitional onset where the subject was prompted to move to the next emotion. A 20th day's data set was created out of a combination of partial records in which some of the sensors had failed (Picard et al., 2001).

Data Set II also contained data from 20 days that none of the sensors failed, including data of 16 days from Data Set I. From each day, they used all the samples available for each emotion, including those from the transitional period. Data Set II resulted in a 10% gain in performance. Using Sequential Floating Forward Search and Fisher Projections, they developed algorithms for processing the data. Accuracy then increased by at least 33% over random guessing for all eight emotions. For a subset of three (anger, joy, and reverence), accuracy increased 50% (with a confidence level of > 99.99 percent). A copy of their final classification results data is shown below. The row headers show the emotion felt by the subject while the column headers display what it was classified as (Picard et al., 2001).

	N	A	H	G	P	L	J	R	Total
(N)eutral	17	0	0	0	3	0	0	0	20
(A)nger	0	17	0	0	2	1	0	0	20
(H)atred	0	0	14	1	0	0	3	2	20
(G)rief	0	0	1	15	0	0	4	0	20
(P)latonic Love	0	0	0	0	17	2	1	0	20
Romantic (L)ove	1	1	0	0	3	14	1	0	20
(J)oy	0	0	1	2	0	0	17	0	20
(R)everence	0	0	0	1	0	0	0	19	20
Total	18	18	16	19	25	17	26	21	160

Table 2.2.2: How each emotion was classified by the system.

Section 2.3: Detecting Emotion Using Facial Expressions

Matthew S. Ratliff and Eric Patterson (2008) worked on creating a method of recognizing emotion from facial expressions using active appearance models (AAM) and still images. An AAM is a method of matching images to a model using landmarks that appear in each image. For their testing data, they used the

publicly available facial expression database “FEEDTUM”. “This database contains still images and video sequences of eighteen test subjects, both male and female, of varying age.” Instead of using actors to act out the emotions, the database curators tried to generate the emotions from subjects organically by showing a series of videos designed to elicit a particular emotion.

To begin, Ratliff and Patterson assigned a score to each of the 500+ images in the database based on clarity (“the clarity of the emotional content”), sincerity (“how well the subject conveys the intended emotion”), and how much the subject’s head moved. For example, Subject 2 had a sincerity score of 4/10 and a clarity score of 7/10 with no movement leading to an overall score of 7.5. Low-scoring images were not used to train the model; additionally, images from three subjects were rejected altogether due to facial obstructions such as hair or eyewear (2008).

Based on the images, 113 landmarks were chosen for the model. These included the “brow, eyes, mouth, and nasio-labial [*sic*] region as formed by the underlying muscles” as well as the general outline of the face. To determine an emotion, distances between landmarks were compared to mean distances found in the model, classifying it as one of six emotions based on these distances. The system correctly classified each subject between 60% and 100% of the time, with most being in the 80-90% range (Ratliff & Patterson, 2008).

Section 2.4: Detecting Emotion Using Facial Expressions

Björn Schuller, Gerhard Rigoll, and Manfred Lang (2004) worked on creating a hybrid system using both acoustic and linguistic features to recognize a specific emotion. For their speech corpus, “German and English sentences of 13 speakers, one female, were assembled”; this corpus was used for both training and evaluating both halves of the system. The team focused on recognizing anger, disgust, fear, joy, sadness, surprise, and neutral speech. In classifying the speech acoustically, they relied on 33 features, such as the standard deviation of pitch and the rate of voiced sounds. After testing various classifiers, they decided to use Support Vector Machines (SVM). An SVM is a method of classifying in which data is grouped into one of two categories. Schuller, et al. used three passes to classify a speech utterance as an emotion. For example, a speech utterance that was fearful would be classified as “anger, neutral, fear, joy” then as “fear, joy”, then finally as “fear”. Using this classifier saw error rates as high as 7% when using a corpus of a single speaker and as high as 24% when using multiple speakers.

To classify the speech linguistically, a standard speech recognition algorithm was used followed by each word being categorized in a Belief Network. For example, in the sentence “I do not feel too good at all”, the word “good” would be classified as positive until the word “not” negates it. And the “too” would classify how badly the speaker felt. Finally, the two classifications are fed into a neural network that takes into account 14 dimensions consisting of “7

confidences of each the acoustic, and linguistic analysis.” The overall system has an error rate of up to 8%.

Chapter 3 – Methodology

Section 3.1 Introduction of Components Used

Instead of creating another system for detecting emotion, two existing programs were instead combined to create a single new system. openSMILE ("SMILE is an acronym for Speech and Multimedia Interpretation by Large-space Extraction") is an open-source audio feature-extraction program (Eyben, F., Weninger, F., Gross, F., & Schuller, B., 2013). openEAR is a toolkit that combines openSMILE with some related tools and sample scripts (Eyben, F., Wollmer, M., & Schuller, B., 2009). Both were developed at the Technische Universität München. Weka is an open-source Java program containing a set of "machine learning algorithms for data mining tasks." (Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H., 2009) It is used for the processing and analyzing of any-size data sets. "Human Emotion Detection from Image [*sic*]" is a C# program that uses the shape and distance of the eyes and mouth to predict the emotion displayed (shakil0304003, 2010).

Early on the decision was made to develop this system in a Linux environment since we have a more detailed knowledge of the Linux command-line than Windows, a necessity since openSMILE and openEAR do not have a graphical interface. That coupled with the fact that the Mono project allows .NET frameworks to be built and ran in a Linux environment, allowed us to run all necessary components in the same environment without resorting to work-arounds like Wine (a software application that lets Windows applications run in Unix-like environments).

Section 3.2: Human Emotion Detection from Image

Subsection 3.2.1: Porting to Linux

We began by loading the solution into MonoDevelop, an open-source integrated development environment (IDE) designed to help build C# and .NET projects in Mono.

When we first compiled with it as is, the project built with no errors (Figure 3.2.1.1).

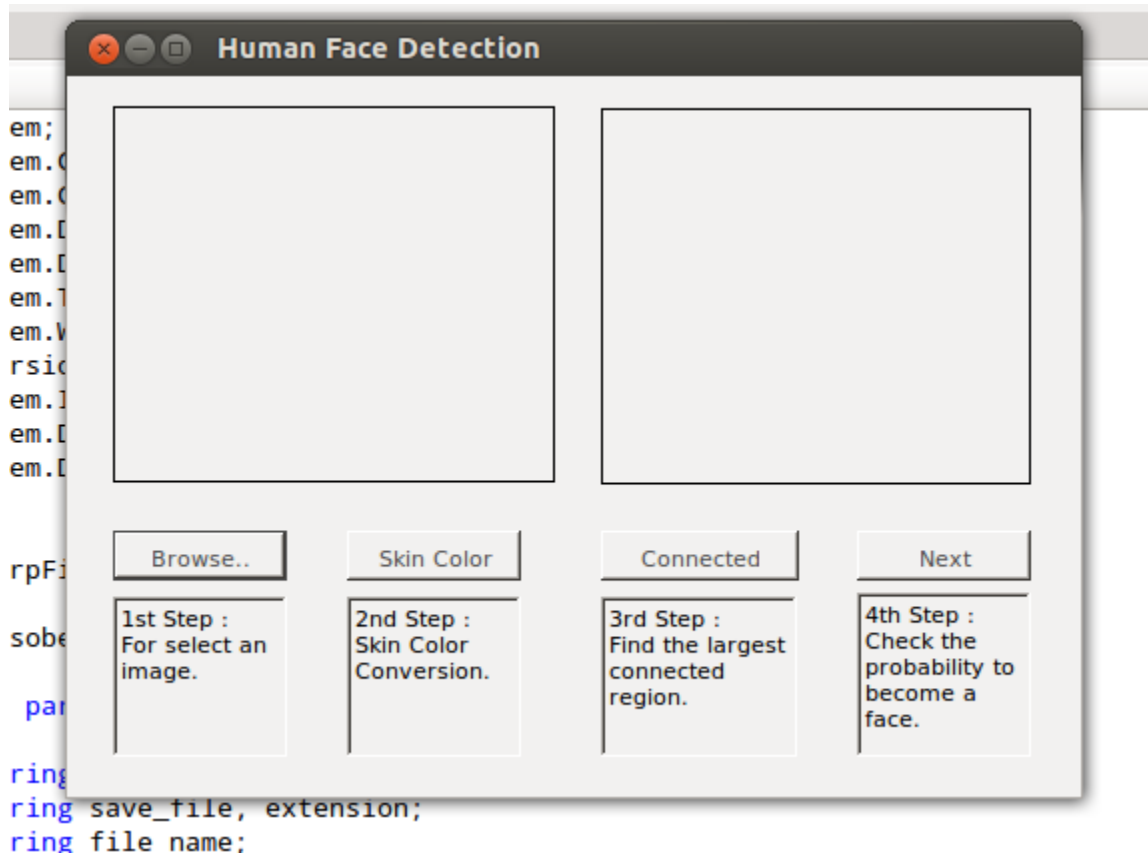


Figure 3.2.1.1: Start screen of Human Emotion Detection.

Without code modification, it worked until we attempted to access the Access database where it died due to missing libraries, specifically libGDA. After installing the libraries, the package died again trying to accomplish the same task but gave a different error message. We then discovered that Mono has stopped supporting the

System.Data.OleDb provider and instead suggests System.Data.Odbc (OLE DB). We edited the code to use the new provider resulting in minimal work.

After failing to access the database a third time, we decided to change our database provider to use MySQL since it is more supported in Linux than Access is. On a Windows install, we exported the Access database to MySQL SQL using a third-party tool and imported it to our MySQL server (BullZip, 2013). We edited the connection strings and query commands to reflect this change. Figures 3.2.1.2 and 3.2.1.3 show the before and after of such code changes.

```
OleDbConnection myConnection = new OleDbConnection();
OleDbCommand myCommand = new OleDbCommand();
string connectionString = @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + _dataBasePath;
string query = "select * from [Position]";
myConnection.ConnectionString = connectionString;
myConnection.Open();
myCommand.Connection = myConnection;
myCommand.CommandText = query;

OleDbDataReader reader;
reader = myCommand.ExecuteReader();
```

Figure 3.2.1.2: How the database was queried using Access.

```
string connectionString = "Server=localhost;Database=movedb;Uid=root;Pwd=123456;";
IDbConnection dbcon;
dbcon = new OdbcConnection(connectionString);
dbcon.Open();
IDbCommand dbcmd = dbcon.CreateCommand();
string query = "select * from Position";
dbcmd.CommandText = query;
IDataReader reader = dbcmd.ExecuteReader();
```

Figure 3.2.1.3: How the database was queried after switching to MySQL.

When built this time, the code ran as expected and emotion was detected (Figure 3.2.1.4).

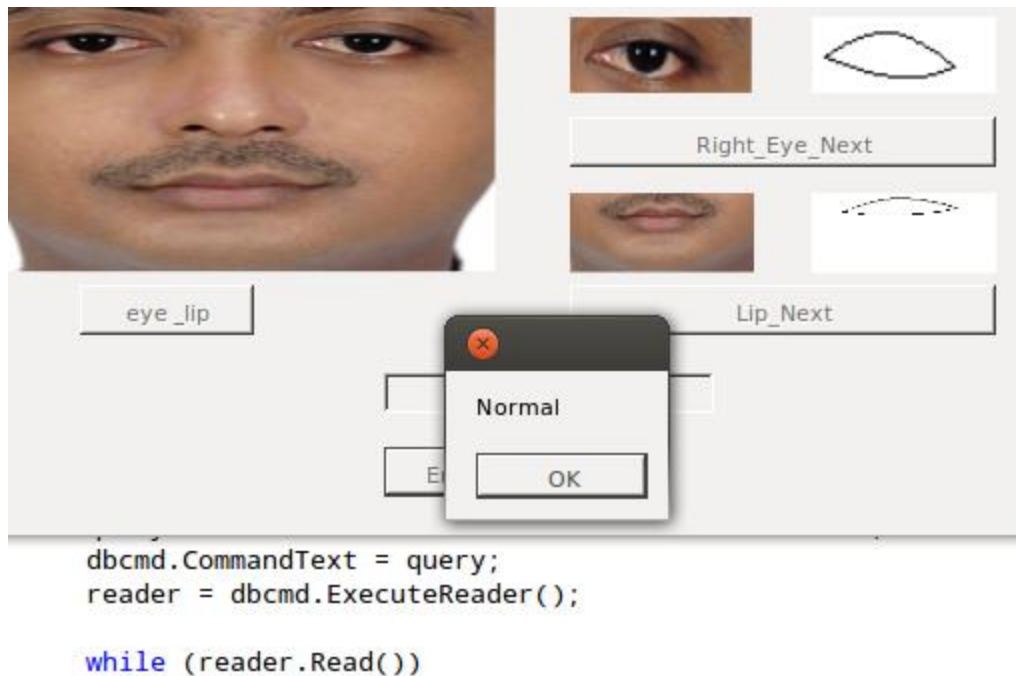


Figure 3.2.1.4: Successfully detecting an emotion (or rather a lack of one).

Subsection 3.2.2: Removing GUI Elements

Because we'll be accessing this tool from another program, this must be able to be used solely from the command-line. As it currently stands, the tool requires significant user interaction; two windows and at least 21 clicks are required to get from selecting an image to receiving the result. We began stripping out GUI elements and making the tool more automated.

We started by focusing on Form 1, the first window that appears when running the program. First, we changed the "Browse..." button functionality by hard-coding an image location; later this will be replaced to use a path specified by a command-line argument. The code from the other buttons ("Skin color", "Connected", and "Next") were appended to the "Browse..." button. Now when the button is pressed, the image is loaded to the picturebox, the image is contrasted, the largest connected region is selected, and

the next form (Form 2) is loaded with the selected image. Once it was shown that these changes do not affect the program negatively, we began removing the code included with the vestigial buttons, taking care to test the code regularly. When finished, an empty Form 1 appears followed immediately by Form 2 loaded with the processed image.

Following a similar process, we begin by making “Left_Eye_Next”, “Right_Eye_Next”, and “Lip_Next” work by only clicking once, instead of the three to four clicks they currently require. Next, we placed the code required in those three buttons inside the “eye_lip” button, continuing backward until all image processing is done when “Binary Image” is pressed. At this point, we leave the code that selects the emotion (located inside “Emotion”) alone.

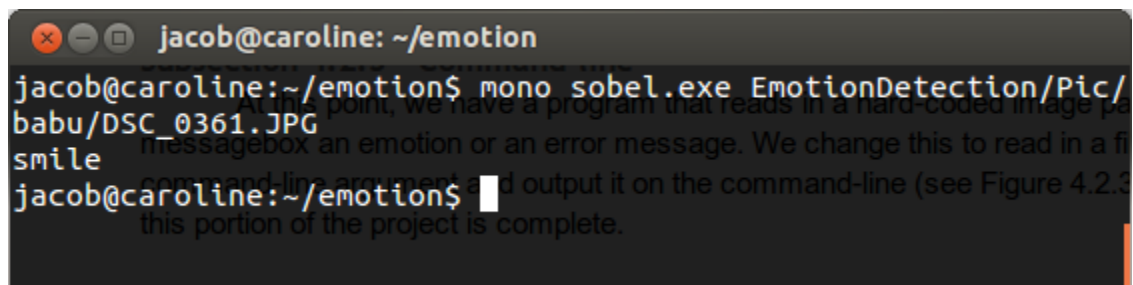
It was then decided that instead of trying to edit the forms and hoping that each instance of each reference to the GUI elements is changed accordingly, we slowly built a new class that recycles the code from the forms but with no reference to the elements declared. This allowed the code to break fast so it can be fixed quickly and allowed us to keep a working copy of the original. We began copying only the code from our “Browse...” in Form 1 to a new class called “primary”. When built, it obviously failed due to missing functions and references to GUI elements that do not exist, but we quickly added these functions from Form 1 and edited those references to point to image classes instead of pictureboxes.

Soon we have one class that does everything from initially loading the image to outputting the emotion. Immediately we discover that accuracy has dropped severely. When we examined the code, it was discovered that code from Form 1 uses class variables that have the same name as variables from Form 2. We decided to save time

and simply build two classes separated as it was in the original codebase. After doing so, accuracy returned.

Subsection 3.2.3 - Command-line

At this point, we had a program that reads in a hard-coded image path and outputs an emotion or an error message via a messagebox. We changed this to read in a file from a command-line argument and output it on the command-line (see Figure 3.2.3.1). With that done, this portion of the project is complete.

A terminal window with a dark background and light text. The title bar shows window control buttons and the text 'jacob@caroline: ~/emotion'. The terminal content shows a command prompt 'jacob@caroline:~/emotion\$' followed by the command 'mono sobel.exe EmotionDetection/Pic/babu/DSC_0361.JPG'. The output of the command is 'smile' on the next line. The prompt 'jacob@caroline:~/emotion\$' is visible again on the third line, with a cursor at the end.

```
jacob@caroline: ~/emotion
jacob@caroline:~/emotion$ mono sobel.exe EmotionDetection/Pic/
babu/DSC_0361.JPG
smile
jacob@caroline:~/emotion$
```

Figure 3.2.3.1: Detecting an emotion using only the command-line.

Subsection 3.2.4 - Ramifications of Porting

It should be noted that after porting this project to Linux and MySQL, the accuracy of the project as a whole dropped ~25%. It is unclear whether this is from the way Mono builds the solution, the way it uses MySQL, or some other mitigating factor. Uncovering why this issue occurs is beyond the scope of this project but future research should be done.

Table 3.2.4.1 shows some of the results from running the program on Windows 7 and using an Access database and on Ubuntu 12.04 using MySQL. These images can be found in the `Pic/babu/` directory included with the package.

	Actual emotion	Windows 7	Ubuntu 12.04
Image used			
1.JPG	Surprise	Surprise	Normal
2.JPG	Smile	Smile	Normal
11.JPG	Normal	Normal	Normal
dg 21774 (6).JPG	Sad	Sad	Sad
dg 21774 (7).JPG	Normal	Normal	Normal
dg 21774 (8).JPG	Smile	Normal	Smile
DSC_0361.JPG	Smile	Smile	Smile
Sample accuracy		85.71%	71.43%
Total accuracy		93.55%	67.74%

Table 3.2.4.1: Accuracy dropped after porting Human Emotion Detection to Linux.

Section 3.3: openSMILE/openEAR and Weka

Subsection 3.3.1: Setting up the Corpus

Before using the openSMILE library, a corpus of audio files must be set up. These audio files will be used by openSMILE and a script from openEAR to generate a model that Weka can then use to classify the emotion. We used the Berlin Database of Emotional Speech (Burkhardt, F., Paeschke, A., Rolfes, M., Sendlmeier, W. F., & Weiss, B., 2005). It features “[t]en actors (5 female and 5 male) simulated the emotions, producing 10 German utterances (5 short and 5 longer sentences) which could be used in everyday communication and are interpretable in all applied emotions.” It consists of 535 WAV audio files in one folder with the following naming scheme: [speaker][spoken text][emotion][version]. For example, the file named “08b10Fd” specifies speaker 08 (a 34-year-old female), text b10 (“Die wird auf dem Platz sein, wo wir sie immer hinlegen.” Translated: “It will be in the place where we always store it.”), happiness (“F” stands for “Freude”, German for “happiness”), and version D (Burkhardt, F., Paeschke, A., Rolfes, M., Sendlmeier, W. F. & Weiss, B., 2013).

For openEAR to build the model, we must separate each of the files into folders that specify the expressed emotion; all files for “anger” must go in an “anger” directory, all files for “boredom” go in a “boredom” directory, etc. Figure 3.3.1.1 shows the directory structure and the number of files in each.

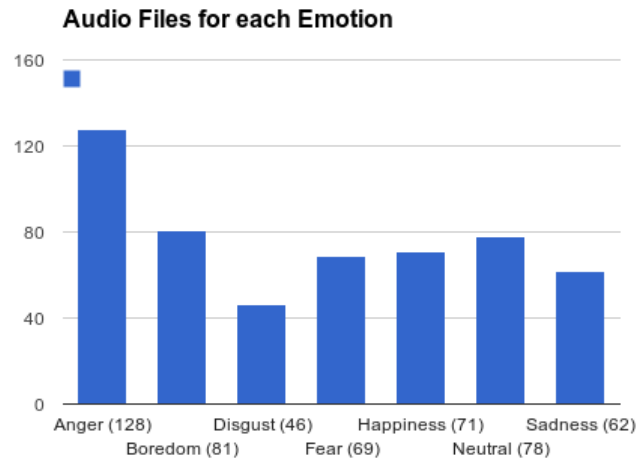


Figure 3.3.1.1: Number of audio files for each emotion in the corpus.

Subsection 3.3.2: Building the Model

With this done, a model can now be built. We started by navigating to the openEAR model training directory. We executed the Perl script `makemodel.pl`, passing in the root directory of the corpus and one of the configuration files located in `config/`; in this case the “emobase” configuration file will be used. Figure 3.3.2.1 demonstrates the executed commands and the beginning output. After completion a few errors are displayed, but these can be safely ignored.

```

jacob@caroline:~/test/openEAR$ cd scripts/modeltrain/
jacob@caroline:~/test/openEAR/scripts/modeltrain$ perl makemodel.pl ../../corpus/ emobase.conf
-- Corpus mode --
Running feature extraction on corpus ' ../../corpus/' ...
label: anger
label: boredom
label: disgust
label: fear
label: happiness
label: neutral
label: sadness
anger
../../corpus//anger/03a01Wa.wav... OK.
../../corpus//anger/03a02Wb.wav... OK.
../../corpus//anger/03a02Wc.wav... OK.
../../corpus//anger/03a04Wc.wav... OK.
../../corpus//anger/03a05Wa.wav... OK.

```

Figure 3.3.2.1: Building the openSMILE model using the Berlin Database of Emotional Speech.

A new directory `work/` was created. Within, a Weka ARFF file was generated: `emobase.arff`. We can now use Weka and this file to generate the model proper. We copied the file to our main work directory. We are finished with openEAR so after copying the Linux binary `SMILEExtract` and the configuration file `emobase.conf` to our work directory, we deleted openEAR (Figure 3.3.2.2).

```

../../corpus//sadness/16b10Tb.wav... OK.
../../corpus//sadness/16b10Td.wav... OK.
Converting arff to libsvm feature file (lsvm) ...
Training libsvm model ...
sh: 1: libsvm-small/svm-scale: not found
building an SVM CLASSIFICATION model...
cp: cannot stat `work//emobase.scale': No such file or directory
cp: cannot stat `work//emobase.model': No such file or directory
jacob@caroline:~/test/openEAR-0.1.0/scripts/modeltrain$ cp work/emobase.arff ../../
jacob@caroline:~/test/openEAR-0.1.0/scripts/modeltrain$ cd ../../
jacob@caroline:~/test$ cp openEAR-0.1.0/SMILEExtract .
jacob@caroline:~/test$ cp openEAR-0.1.0/config/emobase.conf .
jacob@caroline:~/test$ rm -rf openEAR-0.1.0/
jacob@caroline:~/test$

```

Figure 3.3.2.2: Build complete and copying appropriate files to work directory.

We used Weka to build the model with the following command: `java -classpath weka.jar weka.classifiers.trees.J48 -t emobase.arff -d emobase.model`. This generated a new file

emobase.model which will be used by Weka later to classify the ARFF files generated by openSMILE. emobase.arff can be now deleted.

Subsection 3.3.3: Classifying with Weka

Before we could begin classifying, a few changes needed to be made to our configuration file. With these changes, the name and timestamps of the audio file are not included, ensuring our soon-to-be generated ARFF files will be compatible with the model. We also set the variable we're looking for, "emotion" to be unknown by changing the class of that attribute to a "?". A diff of the changes made can be seen in Figure 3.3.3.1.

```

1 1 [arffsink:cArffSink]
2 2 reader.dmLevel=func
3 3 ; do not print "frameNumber" attribute to ARFF file
4 4 number=0
5 + timestamp=0
5 6 ; name of output file as commandline option
6 7 filename=\cm[arffout(0){output.arff}:name of WEKA Arff output file]
7 8 ; name of @relation in the ARFF file
8 9 relation=\cm[corpus{SMILEfeatures}:corpus name, arff relation]
9 10
10 - ; name of the current instance (usually file name of input wave file)
11 - instanceName=\cm[instname(N){noname}:name of arff instance]
12 - ;; use this line instead of the above to always set the instance name to the
13 - ;; name of the input wave file
14 - ;instanceName=\cm[inputfile]
15 -
16 11 ; name of class label
17 12 class[0].name = emotion
18 13 ; list of nominal classes OR "numeric"
19 14 class[0].type = \cm[classes{unknown}:all classes for arff file attribute]
20 15 ; the class label or value for the current instance
21 - target[0].all = \cm[classlabel(a){unknown}:instance class label]
16 + target[0].all = \cm[classlabel(a){?}:instance class label]
22 17 ; append to an existing file, so multiple calls of SMILEExtract on different
23 18 ; input files append to the same output ARFF file
24 19 append=1

```

Figure 3.3.3.1: Diff of changes made to config file.

We can now begin classifying emotions. First, we used openSMILE to generate a Weka ARFF file. We set the configuration file to our edited `emobase.conf` and use a random audio file from the corpus, in this case `disgust/03b10Ec.wav`, and set the possible emotions it could be. This created a new file `output.arff`. Figure 3.3.3.2 shows the command in full and its output.

```
jacob@caroline:~/test$ ./SMILExtract -C emobase.conf -I corpus/disgust/03b10Ec.wav -classes \{anger
,boredom,disgust,fear,happiness,neutral,sadness\}
(MSG) [2] in SMILExtract : openSMILE starting!
(MSG) [2] in SMILExtract : config file is: emobase.conf
(MSG) [2] in cComponentManager : successfully registered 68 component types.
(MSG) [2] in configManager : reading config file 'emobase.conf'
(MSG) [2] in cComponentManager : successfully finished createInstances
(23 component instances were finalised, 1 data memories were final
ised)
(MSG) [2] in cComponentManager : starting single thread processing loop
(MSG) [2] in cComponentManager : Processing finished! System ran for 328 ticks.
jacob@caroline:~/test$
```

Figure 3.3.3.2: Generating the `output.arff` file to be used by Weka.

Finally, we used Weka to classify the data and predict which emotion it is (Figure 3.3.3.3). As shown, Weka said that this was “disgust” with 66.7% certainty.

```
(MSG) [2] in cComponentManager : Processing finished! System ran for 328 ticks.
jacob@caroline:~/test$ java -classpath weka.jar weka.classifiers.trees.J48 -l emobase.model -T outp
ut.arff -p 0
(MSG) [2] in SMILExtract : openSMILE starting!
(MSG) [2] in SMILExtract : config file is: emobase.conf
(MSG) [2] in cComponentManager : successfully registered 68 component types.
(MSG) [2] in configManager : reading config file 'emobase.conf'
=== Predictions on test data ===
inst# actual predicted error prediction
1 1: 3:disgust 0.667
```

Figure 3.3.3.3: Classifying the emotion using Weka.

After deleting the output file, we can choose another file (`happiness/12b02Fb.wav`) and do it again, with it predicting “happiness” at 100% certainty (Figure 3.3.3.4).

```
1 As you see, the program has predicted that this is a disgust with 66.7% certainty.
1: ? 3: disgust 0.667

jacob@caroline:~/test$ rm output.arff
jacob@caroline:~/test$ ./SMILExtract -C emobase.conf -I corpus/happiness/12b02Fb.wav -classes \{anger,boredom,disgust,fear,happiness,neutral,sadness\}
(MSG) [2] in SMILExtract : openSMILE starting!
(MSG) [2] in SMILExtract : config file is: emobase.conf
(MSG) [2] in cComponentManager : successfully registered 68 component types.
(MSG) [2] in configManager : reading config file 'emobase.conf'
(MSG) [2] in cComponentManager : successfully finished createInstances
(23 component instances were finalised, 1 data memories were finalised)
(MSG) [2] in cComponentManager : starting single thread processing loop
(MSG) [2] in cComponentManager : Processing finished! System ran for 338 ticks.
jacob@caroline:~/test$ java -classpath weka.jar weka.classifiers.trees.J48 -l emobase.model -T output.arff -p 0

=== Predictions on test data ===
=== Predictions on test data === prediction
1: ? 3: disgust 0.667

inst# actual predicted error prediction
1 1: ? 5: happiness 1 0.333
```

Figure 3.3.3.4: Generating another output.arff and classifying.

Section 3.4: Merging the databases

Because we receive very little data in terms of what each database detected (with nothing but a single word from the facial detection program), a new table was created in our MySQL database to store what little information we can acquire: the names of the emotions from the two databases, their given (or assumed) percentages, and the emotion it should be via a training run. To this end, a Python script was created that calls each program, does a comparison to the results stored in our table, and prints out a single emotion.

Two new directories were created in our new merge/ directory: ears/ and eyes/. The facial detection binary was copied to the eyes/ directory and our necessary files from openSMILE were copied to the ears/ directory. The Python script can now be built.

We began by getting the output from the facial detection script. Since its output is a single word written to the console, we just read it in. And because it does not give a percentage, we hard-coded the percentage as found in Section 3.2.4. Working with openSMILE was similar except we needed to make two external calls, one to openSMILE and one to Weka. The output from Weka is more complicated than the first, but it is structured enough that we can reliably obtain both the emotion predicted and its given percentage. To help with development, we included a few debug statements. Input to the script is given by command-line arguments. Figure 3.4.1 shows the script as it is with Figure 3.4.2 showing the output with the debug statements enabled.

```

1  #!/usr/bin/python
2
3  import sys, subprocess, os
4
5  # -----
6  debug = True
7  # -----
8
9  if debug: print "DEBUG:\tRunning it by the eyes"
10 with open(os.devnull, "w") as fnull: resultFromEyes = subprocess.check_output(["mono", "./eyes/sobel
    .exe", sys.argv[2]], stderr=fnull)
11 emotionFromEyes = resultFromEyes.split("\n")[len(resultFromEyes.split("\n"))-2]
12 emotionFromEyesPercent = 0.68
13 if debug: print "DEBUG:\t\tEmotion detected: " + emotionFromEyes + ". Percent: " + str(
    emotionFromEyesPercent)
14 emotionFromEyesPercent = 0.68
15
16 if debug: print "DEBUG:\tRunning it through the ears"
17 try:
18     os.remove("output.arff")
19 except OSError:
20     pass
21 with open(os.devnull, "w") as fnull: subprocess.check_output(["./ears/SMILExtract", "-C", "./ears/
    emobase.conf", "-I", sys.argv[1], "-classes", "{anger,boredom,disgust,fear,happiness,neutral,
    sadness}"], stderr=fnull)
22 if debug: print "DEBUG:\t\tarff file generated. Identifying emotion"
23 with open(os.devnull, "w") as fnull: resultFromEars = subprocess.check_output(["java", "-classpath",
    "./ears/weka.jar", "weka.classifiers.trees.J48", "-l", "./ears/emobase.model", "-T", "output.
    arff", "-p", "0"], stderr=fnull)
24 emotionFromEars = resultFromEars.split("\n")[5].split()[2].split(":")[1]
25 if emotionFromEars == "happines": emotionFromEars = "happiness"
26 emotionFromEarsPercent = float(resultFromEars.split("\n")[5].split()[3])
27 if debug: print "DEBUG:\t\tEmotion predicted: " + emotionFromEars + ". Percent: " + str(
    emotionFromEarsPercent)]

```

Figure 3.4.1: Python script that runs both the Human Emotion Detection and openSMILE programs at once and outputs the results.

```

jacob@caroline:~/emotion/merge$ ./merge.py ../corpus/anger/10a01Wa.wav ../EmotionDetection/Pic/ibrahim\ sir/dg\ 21951.jpg
DEBUG: Running it by the eyes
DEBUG: Emotion detected: surprise. Percent: 0.68
DEBUG: Running it through the ears
DEBUG: arff file generated. Identifying emotion
DEBUG: Emotion predicted: anger. Percent: 0.974
jacob@caroline:~/emotion/merge$ █

```

Figure 3.4.2: Output of Python script.

We then created the training table in a MySQL database to hold the emotions and percentages generated by these two emotions plus the expected emotion. We have five columns: “id” (an auto-incremented integer), the emotion from openSMILE/Weka “eEars” (text), the percentage from openSMILE/Weka “eEarsP” (float 3,2), the emotion from the facial detection program “eEyes” (text), the percentage from the facial detection program “eEyesP” (float 3,2), and the expected emotion “emotion” (text). For simplicity’s sake we added the table to the same database used by the facial detection program. The table information was then added to the script.

To detect the emotion, we compare the output and percentages from the two programs to stored rows in the database. If the emotions match and the combined percentages are within an accepted range, we say that row matches and output the stored emotion. If we are unable to find any rows that match or match within the accepted range, we’re forced to output from the program that had the higher percentage of certainty. If we’re in training, we add the emotions and percentages to the database. Figure 3.4.3 shows this logic in the program with Figure 3.4.4 showing the output both of a row matching and of one that does not.

```

db = MySQLdb.connect(host=server, user=user, passwd=pword, db=database)
cur = db.cursor()

if debug: print

if len(sys.argv) == 3:
    print "DEBUG:\tSearching through database"
    cur.execute("SELECT * FROM weights WHERE `eEars` = %s AND `eEyes` = %s ", (emotionFromEars, emotionFromEyes))
    rows = cur.fetchall()

    emotion = "unknown"
    if len(rows):
        if debug: print "DEBUG:\t\tFound " + str(len(rows)) + " row(s) in database"
        for i in range(0, len(rows)):
            if abs(abs(percent(rows[i][2]) - percent(emotionFromEarsPercent)) + abs(percent(rows[i][4]) - percent(
                emotionFromEyesPercent))) < minimumWeight:
                emotion = rows[i][5]
                minimumWeight = abs(abs(percent(rows[i][2]) - percent(emotionFromEarsPercent)) + abs(percent(rows[i
                    ][4]) - percent(emotionFromEyesPercent)))
        if emotion == "unknown":
            if debug and len(rows): print "DEBUG:\t\tNo rows matched enough"
            if debug: print "DEBUG:\tComparing percentages"
            if emotionFromEyesPercent > emotionFromEarsPercent and emotionFromEyes != "ambiguous":
                emotion = emotionFromEyes
            else:
                emotion = emotionFromEars

    print emotion
else:
    if debug: print "DEBUG:\tInserting results into database"
    cur.execute(
        "INSERT INTO `weights`(`eEars`, `eEarsP`, `eEyes`, `eEyesP`, `emotion`) VALUES (%s,%s,%s,%s,%s)",
        (
            emotionFromEars,
            emotionFromEarsPercent,
            emotionFromEyes,
            emotionFromEyesPercent,
            sys.argv[3]
        )
    )
    cur.connection.commit()

```

Figure 3.4.3: Final draft of Python script.

```

jacob@caroline:~/emotion/merge$ ./merge.py ../corpus/anger/10a01Wa.wav ../EmotionDetection/Pic/ibrahim\ sir/dg\ 21951.jpg
DEBUG: Running it by the eyes
DEBUG: Emotion detected: surprise. Percent: 0.68
DEBUG: Running it through the ears
DEBUG: arff file generated. Identifying emotion
DEBUG: Emotion predicted: anger. Percent: 0.974

DEBUG: Searching through database
DEBUG: Found 1 row(s) in database
anger
jacob@caroline:~/emotion/merge$ ./merge.py ../corpus/sadness/10a05Tb.wav ../EmotionDetection/Pic/ibrahim\ sir/dg\ 21774\ \ (4\).JPG
DEBUG: Running it by the eyes
DEBUG: Emotion detected: neutral. Percent: 0.68
DEBUG: Running it through the ears
DEBUG: arff file generated. Identifying emotion
DEBUG: Emotion predicted: sadness. Percent: 0.889

DEBUG: Searching through database
DEBUG: Comparing percentages
sadness
jacob@caroline:~/emotion/merge$ █

```

Figure 3.4.4: Output of Python script with debugging on.

Finally, to train the program we add another argument with the emotion it should be (Figure 3.4.5).

```

jacob@caroline:~/emotion/merge$ ./merge.py ../corpus/sadness/10a05Tb.wav ../EmotionDetection/Pic/ibrahim\ sir/dg\ 21774\ \ (4\).JPG sadness
DEBUG: Running it by the eyes
DEBUG: Emotion detected: neutral. Percent: 0.68
DEBUG: Running it through the ears
DEBUG: arff file generated. Identifying emotion
DEBUG: Emotion predicted: sadness. Percent: 0.889

DEBUG: Inserting results into database
jacob@caroline:~/emotion/merge$ █

```

Figure 3.4.5: Output of Python script when training with debugging on.

Chapter 4 - Conclusion

"Human Emotion Detection from Image" was ported to Linux and MySQL. While accuracy dropped ~25% after porting, it is still as high as what is acceptable for humans according to Picard, et al (2001). But further research and evaluation should be done to determine what caused the drop and what can be done to remedy it.

openSMILE was configured to work with the Berlin Database of Emotional Speech, and models based on the database were successfully built using openEAR and Weka. Accuracy varies based on which audio clip is used but is usually quite high.

These two disparate systems were successfully combined into a single system that uses both the emotions detected but also the percentage of accuracy of each to determine a single emotion. This can be further enhanced by training the model.

In the end, we have a system in place that can, while using two different emotion detectors in the form of openSMILE and "Human Emotion Detection from Image", recognize a person's emotive state with a higher degree of accuracy than if alone.

References

- BullZip. (2013). Access to MySQL (Version 5.1.0.242) BullZip.
- Burkhardt, F., Paeschke, A., Rolfes, M., Sendlmeier, W. F. & Weiss, B. (2013). Berlin database of emotional speech: Additional information. Retrieved, 2014, Retrieved from <http://pascal.kgw.tu-berlin.de/emodb/docu>
- Burkhardt, F., Paeschke, A., Rolfes, M., Sendlmeier, W. F., & Weiss, B. (2005). A database of German emotional speech. Paper presented at the *Interspeech*, 1517-1520.
- Eyben, F., Weninger, F., Gross, F., & Schuller, B. (2013). Recent developments in openSMILE, the Munich open-source multimedia feature extractor. Paper presented at the *Proceedings of the 21st ACM International Conference on Multimedia*, 835-838.
- Eyben, F., Wollmer, M., & Schuller, B. (2009). OpenEAR—introducing the Munich open-source emotion and affect recognition toolkit. Paper presented at the *Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009. 3rd International Conference*, 1-6.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1), 10-18.
- OLE DB. Retrieved, 2013, Retrieved from http://mono-project.com/OLE_DB

- Picard, R. W., Vyzas, E., & Healey, J. (2001). Toward machine emotional intelligence: Analysis of affective physiological state. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(10), 1175-1191.
- Ramakrishnan, S. (2012). Recognition of emotion from speech: A review. *Speech Enhancement, Modeling and recognition—algorithms and Applications*, 121.
- Ratliff, M., & Patterson, E. (2008). Emotion recognition using facial expressions with active appearance models. Paper presented at the *Proceedings of the 3rd IASTED International Conference on Human-Computer Interaction, HCI*, 138-143.
- Reeves, B., & Nass, C. I. (1996). *The media equation: How people treat computers, television, and new media like real people and places*. Chicago, IL, US: Center for the Study of Language and Information; New York, NY, US: Cambridge University Press.
- Schuller, B., Rigoll, G., & Lang, M. (2004, May). Speech emotion recognition combining acoustic features and linguistic information in a hybrid support vector machine-belief network architecture. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on* (Vol. 1, pp. I-577). IEEE.
- shakil0304003 (2010). Human Emotion Detection from Image [Computer Software]. Bangladesh: CodeProject. Retrieved October 14, 2013. Available from <http://www.codeproject.com/Articles/110805/Human-Emotion-Detection-from-Image>