ABSTRACT

With the advancement of computing power available on most home computers, interacting with the computer by visual means is becoming possible and probable. Computer vision provides the ability for face recognition and other biometrics, emotion sensing, robotics, and many other forms of autonomous interaction. All of these tasks require that the computer be able to locate and track the human face, through a visual sensor, like a camera. Through the use of Haar cascade classifiers for detection and the Lucas-Kanade algorithm, the CAMSHIFT algorithm, and active contours for feature tracking, this project created an application that can detect and track the human face, mouth, nose, and eyes when they are present in a camera video stream. By using the Open Computer Vision (OpenCV) library, this program can easily be ported to a new platform/
## TABLE OF CONTENTS

Abstract .......................................................................................................................................................... ii

Table of Contents .......................................................................................................................................... iii

List of Figures ................................................................................................................................................ vi

List of Tables ................................................................................................................................................ vii

1. Introduction and Background .......................................................................................................................... 1

   1.1 Haar Classifier Cascades .......................................................................................................................... 2

       1.1.1 Integral Image .................................................................................................................................. 2

       1.1.2 Classifiers Cascaded ....................................................................................................................... 4

       1.1.3 Problems with Haar Classifiers ..................................................................................................... 5

   1.2 Tracking Algorithms ................................................................................................................................. 6

       1.2.1 Lucas Kanade Algorithm ................................................................................................................. 6

       1.2.2 Tracking Through Contours ............................................................................................................. 7

       1.2.3 CAMSHIFT ...................................................................................................................................... 8

   1.3 Open Computer Vision Library .................................................................................................................. 9

       1.3.1 Applicable OpenCV Functionality .................................................................................................... 10

       1.3.2 Author’s Previous Work with OpenCV .......................................................................................... 11

2. Face Feature Detection and Tracking ............................................................................................................ 13

   2.1 Method used for Detection ....................................................................................................................... 13

   2.2 Methods used for Tracking ...................................................................................................................... 14

   2.3 Face Detection and Tracking Program ................................................................................................... 15

3. System Design ............................................................................................................................................... 16

   3.1 Main Program Module ............................................................................................................................. 16
# Table of Contents

3.1.1 Configuration File Processing ................................................................. 17
3.1.2 Main Module Flow ................................................................................... 18

3.2 Facial Feature Detection Module ................................................................. 22
  3.2.1 Training the Classifier ............................................................................ 22
  3.2.2 Face Detection and Regionalization ...................................................... 23
  3.2.3 Face Feature Detection .......................................................................... 25

3.3 Facial Feature Tracking Module ................................................................. 26
  3.3.1 Tracking with Lucas-Kanade ................................................................. 27
  3.3.2 Tracking with CAMSHIFT ................................................................. 27
  3.3.4 Tracking with Active Contours .............................................................. 28

4. Evaluation and Results .................................................................................... 30
  4.1 Facial Feature Detection ........................................................................... 30
     4.1.1 Classifier Accuracy ............................................................................. 30
     4.1.2 Module Detection Accuracy ............................................................... 33
     4.1.3 Detection Problems Explained ........................................................... 34
  4.2 Facial Feature Tracking ............................................................................ 36
     4.2.1 Lucas-Kanade Tracking Results ........................................................ 36
     4.2.2 CAMSHIFT Tracking Results ............................................................. 37
     4.2.3 Active Contours Tracking Results ..................................................... 38
  4.3 Program Performance ............................................................................... 39
     4.3.1 Frame Rates ..................................................................................... 40

5. Future Work .................................................................................................. 42
  5.1 Adaptation to HCI ..................................................................................... 42
LIST OF FIGURES

Figure 1.1 Common Haar Features..................................................................................... 2
Figure 1.2 Summed area of integral image................................................................. 3
Figure 1.3 Summed area of rotated integral image.................................................. 4
Figure 1.4 Canny Edge Image .................................................................................... 8
Figure 1.5 Detected Objects...................................................................................... 11
Figure 1.6 Inaccurate Detection................................................................................ 12
Figure 3.1 Threaded Program Flow .......................................................................... 19
Figure 3.2 Non-threaded Program Flow ............................................................... 20
Figure 4.1 Haar Eye Classifier Detection ............................................................... 32
Figure 4.2 Face Feature Detection with Images ...................................................... 34
Figure 4.3 Face Feature Detection with Streams..................................................... 34
Figure 4.4 Inaccurate Face Detection ...................................................................... 35
Figure 4.5 Lucas-Kanade Tracking......................................................................... 36
Figure 4.6 Points Lost Due to Profile View............................................................. 37
Figure 4.7 Inaccurate CAMSHIFT Tracking............................................................ 38
Figure 4.8 Active Contour Tracking......................................................................... 39
LIST OF TABLES

Table 3.1 Size of the Haar Classifiers Cascades............................................................... 23
Table 4.1 Detection Rate of Haar Classifiers Cascades.................................................... 31
Table 4.2 False Positive Rate of Haar Classifiers Cascades............................................. 32
Table 4.3 Detection Rate of Face Feature Detection......................................................... 33
Table 4.4 Frame Processing Rates.................................................................................... 41
1. INTRODUCTION AND BACKGROUND

With personal computers becoming more powerful and now even capable of parallel processing, real-time computer vision is becoming possible with a home computer. Computer vision is the field of research that focuses on allowing the computer to interact with the environment by visual means, such as a camera [OpenCV 2004, Wilson 2005]. One of the most popular focuses of computer vision has been object detection, tracking, and recognition. Differentiating an object from the background is one of the most difficult tasks for computer vision. Tracking that object from one image or frame to the next makes the task even more complicated.

The human face poses more problems than other objects since the human face is a dynamic object that comes in many forms and colors [Muller 2004]. However, facial detection and tracking provides many benefits. Facial recognition is not possible if the face is not isolated from the background. Human Computer Interaction (HCI) could greatly be improved by using emotion, pose, and gesture recognition, all of which require face and facial feature detection and tracking [Bradski 1998].

Although many different algorithms exist to perform face detection, each has its own weaknesses and strengths. Some use flesh tones, some use contours, and others are even more complex involving templates, neural networks, or filters. These algorithms suffer from the same problem; they are computationally expensive [Bradski 1998]. An image is only a collection of color and/or light intensity values. Analyzing these pixels for face detection is time consuming and difficult to accomplish because of the wide variations of shape and pigmentation within a human face. Pixels often require reanalysis for scaling and precision. Viola and Jones devised an algorithm, called Haar Classifiers,
to rapidly detect any object, including human faces, using AdaBoost classifier cascades that are based on Haar-like features and not pixels [Viola 2001].

1.1 Haar Classifier Cascades

The core basis for Haar classifier object detection is the Haar-like features. These features, rather than using the intensity values of a pixel, use the change in contrast values between adjacent rectangular groups of pixels. The contrast variances between the pixel groups are used to determine relative light and dark areas. Two or three adjacent groups with a relative contrast variance form a Haar-like feature. Haar-like features, as shown in Figure 1.1, are used to detect an image [OpenCV 2004]. Haar features can easily be scaled by increasing or decreasing the size of the pixel group being examined. This allows features to be used to detect objects of various sizes.

1. Edge features
   1. (a) (b) (c) (d)
2. Line features
   1. (a) (b) (c) (d) (e) (f) (g) (h)
3. Center-surround features
   1. (a) (b)

Figure 1.1 Common Haar Features

1.1.1 Integral Image

The simple rectangular features of an image are calculated using an intermediate representation of an image, called the integral image [Viola 2001]. The integral image is an array containing the sums of the pixels’ intensity values located directly to the left of a
pixel and directly above the pixel at location \((x, y)\) inclusive. So if \(A[x,y]\) is the original image and \(AI[x,y]\) is the integral image then the integral image is computed as shown in equation 1.1 and illustrated in Figure 1.2.

\[
AI[x, y] = \sum_{x' \leq x, y' \leq y} A(x', y') 
\]  

(1.1)

![Figure 1.2 Summed area of integral image](image)

The features rotated by forty-five degrees, like the line feature shown in Figure 1.1 2(e), as introduced by Lienhart and Maydt, require another intermediate representation called the rotated integral image or rotated sum auxiliary image [Lienhart 2002]. The rotated integral image is calculated by finding the sum of the pixels’ intensity values that are located at a forty-five degree angle to the left and above for the \(x\) value and below for the \(y\) value. So if \(A[x,y]\) is the original image and \(AR[x,y]\) is the rotated integral image then the integral image is computed as shown in equation 1.2 an illustrated in Figure 1.3.

\[
AR[x, y] = \sum_{x' \leq x, y' \leq x-|y-x'|} A(x', y') 
\]  

(1.2)
Figure 1.3 Summed area of rotated integral image

It only takes two passes to compute both integral image arrays, one for each array. Using the appropriate integral image and taking the difference between six to eight array elements forming two or three connected rectangles, a feature of any scale can be computed. Thus calculating a feature is extremely fast and efficient. It also means calculating features of various sizes requires the same effort as a feature of only two or three pixels. The detection of various sizes of the same object requires the same amount of effort and time as objects of similar sizes since scaling requires no additional effort [Viola 2001].

1.1.2 Classifiers Cascaded

Although calculating a feature is extremely efficient and fast, calculating all 180,000 features contained within a 24 × 24 sub-image is impractical [Viola 2001, Wilson 2005]. Fortunately, only a tiny fraction of those features are needed to determine if a sub-image potentially contains the desired object [Menezes 2004]. In order to eliminate as many sub-images as possible, only a few of the features that define an object are used when analyzing sub-images. The goal is to eliminate a substantial amount, around 50%, of the sub-images that do not contain the object. This process continues, increasing the number of features used to analyze the sub-image at each stage.
The cascading of the classifiers allows only the sub-images with the highest probability to be analyzed for all Haar-features that distinguish an object. It also allows one to vary the accuracy of a classifier. One can increase both the false alarm rate and positive hit rate by decreasing the number of stages. The inverse of this is also true. Viola and Jones were able to achieve a 95% accuracy rate for the detection of a human face using only 200 simple features [Viola 2001]. Using a 2 GHz computer, a Haar classifier cascade could detect human faces at a rate of at least five frames per second [Lienhart 2002].

1.1.3 Problems with Haar Classifiers

As effective and efficient as Haar classifier cascades are, there are a few problems. If the desired object is relatively plain or featureless, classifiers cannot successfully eliminate enough sub-images without potentially eliminating the object itself. This problem has surfaced with facial feature detection. Certain facial features do not by themselves have enough unique Haar features to create a high positive hit rate without greatly increasing the false positive rate [Cristanccce 2003]. By decreasing the number of stages a face feature classifier uses and using the detector to only analyze a section of the face, one can greatly increase the accuracy of the classifier.

Another problem involves the inability of a classifier to cope with a perspective change. As the view angle changes, so do the features that an object exhibits. This means that a well trained Haar classifier only has an effective perspective of plus or minus fifteen degrees [Menezes 2004]. If an object has the potential for being viewed at several angles, several Haar classifiers are needed to provide high accuracy. Such a
problem exists with the human face and face features. One can easily turn one’s head, changing the Haar features and hiding potential facial features like the eye.

1.2 Tracking Algorithms

Once an object is detected in a frame it needs to be tracked from one frame to the next. Tracking algorithms in general are more efficient and precise than detection since they utilize prediction to greatly reduce the search area. Unfortunately there is not a single algorithm that accurately tracks all types of objects under all circumstances [Welch 2002]. This requires the use of many different algorithms for different objects and purposes.

1.2.1 Lucas Kanade Algorithm

The Lucas-Kanade tracking algorithm uses points to track objects. The Lucas-Kanade algorithm utilizes an intensity gradient to track specified points [Lucas 1981]. Although this algorithm was originally intended to track points of an object between two camera images for stereo vision, this algorithm can be used to track a point between two frames [Tomasi 1991]. The key to tracking a particular point from frame to frame is that the point is relatively brighter or darker than its surrounding neighbors. These points are not individual pixels but small sub-windows of pixels.

The goal of the algorithm is to find the displacement of the point from one frame to the next. The theory is a pixel group or point will move at a certain velocity from frame to frame, but will not greatly change in intensity value. Thus searching for the point is a matter of minimizing the difference of the sum of squared intensity values between the two frames [Lucas 1981, Tomasi 1991]. For a large area this can be quite
time consuming, so small windows are analyzed, sometime in a pyramidal form [Bouguet 1999].

Not all areas of an image are suitable for tracking. The boundaries of an image have a probability of a tracked point entering or leaving the frame from one frame to the next. Thus, the points on the boundaries are not suitable for tracking. A suitable point for tacking must be one whose eigen value larger than those of the surrounding pixels [Bouguet 1999, Tomasi 1991]. These points are often areas of high texture. It should be noted that since points are being tracked via displacement, if a point is occluded or moves a great distance across the image from one frame to the next, it will not be properly tracked [Bourel 2000].

1.2.2 Tracking Through Contours

Another method of tracking an object is through its contours or boundaries. When the contours are dynamic, as is the case with facial features, active contour models, also known as “snakes”, can be used. A deformable shape template requires that a set of nodes be defined with the connectivity relationship between those nodes defined so as to control the shape model of the contour [Malciu 2000]. The template requires some knowledge about the object being tracked. For example parabolas can be used to describe the upper and lower lips [Barnard 2002].

The contour is tracked and is updated from frame to frame by predictive motion techniques and by reapplying the template [Gavrila 1996]. The motion prediction narrows the search area for reacquiring the object. The contour templates for the human face can be described using Canny edge detection, shown in Figure 1.4. Canny edge
detection isolates the edges within an image using gradients [Ali 2001]. From the Canny image, points of the edge can be found to form the contours.

![Original Image (Left), Canny Edge Image (Right)](image)

**Figure 1.4 Original Image (Left), Canny Edge Image (Right)**

*Image provided by [FERET 2003]*

The main problem with contours is the fact that certain facial features, like the mouth, can be obscured by other contours [Barnard 2002, Moses 1995]. The human mouth can be obscured by facial hair, giving an inaccurate contour. Contours can drift from frame to frame. This problem is more prevalent in rapid moving objects like the mouth. However, when used with the eyes and brows and other features with a smaller degree of natural movement they tend to be more reliable.

### 1.2.3 CAMSHIFT

The Continuously Adaptive Mean SHIFT (CAMSHIFT) utilizes a color histogram and a probability distribution to relocate the object in a new frame [François 2004]. The histogram is formed using the Hue Saturation Value color space [Bradski 1998]. The histogram is then stored for later use as a model or a look-up table. When a new frame is analyzed in the window where the object was previously located, a probability distribution is computed based on the stored histogram [Bradski 1998]. If the object was not located or does not completely occupy the window, it is scaled.
Unlike the previously mentioned tracking methods, no precise point detail is given about the object, just its location. CAMSHIFT can however detect changes in an object’s position, such as a head turn or roll. Since this tracking method relies on color, sudden changes in light conditions, such as white balance adjustment, will affect the accuracy of the tracking [François 2004]. If the image has low light or contains several similar objects, then a result could be that the tracked area becomes larger than the actual object [Bradski 1998].

1.3 Open Computer Vision Library

Intel developed an open source library devoted to easing the implementation of computer vision related programs called Open Computer Vision Library (OpenCV). The OpenCV library is designed to be used in conjunction with applications that pertain to the field of HCI, robotics, biometrics, image processing, and other areas where visualization is important [OpenCV 2004]. OpenCV provides many useful programs, application programmer interfaces (APIs), and functions to easily perform research in the computer vision fields without the large learning curve [Wilson 2005].

Since the C/C++ library is available for both Linux and Windows platforms, porting most programs from one platform to the next simply requires recompiling the code on the new platform. There are a few functions and APIs, such as the GUI windowing functions and the web camera API, where not all functionality is available on both Linux and Windows. In such cases, the differences are well documented in the Web version of the manual [OpenCV 2004].
1.3.1 Applicable OpenCV Functionality

One of the key tasks to any image processing is to acquire the image to process. OpenCV provides an API for interacting with various image sources, such as Web cameras, AVI files, and image files. The data structure for storing single images or video frames is the IPL image format from Intel’s Image Processing Library. This allows for a single image, like a JPEG image, to be processed in the same manner as a frame from a video stream.

OpenCV also provides a method to train Haar classifier cascades and utilize the classifier to detect objects from an image or frame. To train the Haar classifier, one needs to provide a list of at least 500 images containing the desired object and its bounding box, the upper left pixel, height and width of the rectangle. In addition, a list of at least 500 images not containing the object is needed. The gentle adaptive boost (AdaBoost) process is used to determine what Haar features are relevant for object detection. The training process is very time consuming [Viola 2001], and it can take between one week and one month to develop a suitable classifier for a facial feature, based on experience.

In addition to the Haar detection, OpenCV provides several tracking algorithms, including the ability to use snakes, also known as contours, the Lucas Kanade algorithm, and CAMSHIFT [OpenCV 2004]. These tracking methods allow faster and more accurate information to be gathered about an object. Object detection simply gives one the bounding box of an object while these tracking methods give precise point and/or locations of the object.
1.3.2 Author’s Previous Work with OpenCV

Through work as a graduate research assistant, experience had already been gathered on using OpenCV. This led to a program that performs facial feature detection. That program included three Haar classifiers, one for the eyes, one for the mouth, and one for the nose. Although these classifiers worked as shown in Figure 1.5, they were not perfect. The mouth and nose classifier did not work under many angles.

![Figure 1.5 Detected Objects: Face (white), Eyes (red), Nose (blue), and Mouth (green)](image)

The mouth classifier was trained using a square bounding box which greatly reduces accuracy and enlarges the detected region.

The previous program written using OpenCV has several limitations. No tracking algorithm(s) were implemented, allowing once detected objects to be lost if the head was turned by such an angle that the classifier stopped working. In addition, the face had to make up a majority of the image since the classifiers required that the features must not be smaller than $24 \times 24$ pixels. This required the face to be at least $72 \times 72$ pixels. The face classifier provided by OpenCV requires a minimum of $40 \times 40$ pixels to accurately
detect a face [OpenCV 2004]. Thus, a face may have been detectable but its features may not.

The face was used to regionalize facial feature detection by limiting what areas of the face each classifier could search. Since the area analyzed by the facial feature classifiers is greatly reduced with regionalization, the time needed to compute the integral image and Haar features is dramatically reduced. If a face was not detected due to an obstruction or some other factor, inaccuracy becomes extremely prevalent as shown in Figure 1.6 and confirmed by [Cristanço 2003]. Regionalization also improved accuracy by removing the other facial features which could be detected as false positives.

Figure 1.6 Inaccurate Detection: Eyes (red), Nose (blue), and Mouth (green)
Image provided by [FERET 2003]
2. FACE FEATURE DETECTION AND TRACKING

The human face has four highly distinguishable, dynamic facial features visible, the eyes, eyebrows, nose, and mouth. These features play the most significant role in many areas of research, including emotion recognition, lip reading, and some forms of biometrics [Moses 1995]. The remaining features, such as the forehead, the chin, and cheeks, are relatively static compared to the other four features. It is for these two reasons that this project focuses on the detection and tracking of the four dynamic facial features. This program considers each eye and its brow to be a facial feature.

2.1 Method used for Detection

As previously mentioned, Haar cascade classifiers are quite accurate when detecting a human face, but inaccurate when detecting individual facial features. This project entails using Haar classifiers for detection of the face at various angles as well as the facial features. The facial feature classifiers analyze a regionalized area of the face for the desired feature. This improves accuracy by decreasing false positives and increase efficiency by reducing the search area.

Although all the necessary Haar cascade classifiers already existed, they did not suffice for this project since they had the previously mentioned problems. Thus, this project included training twelve new classifiers, three for the eyes, three for the face, three for the nose, and three for the mouth. The images used to train the new classifiers come from the Facial Recognition Technology (FERET) database which contains thousands of color and grayscale images of people from various angles and poses [Wilson 2005].
The new classifiers only have a viable range of plus or minus twelve degrees. It is for this reason that there are three classifiers for the face and facial feature to account for a frontal view with zero degrees of rotation, a quarter turn view with a plus or minus 25 degree rotation from a frontal view, and a half turn view with a plus or minus 40 degree rotation from a frontal view. A 60 to 90 degree rotation, or profile view, yields very little useful feature information since a majority of the facial features are obscured, so it will not be used.

2.2 Methods used for Tracking

Since there is no one ideal tracking algorithm, this project uses a combination of the Lucas-Kanade algorithm, the CAMSHIFT algorithm, and active contours for tracking. Once a facial feature is detected, it is then tracked from each frame to the next. Each algorithm is implemented for each of the facial features, the mouth, nose, left eye, and right eye. Since the Haar classifier does not differentiate between the left and right eyes, the differentiation occurs based on the eye’s location with respect to the other facial features.

By having several algorithms available for tracking, the end-user and any programs built using this application are able to use the algorithm best suited for its needs. Each of the tracking algorithms utilizes the OpenCV implementation of the various algorithms. The required specifications for each algorithm, like the contours required for snakes, are based on each facial feature’s needs. The Lucas-Kanade algorithm provides up to twenty points for each facial feature. The contours provide contours for the upper and lower lip, nostrils, eye brows, eye outline and pupil. The
CAMSHIFT algorithm provides a rectangular outline of the feature. If a feature disappears from view, as can happen when the head is turned or partially occluded, the missing feature is redetected in the next frame.

### 2.3 Face Detection and Tracking Program

The program, *FaceDetectAndTrack.exe*, was designed and implemented to detect and track a face and the facial features within an image. This program serves as a foundation for other programs that require detection and/or tracking of facial features. The source of the image(s) used by the program is specified in the configuration file *FaceDetectAndTrack.cfg* and may come from a Web camera, an AVI file, an image file, or a list of image files. The various formats allow repeatable tests to be performed. The different facial classifiers are used to estimate the angle of rotation of the face. The location of the various classifiers can be changed using the configuration file.

On streamed input, AVI files or Web cameras, tracking occurs instead of detection if the facial feature was identified in a previous frame. The method to be used to track each facial feature is selectable via the configuration file. This allows modifications to be made to the way the program operates without recompiling the source code. Output can be saved to a configuration file by specifying the output AVI file name in the configuration file. Face detection and tracking utilize threading to allow multiple steps to run concurrently, increasing the frame rate on multiple processor or multiple core machines. The only interactive input available to the user is stopping frame capture by pressing any key or ending frame processing by pressing the “Esc” key.
3. SYSTEM DESIGN

The Facial Feature Detection and Tracking was implemented using C++ and the OpenCV library for Windows. Since the OpenCV library is available on both Linux/Unix platforms and Microsoft Windows platforms, many pieces of the program can be migrated without requiring any code changes. The program consists of three modules: the main program, the face feature detector, and the face feature tracker. The face feature detector and face feature tracker modules are actually objects that can easily be reused in other programs. For example, with facial recognition research one can use only the face feature detector to isolate the face and its features. They also do not have any platform dependent code, allowing them to be ported to another operating system by simply recompiling them.

3.1 Main Program Module

The main program module, FaceDetectAndTrack.cpp, handles all input functionality of the program. It also acts as an intermediate module for the tracking module and the detection module. The program has three inputs, only one of which is interactive. One of the inputs is the video image or stream. The interactive input is from the keyboard which stops program execution. The primary input is the required configuration file, FaceDetectAndTrack.cfg, shown in Appendix A. This file allows the user to set various parameters, such as the Haar classifier cascade locations, input source type, and tracking methods to be used. By using a configuration file, settings can easily be changed without requiring recompilation of the program or prompting for user input every time the program is executed.
3.1.1 Configuration File Processing

When the program begins execution, the first task is to load and process the configuration file. The configuration file consists of sections in no specific order, parameter/value pairs that can be in any order within those sections, and comments that can be located anywhere within the text file. Since processing the task must be fault tolerant and easily adaptable a sub-module, ConfigFileLoader.cpp and ConfigFileLoader.h was created for processing the file. This file uses Windows specific string processing functions that prevent a direct recompilation on another operating system without minor code modification.

The first task of the configuration file sub-module is to open the file for reading. If this process fails, as it would if the file does not exist, a file not found message is printed to the console and the sub-module returns false. This return value causes the main module to terminate, ending the program. Once this file is open for reading, the default value for each possible parameter is set. This allows the configuration file to not contain all possible parameter/value pairs and the main program still function without any problems. Then each line of the program is read and processed.

A function parses out the parameter/value pair, omitting any comments. If the parameter is surrounded by brackets, “[parameter]”, it is considered to be a section header. All parameters and their values following that section header are considered to belong to that section until another section header is found. Invalid section headers are ignored. Invalid parameters are also ignored. After all parameter value pairs have been read in and set, the main module can access then through global variables.
3.1.2 Main Module Flow

After the configuration file has finished loading, the Haar classifier cascades need to be loaded into the face detection module. The precise number of classifiers is variable depending on how many files the user has specified in the configuration file. There is a minimum requirement of one face feature classifier and one face classifier. There can be at most ten of each type of classifier. If the minimum required classifiers were not loaded, then the program terminates since detection cannot occur.

The main module has the ability to be threaded. Whether or not it is used is determined by a setting in the configuration file. By threading the application, there is a potential for a performance gain on multi-core processors that are now available in personal computers. It also provides the added benefit of increasing the frames per second captured by buffering the frames. However, the calls to create threads are platform dependent, making platform migration slightly more difficult.

Since threading requires shared global variables, these global variables are initialized prior to any thread creation and are used whether or not threads are enabled. The flow of the program depends of two factors, the input type and whether threading is enabled. This report will focus on the flow with threading enabled and using a stream, like an AVI file or web camera, as the input type. If the input type is a single image file, threading is never used and processing is the same as non-threading. If a list of image files is used, the flow is the same with the exception that tracking is not performed.

The next major task after loading the configuration file is initializing the input source for image capture, whether it is a camera, video file, image file, or a list of image. It is at this point that the three additional threads are created. The first thread captures the
images or frames and stores them in a buffer. The second thread processes the buffered frames, performing detection and tracking. The third thread marks up the image with the tracking and detection data and outputs the resulting frame to the screen and to the AVI file if desired. The process flow is shown in Figure 3.1. If threading is not enabled, the sequence of capture, process, and output occurs sequentially instead of concurrently as shown in Figure 3.2.

Figure 3.1 Threaded Program Flow
Once the threads have been created, capturing begins. This thread continues to execute while there are frames to capture or until a key is pressed. All captured frames are added to a circular frame buffer. Once the buffer is full, any additional captured frames are dropped until space is made available in the buffer. The buffer can store up to 512 frames. This allows for between seventeen seconds and thirty four seconds of video to be buffered depending on the frame rate. Memory to store each frame in the buffer is not allocated until it is needed. This prevents memory from being allocated if the entire buffer is not needed, like when threading is not used.

Once at least one frame has been captured, the processing thread can begin execution. This thread will continue to execute as long as frames are being captured or there are frames to be processed. Pressing the “Esc” key causes this thread to terminate even if there are unprocessed, buffered frames. The first step this thread performs is determining which facial features were tracked in the previous frame. If this is the first frame, naturally, no facial features have been tracked. Then an attempt to detect all non-tracked features is made using the detection module. This allows facial features lost from
one frame to the next to be redetected if they reappear. This can happen if, for example, a person turns his or her head obscuring one eye with the nose.

The detected features are then passed to the tracking module. An attempt to track all facial features is made. If a feature was not previously tracked and was detected, the tracking initialization occurs during this step. The results of the tracking and detection are added to a buffer so they can be retrieved by the display thread. The process repeats for the next frame in the buffer.

Once at least one frame has been processed, the markup and display thread begins execution. This thread continues to execute as long as there are frames being processed. The first task this thread performs is to initialize the output AVI, if writing an AVI file. It waits until at least forty five frames have been captured before initialization occurs to insure an accurate frame rate for the output file.

The thread then begins to mark up the images with the detection and tracking data. For detection, a rectangle is added around the objects detected. The color depends on the facial feature. White is used for the face, blue for the right eye, yellow for the left eye, green for the nose, and red for the mouth. These colors are the same for tracking and detecting, allowing one to easily distinguish which feature is begin represented. For features that used CAMSHIFT tracking, a rectangle is draw around the area of the facial feature. For features that used active contours for tracking, the contours are drawn on the image. For features that used the Lucas-Kanade tracking, the points are added to the image. After a frame has been marked up, it is displayed on the program’s GUI window. If an AVI file is being recorded, the frame is then added to the video. The process continues for every frame that has been processed.
The original thread performs event checking. Since the original thread creates the GUI window, the original thread must perform event checking. Windows requires this in order for repainting to occur. Without it, images within the window could not be changed. It is also this thread that catches any key presses that occur while the frame capture and processing occurs. After the other threads have terminated, this thread cleans up any memory allocated and computes the frame capture rate and frame process rate using elapsed clock ticks.

3.2 Facial Feature Detection Module

The face feature detection module, *FaceFeatureDetect.cpp* and *FaceFeatureDetect.h*, performs face and facial feature detection on an image. This image is passed from the main module when a frame or image is missing any facial feature locations. This module can use up to ten Haar classifier cascades for each face or facial feature to perform the detection, although only three classifiers for each feature are currently used. Each classifier location and the size of the classifier are supplied by the main module.

3.2.1 Training the Classifiers

Since the effectiveness of the detection is solely based on the accuracy of the classifiers, the steps taken to train the classifiers are as equally important as the module used for detection. Each classifier was trained based on two sets of images. One set consisted of positive images, images containing the facial feature, and the other set consisted of negative images, images not containing the facial feature. For the positive
images, a text file was created containing the filename, the upper left pixel coordinate, and the height and the width of the face or facial feature [OpenCV 2004].

The classifiers for this project were trained using facial images from the FERET database. Between 1,925 and 2,000 positive images were used for each classifier. The positive image set used over 300 people of different ethnicities, ages, and sex. This provides a large enough sample set to get the wide variance required to produce a robust classifier [Adolf 2003]. The negative image set consisted of 4473 photographs of natural sceneries and household objects, like hammers, phones, etc., with at least one mega pixel resolution.

The minimum size needed for a classifier to detect the object had to be specified at the start of the training process. The exact dimensions depended on the average height and width of the positive images and are shown in Table 3.1. The classifiers were trained to twenty stages, except for the frontal face classifier which was trained to 19 stages.

<table>
<thead>
<tr>
<th></th>
<th>Face Classifiers</th>
<th>Eye Classifiers</th>
<th>Mouth Classifiers</th>
<th>Nose Classifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontal</td>
<td>30 × 30</td>
<td>24 × 21</td>
<td>24 × 16</td>
<td>24 × 21</td>
</tr>
<tr>
<td>Quarter Turn</td>
<td>30 × 30</td>
<td>24 × 23</td>
<td>24 × 15</td>
<td>24 × 22</td>
</tr>
<tr>
<td>Half Turn</td>
<td>25 × 30</td>
<td>22 × 24</td>
<td>24 × 17</td>
<td>24 × 24</td>
</tr>
</tbody>
</table>

Table 3.1 Size of the Haar Classifier Cascades (in pixels)

3.2.2 Face Detection and Regionalization

The module is capable of detecting only certain facial features or all of the facial features within an image. Either way, facial feature detection is first performed. The first face classifier, used to detect frontal faces, attempts to detect a face. Once faces are detected, they are sorted by area from largest to smallest. If no faces are detected then
the next face classifier attempts to detect a face. This process continues until either a face is detected or all classifiers have attempted to detect a face. By using several classifiers the probability for detection improves and the approximate angle can be estimated.

Since this module was designed to be reusable, the locations of all faces are saved within the object. Although the main module only uses the first and largest face, the ability resides within the module to get the locations of the faces and any detected facial features of each of the detected faces. Each time detection is performed on another image, the stored location data is reset to zero preventing contamination of data from one frame to the next.

Facial feature detection can be performed on an individual level, or it can be performed for all features. This module has the capability to do either. As previously mentioned, before feature detection occurs, face detection is performed. In the case of individual feature detection, facial detection only happens with the first attempted feature detection for each frame. This means that if only mouth and nose detection is being performs, facial detection only occurs once, when mouth detection is attempted.

Once at least one face is detected, facial feature detection is performed on all detected faces. For each facial feature, the face area is regionalized to reduce the false positive rate of the feature detection. The region of the face that a classifier performs detection on is limited to the area of the face that the facial feature is likely to be located. The eye detection is limited to the upper half of the face. The nose detection is limited to the middle section of the face. The mouth is limited to the lower half of the face and a 1/8 below the face region. The mouth analyzes a small section outside the face because the facial classifiers usually use the lower lip as the boundary of the face region.
3.2.3 Face Feature Detection

After the face has been regionalized, facial feature detection is performed. This
detection utilizes the corresponding classifier to the classifier that detected the face.
Therefore, when the second face classifier detects the face, the second facial feature
classifiers are used to perform detection. This allows for the pose estimation to apply to
the facial features, increasing the probability of accurate detection.

Since the face has been regionalized, detection should only find one facial feature,
or two in the case of the eyes. However, Haar classifiers scale, and it could be possible
that more than one matching feature exists. This can occur when two different scales of
the classifier find the object, which results in the bounding box of the facial feature
inscribing another bounding box for the same feature. It is for that reason that all features
detected must be analyzed to find the outermost bounding box of the feature. This is done
by looking for the largest area. By using the largest detected feature, it also helps
eliminate any residual false positives. Since the desired facial feature in the regionalized
area will be larger than any portion of any other facial features within the region, the
largest feature has the highest probability of being the actual facial feature.

With the eyes, however, there can be two features that are legitimate eyes. This
requires differentiation between the left and right eyes. There are two methods used to
determine which eye is the left and which is the right. This first method involves using
another facial feature as a reference point. If the eye is to the left of the center of the
mouth or nose then it is the left eye. This method allows a determination to be made
even if there is only one eye. It is for that reason that when individual feature detection
occurs with the eye, mouth detection is also performed. The other method is to determine
the spatial relationship between the two eyes. The one on the left is the left eye and the one on the right is the right eye. The largest left and largest right eyes are the ones considered to be the accurate feature. If only one eye is found and no other feature can be used to determine whether it is a left or right eye, then the feature is saved as both the left and right eye.

3.3 Facial Feature Tracking Module

Once a facial feature has been detected in a frame, it will be tracked by the facial feature tracking module, *FaceFeatureTrack.cpp* and *FaceFeatureTrack.h*, in successive frames. The algorithm used to track each feature is selectable via the configuration file and the main module sets the tracking methods within this module. This allows different algorithms to be used for different features. The three previously mentioned algorithms, Lucas-Kanade, active contours, and CAMSHIFT, have been implemented for each facial feature.

Facial feature tracking is iterative, requiring the location of the feature in the previous frame. This means that once a facial feature is detected, the location must be passed by the main module to the tracking module so it can be tracked in the next frame. The main module also must be aware of what was tracked in the previous frame in order to perform the necessary facial feature detection for lost facial feature reacquisition. Since this module was built for reuse, multiple tracking methods can be used to track the same feature. If at least one of those methods succeeds in tracking a facial feature, then a flag is returned to the calling module.
3.3.1 Tracking with Lucas-Kanade

In the case that tracking has failed or never been performed, the tracking method must be initialized with the location of facial feature via the `InitLucasKanade`, `InitCAMSHIFT`, or `InitActiveContour` functions. In the case of Lucas-Kanade algorithm, the points to track must be found. This operation is performed if there is a bounding box for the facial feature and there are no tracking points. OpenCV provides the function to identify the good points to track [OpenCV 2004]. This function identifies up to the specified number of points that should be able to be tracked from frame to frame. To reduce the number of points that could potentially wander, at most twenty points are tracked for each feature.

Tracking of these points is performed using the Lucas-Kanade algorithm implemented using pyramids as describe in [Bouguet 1999]. The pyramids for an image are only calculated once per frame, increasing the efficiency when more than one facial feature is tracked using the Lucas-Kanade algorithm. The module then locates the tracking points in the frame. If less than half of the tracking points are found then tracking has failed. To force initialization and detection again, the number of points being tracked is set to zero. This also allows the `FeaturesTracked` function to know Lucas-Kanade did not track the frame.

3.3.2 Tracking with CAMSHIFT

Features that are tracked using CAMSHIFT must have a color histogram of the feature being tracked. When this algorithm is performed for the first time with a facial feature, or after a feature has been lost, the algorithm is initialized by finding the histogram. The histogram is created from the hue, saturation, and value color space
instead of the red, blue, and green color space. OpenCV can easily convert from one image format to another. The histogram is then scaled to keep the values between 0 and 255.

Once a histogram has been created, a back projection of the HSV image is created. The back projection is masked with minimum saturation and values to eliminate noise, causing the facial features to stand out more. The CAMSHIFT algorithm, already implemented in OpenCV, is then performed. The result is a rotated square bounding box. This bounding box is then converted into a bounding rectangle similar to the one resulting from detection. The bounding rectangle is analyzed to ensure that the feature was tracked. If the rectangle has grown to encompass most of the image, or the rectangle is only a few pixels in size, then tracking has obviously failed. In that case, the histogram is released. Since a histogram no longer exists for that feature, the FeaturesTracked function is aware that tracking failed. It also forces detection and initialization to reoccur.

3.3.3 Tracking with Active Contours

Features that are tracked using active contours need to have the contour points found before the active contours, also called snakes, can be tracked. Two steps are needed to find these contours. The first step is to use canny edge detection to find the edges of a facial feature. Then the resulting binary image is dilated to fill in any gaps in the canny edge image. OpenCV provides a function, cvFindContours, which finds all of the contours within the image. This function is used to find the contours within the bounding rectangle. The result is a series of points defining a contour.
Those points are then used by the \textit{cvSnakeImage} function to find the active contours from one frame to the next. Once the new position of the active contours is found, they are analyzed to ensure tracking was successful. Every point is checked against the previous point to make sure they are not the same. If half of the points merged, then the contour is considered lost. Once half of the contours are lost, tracking has failed. As a result, the number of contours is set to zero, signifying that the feature was not tracked to the \textit{FeaturesTracked} function. This also forces detection and initialization to occur in the next frame.
4. EVALUATION AND RESULTS

Since the program is divided into three separate modules, testing and evaluation tests the effectiveness of each module and the classifiers themselves. The testing process is a combination of manual and autonomous tests. The manual tests are based on whether the marked image, accurately illustrates the location of the facial features. Automated testing consists of modifying the testing programs provided by OpenCV for detection.

4.1 Facial Feature Detection

There are three aspects to the facial feature detection module, the classifier detection rate and accuracy, effective perspective angle for each classifier and the accuracy and effectiveness of the cascading of the classifiers. Testing the detection uses two different sources of images: a different set of images from the FERET Database [FERET 2003] and the AVI files recorded from an inexpensive web camera. The FERET images provide a way to test the accuracy with various ethnicities, ages, and sexes with high quality images. The AVI files allow the detection to be performed on images similar to streaming Web camera video.

4.1.1 Classifier Accuracy

The accuracy of the classifiers was determined by using the performance.exe application provided by OpenCV. This tool evaluates the estimated real world performance of the classifiers. Each classifier was evaluated against approximately sixty images for each of the three angle ranges that were not used in training. This not only provided the effectiveness of the classifier within the angle of rotation range, but also the possible performance at other angles.
The testing did not take into account regionalization of the face that the detection module utilizes. This means that the false positive rate is higher than what the face detection module experiences. The detection rate of the classifiers when tested with each set of images is shown in Table 4.1. The false positive rate of the classifiers when tested with each set of images is shown Table 4.2. They show that the classifiers are effective at not only detecting faces and facial features within their specified angles, but also effective in detecting at the lower angles too. The high false positive hit rate, computed by dividing the number of false positives by the number of images, and Figure 4.1 shows the importance of regionalizing the face with facial feature detection.

<table>
<thead>
<tr>
<th>Face Frontal Classifier</th>
<th>Frontal Images (2000 images)</th>
<th>Quarter Turn Images (1925 images)</th>
<th>Half Turn Images (2000 images)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face Quarter Turn Classifier</td>
<td>100.00%</td>
<td>97.96%</td>
<td>19.05%</td>
</tr>
<tr>
<td>Face Half Turn Classifier</td>
<td>76.79%</td>
<td>95.92%</td>
<td>96.83%</td>
</tr>
<tr>
<td>Eyes Frontal Classifier</td>
<td>95.31%</td>
<td>83.82%</td>
<td>56.00%</td>
</tr>
<tr>
<td>Eyes Quarter Turn Classifier</td>
<td>100.00%</td>
<td>97.09%</td>
<td>56.00%</td>
</tr>
<tr>
<td>Eyes Half Turn Classifier</td>
<td>95.31%</td>
<td>98.53</td>
<td>80.00%</td>
</tr>
<tr>
<td>Mouth Frontal Classifier</td>
<td>98.41%</td>
<td>91.84%</td>
<td>19.05%</td>
</tr>
<tr>
<td>Mouth Quarter Turn Classifier</td>
<td>92.06%</td>
<td>97.96%</td>
<td>34.92%</td>
</tr>
<tr>
<td>Mouth Half Turn Classifier</td>
<td>73.06%</td>
<td>100.00%</td>
<td>93.65%</td>
</tr>
<tr>
<td>Nose Frontal Classifier</td>
<td>98.41%</td>
<td>51.02%</td>
<td>1.59%</td>
</tr>
<tr>
<td>Nose Quarter Turn Classifier</td>
<td>100.00%</td>
<td>100.00%</td>
<td>58.73%</td>
</tr>
<tr>
<td>Nose Half Turn Classifier</td>
<td>80.95%</td>
<td>95.92%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 4.1 Detection Rate of Haar Classifier Cascades
<table>
<thead>
<tr>
<th></th>
<th>Frontal Images</th>
<th>Quarter Turn Images</th>
<th>Half Turn Images</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Face Frontal Classifier</strong></td>
<td>0.00%</td>
<td>0.00%</td>
<td>1.59%</td>
</tr>
<tr>
<td><strong>Face Quarter Turn Classifier</strong></td>
<td>6.35%</td>
<td>4.08%</td>
<td>20.63%</td>
</tr>
<tr>
<td><strong>Face Half Turn Classifier</strong></td>
<td>23.81%</td>
<td>12.24%</td>
<td>9.52%</td>
</tr>
<tr>
<td><strong>Eyes Frontal Classifier</strong></td>
<td>90.63%</td>
<td>54.41%</td>
<td>21.33%</td>
</tr>
<tr>
<td><strong>Eyes Quarter Turn Classifier</strong></td>
<td>75.00%</td>
<td>67.65%</td>
<td>57.33%</td>
</tr>
<tr>
<td><strong>Eyes Half Turn Classifier</strong></td>
<td>142.65%</td>
<td>173.53%</td>
<td>264.00%</td>
</tr>
<tr>
<td><strong>Mouth Frontal Classifier</strong></td>
<td>166.67%</td>
<td>89.80%</td>
<td>46.03%</td>
</tr>
<tr>
<td><strong>Mouth Quarter Turn Classifier</strong></td>
<td>192.06%</td>
<td>132.65</td>
<td>82.54%</td>
</tr>
<tr>
<td><strong>Mouth Half Turn Classifier</strong></td>
<td>333.33%</td>
<td>326.53%</td>
<td>328.57%</td>
</tr>
<tr>
<td><strong>Nose Frontal Classifier</strong></td>
<td>9.52%</td>
<td>4.08%</td>
<td>0.00%</td>
</tr>
<tr>
<td><strong>Nose Quarter Turn Classifier</strong></td>
<td>44.44%</td>
<td>38.78%</td>
<td>22.22%</td>
</tr>
<tr>
<td><strong>Nose Half Turn Classifier</strong></td>
<td>236.51%</td>
<td>173.47%</td>
<td>274.60%</td>
</tr>
</tbody>
</table>

**Table 4.2 False Positive Rate of Haar Classifier Cascades**

![Image provided by FERET 2003](Image provided by [FERET 2003])

**Figure 4.1 Haar Eye Classifier Detection:** Frontal Classifier (Left), Quarter Turn Classifier (Middle), Half Turn Classifier (Right)
4.1.2 Module Detection Accuracy

Since testing the classifiers does not take into account face regionalization and using multiple classifiers for face detection, the results only provided a partial glimpse at the effectiveness. In order to test the true accuracy of the facial feature classifiers, they must be tested in conjunction with face detection and regionalization. Since the face feature detection module performs this action, the module is used to test the classifiers.

By using the detection module to perform detection against a series of images from the FERET database, not only are the classifiers tested, but the other functionality is also tested. This functionality includes isolating encapsulated bounding boxes for a feature and left and right eye differentiation. A list of three hundred random images, one hundred of which come from each angle range, is used to test the modules. The results are shown in Table 4.3.

<table>
<thead>
<tr>
<th></th>
<th>Detection Rate</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>95.67%</td>
<td>2.67%</td>
</tr>
<tr>
<td>Left Eye</td>
<td>90.67%</td>
<td>1.00%</td>
</tr>
<tr>
<td>Right Eye</td>
<td>91.00%</td>
<td>1.33%</td>
</tr>
<tr>
<td>Mouth</td>
<td>84.67%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Nose</td>
<td>87.00%</td>
<td>0.33%</td>
</tr>
</tbody>
</table>

Table 4.3 Detection Rate of Face Feature Detection

The module detection rate is fairly good when it is used with high resolution images as shown in Figure 4.2. However, the main purpose of the program is to perform detection on streaming images. The detection module was tested using five AVI files recorded from a Web camera that costs less than fifty dollars. They have a resolution of 320 pixels by 240 pixels. The detection rate is not as good as with the high quality
images, but it still worked well as shown in Figure 4.3, especially considering the quality of the images.

4.1.3 Detection Problems Explained

A majority of the inaccurate detection with the FERET images involves face detection that results in the face region not containing all facial features. Since the region does not contain all of the features, similar to Figure 4.4, the regionalization will be skewed. This causes, not only the feature missing from the face bounding box to not be detected, but the features close to it to not be detected as well. In other word, if the mouth is left off the face, both the mouth and the nose will not be detected.
With streaming images, this is less of a problem since the face may be accurately detected in the next frame or two. With the testing done via streaming images, other problems surfaced. The mouth and nose classifiers tested very well in the individual classifier test. Streaming images provide a wider range of head movement. When the head is tilted up or down more than ten to fifteen degrees the nose and mouth classifiers begin to fail more often. The best was to solve this problem is to train more classifiers. The program is capable of handling up to ten classifiers per feature, so no additional changes to the program would be required.

The other problem deals with the source of the stream image. The recorded AVI is compressed. The images resulting from this compression are not as high quality as the original captured frames. The tradeoff was made to have repeatable tests in lieu of higher quality streams. A better, more expensive camera also would produce better streams.
4.2 Facial Feature Tracking

The same thirty second AVI files used in detection testing were used in the tracking testing. Each video was run using each tracking method. This tested the tracking method module and the redetection ability of the main module. Not all of the tracking methods were successful in tracking the facial features. With each feature, when tracking failed, redetection was successfully performed.

4.2.1 Lucas-Kanade Tracking Results

The Lucas-Kanade algorithm is the most effective at tracking the individual facial features. Since the number of points is limited to twenty, most of the points are very resistant to roaming, illustrated in Figure 4.5. This allows for accurate tracking of the features from frame to frame. Once a feature is detected it is tracked successfully unless it becomes obscured or leaves the frame. The precision of the Lucas-Kanade points themselves is not high enough to guarantee that a point from one frame to the next is on the exact same location of the feature, but it will be within a few pixels of that location.

![Figure 4.5 Lucas-Kanade Tracking](image)

Rapid movement causes the image to blur and points to wonder. There is no error correction, so when a point strays, it will continue to stray for every frame there after. While this project did not entail correcting this problem, there are several solutions to it. In addition to rapid movement, moving the head into a profile view causes all of the
points to align within the same vertical plane. Once the head turns back to a frontal view most of the points will remain in the same plane as shown in Figure 4.6. It was also observed that sudden changes in lighting can result in the points scattering throughout the image.

![Figure 4.6 Points Lost Due to Profile View](image)

### 4.2.2 CAMSHIFT Tracking Results

Tracking using the CAMSHIFT method was marginal at best. Under the best lighting conditions, the nose and mouth were able to be successfully tracked. The eyes would merge with one another, forming a rectangle bounding both eyes. In most lighting conditions, like the one with the test AVI files, all facial features would merge and the whole face would be tracked. In order to try to eliminate this problem, redetection occurs when the tracking rectangles grow too large. It takes several frames for this to occur however, resulting in tracking that is useless as shown in Figure 4.7.
This confirms that CAMSHIFT is a useful method for tracking faces however. Since, most of the frame processing time for detection is from face detection, CAMSHIFT could be used to track the face from frame to frame, precluding the need for face detection once a facial feature is lost.

4.2.3 Active Contours Tracking Results

Unlike the other tracking methods, none of the features could successfully be tracked with active contours, as shown in Figure 4.8. The contours remain fairly consistent unless the feature moves. If any moderate paced movement occurs, the contour is lost. OpenCV provides a function, cvSnakeImage, which uses four main parameters, alpha, beta, gamma, and the search window size. The alpha parameter affects the continuity of the points, or how close the points are from one another. The beta parameter affects the curvature of the points, while the gamma parameter affects the resistances to curve shrinking. The search window determines how many pixels around it within the grayscale image are analyzed to help form the contour.
Several variations of these parameters were tried and tested. None of them made a marked improvement on the tracking of the contours. In an effort to account for the contours being lost with movements, the search window was increased. This, however, led to the contours drifting whether movement occurred or not. The three other parameters were adjusted to try to get the best result. The best result only allowed an approximation of the features. The contours themselves were useless.

4.3 Program Performance

As a whole the program performed very well. The threading adds performance to multiprocessor or multicore machines. This was tested using a dual processor Pentium III machine. Recording AVI files of the marked up Web camera streams allows the results to be viewed at real time speeds even if processing occurs at a slower rate. However, due to unknown reasons, an AVI file cannot be recorded if the input source is an AVI file.
Since the detection module does not directly communicate with the tracking module, it is possible for tracking to track the same eye as the left and the right. This occurs when one eye is detected in a frame; that eye is then tracked. If the same eye is detected in another frame but a determination cannot be made whether it is left or right, that same eye will be tracked as the other eye.

4.3.1 Frame Rates

The program uses clock ticks to determine how much time elapses for capturing and for frame processing. This is the most accurate way to determine the frame rates. Since the processing frame rate is dependent on several factors, including processor speed, the classifier that successfully detected the face, and whether tracking is performed, the frame rates vary greatly. To better understand the efficiency of the algorithms, the frame rates were determined by isolating most of the other factors. For example, to determine detection speed, tracking was disabled.

The speed was performed on an Athlon 1.2 GHz computer with 512 MB of RAM. The capture rate for a USB Web camera averages 15 frames per second. When using an AVI file the capture rates range from 30 frames per second to 60 frames per second depending on the resolution. Frames with detection were by far the slowest. When detection is performed on the frame, FaceDetectAndTrack averages a processing frame rate of around four frames per second if the first classifier detects the face. If no face is detected or the third classifier detects the face then the frame rate is around two frames per second. This results in an average detection frame rate of three frames per second.

The three tracking methods each have different tracking rates. To isolate the tracking rate, only the frames with successful tracking were used. The Lucas-Kanade
tracking algorithm has a frame rate of fifteen frames per second when used with a Web camera. Since this is the speed of capture, and AVI file is used to find the upper limit of tracking. A frame rate of 20 frames per second for Lucas-Kanade tracking is achieved using an AVI file. CAMSHIFT is able to track 15 frames per second. With active contours, the tracking rate is 10 frames per second. A summary of the tracking methods is shown in Table 4.4.

<table>
<thead>
<tr>
<th></th>
<th>Frame Rate</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection</td>
<td>3 fps</td>
<td>High</td>
</tr>
<tr>
<td>Lucas-Kanade</td>
<td>20 fps</td>
<td>High</td>
</tr>
<tr>
<td>CAMSHIFT</td>
<td>15 fps</td>
<td>Med</td>
</tr>
<tr>
<td>Active Contours</td>
<td>10 fps</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 4.4 Frame Processing Rates
5. FUTURE WORK

While this program works, it still has flaws and improvements that can be made. These improvements are outside of the original scope of the project. Although all of the tracking functions did not perform as expected, the CAMSHIFT and Lucas-Kanade algorithms are still very useful.

CAMSHIFT is a nonviable means of tracking facial features. It is just too temperamental. However, as with Haar classifier cascades, it is well suited for the face. The current implementation of the CAMSHIFT algorithm can be adapted to track only the face by removing the upper size limit in the \texttt{TrackFeatureUsingCAMSHIFT} function. This can be used by the Lucas-Kanade tracking to ensure the points have not strayed outside the face area. In addition, regionalization of the face can be implemented to ensure the points have not completely strayed from the facial feature.

Tracking the face using CAMSHIFT can eliminate the need for face redetection when features are lost, greatly improving the detection speed. By using the face area tracked by CAMSHIFT, the area of the image needed to be analyzed would greatly be reduced, increasing the detection speed. This would allow the detection and tracking to be closer to near real time than it currently is.

5.1 Adaptation to HCI

The main purpose of the program is to provide a foundation to face detection and tracking. The eventual goal is to utilize this foundation to analyze faces and facial features for emotion recognition. By recognizing a person’s emotions, the interaction between a person and a piece of software can more accurately be gauged.
This program can be adapted to capture the face and facial feature locations in real-time. The resulting output would be the pixel locations within a video stream and the time of the frame’s capture. This would allow one to analyze the facial feature for emotions at a later date and the synchronization of each frame and emotion with an action performed by a person using some piece of software.
6. CONCLUSION

The face feature detection and tracking program, *FaceDetectAndTrack.exe*, has been completed. It can successfully detect and track facial features a vast majority of the time. Although its accuracy is not 100%, the detection rate is around 90%. The speed of detection varied depending on which set of classifiers actually detects the face and facial features. The actual detection rate on the test computer ranged from two to four frames per second, averaging around three frames per second.

Using CAMSHIFT for facial feature tracking is marginal at its best and tracks the whole face instead, at its worst. The active contours tracking method yields little useful information. However, Lucas-Kanade is very accurate and fast so long as no obstructions to the facial feature appear. Lucas-Kanade was the fastest algorithm, tracking all of the facial features at a rate of twenty frames per second. It was not without flaws, though. A few of the tracking points strayed and obstructions caused all of the points to become inaccurate.

While the tracking methods did not work as originally believed, Lucas-Kanade was still successful. The CAMSHIFT confirmed that it was useful for tracking objects with high color variance, like the face, since the bounding rectangles grew to encompass the face. This provided a glimpse into improvements that could be made that would increase the amount of frames that could be processed per second.

With the completion of this project, further research can be conducted on various computer vision related topics. Such topics include recognition of emotions and facial recognition. By using the tracking algorithm to gather points that define the features,
Relative changes can be observed when an expression changes. This project is a stepping stone for other projects.
BIBLIOGRAPHY AND REFERENCES


APPENDIX A. SAMPLE CONFIGURATION FILE

# FaceDetectAndTrack.cfg: The configuration file for FaceDetectAndTrack.exe

# Section: [HaarCascades] - Parameters for the Haar Classifier Cascades.

# EyesCascade# - The complete or relative path for the XML haar classifier cascade for the eye detection. (replace '#' with a number ranging 0-9)

# FaceCascade# - The complete or relative path for the XML haar classifier cascade for the face detection. (replace '#' with a number ranging 0-9)

# MouthCascade# - The complete or relative path for the XML haar classifier cascade for the mouth detection. (replace '#' with a number ranging 0-9)

# noseCascade# - The complete or relative path for the XML haar classifier cascade for the nose detection. (replace '#' with a number ranging 0-9)

# EyesSize# - The Size (width X Height) that the corresponding eye haar classifier was trained to. (replace '#' with a number ranging 0-9)

# FaceSize# - The Size (width X Height) that the corresponding face haar classifier was trained to. (replace '#' with a number ranging 0-9)

# MouthSize# - The Size (width X Height) that the corresponding mouth haar classifier was trained to. (replace '#' with a number ranging 0-9)

# NoseSize# - The Size (width X Height) that the corresponding nose haar classifier was trained to. (replace '#' with a number ranging 0-9)

# Section: [Input Parameters] - parameters for the input (camera, files, etc.) to the program.

# InputSource - The input type/source (list, file, or camera)

# CameraSelection - The camera for camera input (-1 to select at runtime, 0, for the first camera, 1 for the second, etc.),

# CameraResolution - The desired resolution of the camera (does not work with all cameras)

# InputFileName - The file name including relative or full path of the list file, image file, or video file.

# ThreadingEnabled - Enables/Disables the threading (True/False).

# Section: [Output Parameters] - parameters for the output (AVI video file) from the program.

# OutputFileName - The file name including relative or full path for the video file. NOTE: Must be an AVI file (.avi).

# OutputResolution - The resolution of the AVI frames (width X Height)

# Codec - The four character compression codec for the AVI file.
If -1 is entered the choice will be made at runtime. Use 'DIB' for no compression. Available codec codes can be found at http://www.fourcc.org (not all will work). 'IV50' and 'WMV3' to work the best on Windows machines.

Section: [Tracking Methods] - The tracking method(s) to use for each facial feature.

LeftEyeTracking - The tracking method desired for tracking the left eye (None, Contours, CAMShift, LucasKanade).

MouthTracking - The tracking method desired for tracking the mouth (None, Contours, CAMShift, LucasKanade).

NoseTracking - The tracking method desired for tracking the nose (None, Contours, CAMShift, LucasKanade).

RightEyeTracking - The tracking method desired for tracking the right eye (None, Contours, CAMShift, LucasKanade).

[Haar Cascades]
EyesCascade0=./HaarCascades/EyesFrontal.xml
FaceCascade0=./HaarCascades/FaceFrontal.xml
MouthCascade0=./HaarCascades/MouthFrontal.xml
NoseCascade0=./HaarCascades/NoseFrontal.xml
EyesSize0=24X21
FaceSize0=30X30
MouthSize0=24X16
NoseSize0=24X21
EyesCascade1=./HaarCascades/EyesQtrTurn.xml
FaceCascade1=./HaarCascades/FaceQtrTurn.xml
MouthCascade1=./HaarCascades/MouthQtrTurn.xml
NoseCascade1=./HaarCascades/NoseQtrTurn.xml
EyesSize1=24X23
FaceSize1=30X30
MouthSize1=24X15
NoseSize1=24X22
EyesCascade2=./HaarCascades/EyesHlfTurn.xml
FaceCascade2=./HaarCascades/FaceHlfTurn.xml
MouthCascade2=./HaarCascades/MouthHlfTurn.xml
NoseCascade2=./HaarCascades/NoseHlfTurn.xml
EyesSize2=22X24
FaceSize2=25X30
MouthSize2=24X17
NoseSize2=24X24

[Input Parameters]
InputSource=camera
CameraSelection=-1
CameraResolution=320 X 240
InputFileName=
ThreadingEnabled=True

[Output Parameters]
OutputFileName=
OutputResolution=320 X 240
Codec=WMV3

[Tracking Methods]
LeftEyeTracking=LucasKanade
MouthTracking=LucasKanade
NoseTracking=LucasKanade
RightEyeTracking=LucasKanade