Facial Feature Tracking for Audio and Video Synthesis

Amit Chatwani
Advisors: Szymon Rusinkiewicz* and Daniel Trueman†
Supervisor: Randy Wang
April 29, 2004

1 Abstract

There has been much research in Computer Vision regarding facial feature detection and tracking over the past decade. In this project, we combine neural network, color based, and optical flow based methods to create a system that works inside the Max/MSP/Jitter framework. The set of objects developed in this project output facial feature parameters that can be used with any of the Max/MSP/Jitter audio/video synthesis capabilities. As such, the human face can be used as an instrument or tool for generating and manipulating audio and video.

2 Introduction

The human face is the basis of human expression. Our speech and subconscious and conscious facial movements convey our emotions and feelings. As such, the ability to control music using one’s face would be a very useful creative tool. This project seeks to provide the human face as an instrument for computer musicians. We go about this by attacking the problem of facial feature tracking and detection for the Max/MSP/Jitter framework.

The obvious application that follows from this is the synthesis of musical pieces or musical performance using this project. Another application would be using the system on a dancer’s face, allowing him to affect the music he is dancing to. One could also imagine setting up a scenario in which certain facial parameters affect various musical properties and then running the system on pre-recorded pieces of video and watching how different sequences create different music (i.e. George W. Bush giving a speech vs. Bill Clinton giving a speech). Due to the nature of the Max/MSP/Jitter framework, however, the application of this project extends even just audio synthesis; video could be synthesized or manipulated using these same parameters.

The main contribution of this project is bringing together various facial feature detection and tracking algorithms and putting them in a framework that many people use and have access to. This project brings together and modifies slightly the work and ideas of others to place them in an easy to use setting for audio/video synthesis and manipulation.

This paper will begin with information regarding the Max/MSP/Jitter framework. Then, we will describe some key previous work done in facial feature tracking and detection. This will be followed with the details of our approach, results, and conclusions.

3 Background Information

3.1 Max/MSP/Jitter

"Max/MSP is a graphical environment for music, audio, and multimedia.[?] Max first came about in the mid-1980’s at IRCAM developed by Miller Puckette. Around 1996, David Zicarelli added signal processing to Max in a package called MSP. Max provides the user interface and control while MSP provides signal processing functions. A good description of the goals of Max/MSP is given in the FAQ:

Instead it [Max] is aimed at people who
are not satisfied with General MIDI style wavetable synthesis as a way of realizing their music, so it emphasizes real-time control and endless customization. Another comparison to make is that due to the use of specialized and computationally unchallenging algorithms, dedicated synthesizers probably have more "voices" than a completely flexible architecture such as Max/MSP. But if these "voices" are not the sounds you want to hear, that probably doesn’t matter.

Max/MSP is not an audio editing program. It does all its processing in real time, and when you use it, you build a dynamic process—an instrument if you will—whereas an audio editing program is designed to make a static recording stored on your hard disk. You can certainly use the two together, taking hard-disk audio files and using them as source material for an Max/MSP real-time process, or interacting with an Max/MSP process and recording it, then processing the result further in an audio editing program. [?] Here is an example of a Max/MSP patch:

While MSP provides signal processing features for Max, Jitter provides the ability to generate and manipulate any matrix data. The obvious application of this is to image/video data which is inherently a matrix representation. Jitter offers:

A robust set of mathematical operators, keying/compositing, analysis, colorspace conversion and color correction, alpha channel processing, spatial warping, convolution-based filters, and special effects deliver the building blocks for your own custom video treatments. [?]

Max/MSP/Jitter also offers the ability for the developer to build his own objects which can be called from within the graphical environment. These objects are built in C using the Max/MSP/Jitter API. Examples and help are offered through the SDK and mailing lists.

Based on this description, it is easy to see that Max/MSP/Jitter offers an ideal framework on which to build a facial feature tracking for use with audio and video synthesis.

4 Previous Work

The interesting and novel part of this project is that there was previously no implementation of facial feature detection/tracking for Max/MSP/Jitter. However, the problem of facial feature detection and tracking has been studied in great depth. The use of face tracking for audio synthesis has only been studied to a small extent.

4.1 IBM Face Finding

An example of a skin tone based algorithm is one developed by IBM Research. In this algorithm, they employ a three step process: searching for areas that are within skin tone region, have "high" luminance, and finally have high horizontal texture [?].

Using this technique, candidate regions are identified and the bounding box of the estimated face region is calculated.

4.2 Ian Buck’s Work

In his undergraduate thesis, Ian Buck studied facial feature tracking for use in animated teleconferencing. He proposes both a mouth and eye tracking algorithm. His mouth tracking algorithm is of note because part of it was used in this project.
Ian describes a color and intensity mouth based tracking algorithm that is as follows: in a region that contains the mouth, the mouth pixels will be those pixels that have a high red component (as compared with blue and green) or have a low luminance. This gives a decent approximation of which pixels will be in the mouth region. Unfortunately, this algorithm assumes that the skin color of the user is fairly light and that he is clean shaven (intensity test).

4.3 Neural Networks

Neural Network based approaches to face detection while slow, are considered the most effective. These techniques involve having a set of input data regarding faces that the system is trained on. The system will then be able to recognize a face from user input.

Henry Rowley completed a large neural network based face detection system at Carnegie Mellon. His code gives the bounding box of the faces in the image and the positions of the eyes. The library code from this project was obtained and used for part of this project.

4.4 Mouthesizer

The previous work most correlated to this project is the Mouthesizer. The mouthesizer is a camera device that is positioned in front of the mouth. It uses the same computer vision algorithm as the aforementioned Ian Buck algorithm to detect mouth pixels. The size and shape of this region are then calculated.

The Mouthesizer is more relevant, however, because the size and shape values of the mouth are used to control guitar effects, a keyboard synthesizer, and sequenced loops. This ability to take parameters and plug them into different applications is exactly what this project strives for, but on a much larger scale.

5 Approach

The approach I chose to take was to couple facial feature detection with optical flow tracking. Optical flow tracking algorithms, specifically Lucas Kanade Tracking, allow for fairly robust tracking of points throughout a video stream. I used an existing implementation of LK Tracking for Jitter in cv.jit. The process is as follows: I first detect facial feature points (eyes, center of face, mouth top, mouth bottom, mouth left, mouth right) and use Lucas Kanade Tracking to track these point positions.

5.1 Face and Eyes

The first part of the facial feature detection (face bounding box and eye positions) is done using Henry Rowley’s neural network based algorithm. This algorithm is very accurate but runs too slowly to run in real time.

In my system, I run the neural network based algorithm initially, track the points using Lucas Kanade Tracking, and rerun Rowley’s algorithm regularly to make sure that the points are being tracked correctly and recalibrating them if necessary.

5.2 Mouth

The second part of the detection algorithm is for the mouth points. This is done using an algorithm similar to Ian Buck’s and the Mouthesizer. However, I only focus on finding 4 points for the mouth: top, bottom, left, and right.

To account for different lighting situations, I calculate the average red ratio

\[
arr = \frac{\text{pixel}.r}{\text{pixel}.r + \text{pixel}.g + \text{pixel}.b}
\]

and the average green ratio

\[
agr = \frac{\text{pixel}.g}{\text{pixel}.r + \text{pixel}.g + \text{pixel}.b}
\]

for the entire face (face\_arr and face\_agr). Then, I find a bounding box for where the mouth should exist and for each pixel in this box it is considered ON if pixel\_arr() <= (face\_arr + epsilon) AND pixel\_agr() <= (face\_agr + epsilon).

This formula does not try to set the pixels inside the pixel on. Instead, it only finds the lips. To find the top, bottom, left, and right points of the mouth, I traverse from the middle of each side of the matrix. While traversing, I test to see whether the current pixel is ON and if the three neighbors
in the traversal direction are also ON. If so, then that pixel is the mouth point in the given direction.

For example, to find the top lip point, begin at \(x = \text{box\_width}/2\) and \(y = 0\). Then increment \(y\) testing at each point whether \(\text{Pixel}(x, y) == 1 \&\& \text{Pixel}(x, y+1) == 1 \&\& \text{Pixel}(x-1, y+1) \&\& \text{Pixel}(x+1, y+1) ==1\). When this becomes true, the top point has been found. This routine goes similarly for the other directions.

### 5.3 Blink Detection

At the current time, no good blink detection capability has been implemented. I experimented with methods that look at the darkness of regions surrounding the found eye points and also calculating the variance of the eye regions. Currently, I am working on an approach that looks for downward optical flow in the eye region but not in the center of the face (to make sure the person is not merely moving).

### 5.4 Why This Approach?

This approach is particularly well suited for working in real time due to the fact that it relies heavily on optical flow tracking. I also have not read about an individual approach that combines both the neural network and color based methods.

This algorithm should work well in most all circumstances. Lighting will be a problem more with the mouth algorithm than the eye/face detection because the neural network based approach is not based dependent on colors. However, the lip algorithm should still work well in most lighting situations because of the normalization that occurs in the average ratio calculation.

### 6 Methodology

This project focused mainly on leveraging existing algorithms and ideas and putting them into Jitter to allow usability. In this section, I will go over the methodology taken to implement each one of the steps I took described in the above approach section.

#### 6.1 Face and Eyes

For the face detection, I began by trying several trivial algorithms. Algorithms in Jitter can be developed using the graphical interface or by developing external objects in C that link in as .dll’s. The trivial algorithms were developed using the graphical interface. An example of the simple algorithms I experimented with is detecting color regions in certain lighting conditions that appear to be skin and thresholding based on these values. The centroid of these values can then be calculated to assume the center of the face.

Implementing Henry Rowley’s face detection code for Jitter was very tedious and time consuming, but definitely worth the pain due to its robustness. The code was available as a library with an interface file. The initial challenge was to understand the Max/MSP/Jitter API and how the externals are built. The main difficulty was figuring out which standard libraries the face detection code was developed with and making sure they did not overlap with the libraries being included for the Jitter object. Once this was completed the code worked properly. The msvcrtd.lib library had to be included and the project had to be developed as a single-threaded dll.

To make the Jitter object output integers (face coordinates) instead of a matrix, I employed the custom mproc described in the 3m example of the Jitter SDK. The changes the default object setting of outputting a matrix to allow for outputs of other types.

#### 6.2 Mouth

For the mouth detection, I took a hybrid method of using Jitter’s graphical interface and building external objects. I created a patch that does the average red ratio method described in the approach section. Then, I built an external object to do the matrix traversal and output the values found.

I think this was a decent way to implement my approach. It would be cleaner and faster (although it still works in real time) to do all the work in one patch, and I plan on doing that in the future. For learning and debugging, however, it was a helpful and easier method.
7 Results

The results of the face and eye tracking showed the same behavior as the original results from Rowley’s paper. These results are 91.1% detection rate, 45 missed faces, 945 false positives while testing on a set of 130 images with 507 faces. The numbers are based on calculating 20x20 pixel windows. There were a total of 83099211 of these windows.

The speed of system is:

Pictures:

The mouth tracking algorithm is more sensitive to different lighting conditions but still acceptable. Over 20 tests in various lighting situations with two different users, the system was able to find the mouth points correctly 17 times. The main part of this project was spent in development, obviously more testing needs to be done.

In general, the results are very positive. The metric I am using is how the project completes the goal of giving the user a set of parameters based on facial features, and the system does exactly that.

7.1 Discussion

I am very pleased with the approach I took to this project, and I consider it successful so far and hope it continues to be. Members of the Max/MSP/Jitter community have expressed interest in this idea, and I hope to be able to release in some form a usable version of my work.

The inherent problem with the fact that the detection algorithm is not going on in real time but is replaced by tracking is that basically the user could just select the points he wants to track and then in that manner. But the benefit of this software is that it automates that process and allows for better and more frequent recalibration.

To extend this work, I would like to utilize the capabilities of Henry Rowley’s library to detect multiple faces in an image. I feel like the applications of that are vast when considering audiences, bands, and dancers affected the music they are involved with their facial expressions. I would also like to create a set of demos to bundle with the objects.