18-551, Spring 2003

Group 11, Final Report

Talking Head

Audio/Video Synchronizer

Charlie Butcosk

Ross Kinder

Fred Merkle

May 5, 2003

The Problem

In the world of broadcast video, audio and video infrastructure is separated. Information flows along separate channels. Eventually, usually just before transmission, the two signals are combined to flow along the same path. From a video engineering perspective, this is extremely helpful because it allows broadcast engineers to address problems in video and audio flow separately. Facilities often process their audio and video in different locations, so it simply does not make any sense to route all video feeds through audio processing locations. Similarly, it seldom makes sense for a broadcast audio feed to enter the video processing portion of a television facility.

The separation of audio and video, however, can create a delay between the two signals. If the paths are different enough, this delay can become a problem, especially in situations like news production, where it can be difficult to understand or even watch a program with a significant delay. As the synchronization of audio and video gets worse, it becomes more difficult to understand what an on-screen "talking head" is saying. Eventually, a slight degradation in understanding becomes a near-total inability to pay any attention whatsoever to what the speaker is saying. This slow decay of understanding is largely due to the fact that the brain processes audio and video simultaneously in order to comprehend what is

happening in a scene. By using visual cues that directly relate audio and video (the shape of the lips when we say "ahhh," for example), our brains can improve understanding and retention. Additionally, visual context clues are vital to our thorough comprehension of audio information.

In filmmaking, there is a similar synchronization problem to that of broadcast video. Often, during an on-location shoot, some amount of background noise will be introduced into the recorded audio. The source of this noise can vary: it could be as simple as a light breeze brushing across the microphone element, or it could be as deafening and impossible to ignore as a plane flying overhead. This noise often can affect the overall quality of the film, so these portions are typically re-recorded in a studio afterwards, in a process known as Additional Dialog Recording (ADR). This is accomplished by having the actor repeat his/her lines while viewing the video portion of the film. Because ADR is recorded after the initial shooting of the film, the process necessarily introduces a synchronization problem. Due to variations in stresses and syllable length, ADR is never perfect and is often an arduous process for the recording engineer overseeing the process. Clearly, there is a need for system to dynamically adjust the delay between video frames and recorded audio.

Through the mid-90s, Tektronix—who have since divested all their interests in audio/video synchronization—put a significant amount of research into improving the ADR process. Unfortunately, these systems largely rely on proprietary hardware that has since been abandoned. Their research—and indeed, part of the reason why Tektronix divested their motion picture ADR division—was largely conducted by specialists in the audio field rather than the video field, and so is extremely adept in its research in how the audio can

be manipulated, but somewhat lacking in its video scope..

In video conferencing, designers have solved the problematic cognitive link between audio and video by placing a priority on audio transmission. Rather than degrade the audio stream, video conferencing codecs rate-limit the video transmission. In order to avoid a desynchronization of audio and video, the system will drop frames. If end-to-end throughput seems to be decreasing, most videoconferencing the systems respond by accordingly degrading the video stream, but not the audio stream. Strictly speaking, this reflects a priority on "information" content, but perhaps more importantly reflects the development of video conferencing codecs as an extension to audio conferencing codecs. Yet if there were a way to ensure synchronization of audio and video at the receiving end, it would be possible to maintain a high frame rate while still maintaining audio intelligibility. With video conferencing technology increasingly being used to broadcast live battlefield reports, it becomes critical to make sure that compression technologies are being used to their fullest.

Currently, the industry relies on humans to correct this delay. After a system is installed, delays between processing locations are determined by sending a test audio and video pulse through the signal infrastructure and measuring the difference between the two. This solution is sometimes adequate, but is tedious, expensive, and is problematic for modern routing infrastructures. Consider the case, for example, of a studio complex with three production locations: each with a large router to route signals from a machine room, to a minor production studio, to the major production studio, and finally to the head-end. Clearly, as routing infrastructure becomes increasingly complex and computer-controlled, the chances of finding a routing combination that has not been pre-calculated increase. A dynamic solution

to the synchronization problem would clearly allow for a much more complex video routing infrastructure than currently exist.

Finally, a computer-controlled synchronization system would open the door to packet-based broadcast video facilities. Rather than being tied to one facility wiring structure, a facility could dynamically route packets across a wide array of paths, allowing for better use of facility wiring and hardware. Our system would be able to solve the delay problems that such a scheme would create.

The Solution

We attempted to solve the problem of video and audio synchronization by using face and speech recognition techniques together in order to determine the basic speech and video units that comprise each stream. Once we determined the "content" of the audio and the video—specifically, the presence and order of certain phonemes and visemes and how they match—we then determined the total delay between the string of audio units and the string of video units. We planned to implement face tracking and cropping code on the host PC, with the cropped image being sent to the C67 Evaluation Module (EVM). The EVM was then to perform the final lip tracking, viseme detection and phoneme detection to determine the delay between the video and audio signals. Since the main application for this project would be a very controlled studio environment, we only addressed the situation of so-called "talking head" video. While this data set is certainly an over-simplistic view of the kind of video a broadcast facility typically sees, there are algorithms that exist to find a face in a significant amount of scene noise. Since our project is not intended to directly address problems of face-tracking and finding in a diverse video stream, we feel our simplifications to the data set are justified.

The Prior Work

We were unable to find any previous projects involving viseme detection. It appears that we were the first group to attempt such an implementation. Group 2 in 2002 implemented a form of face recognition. Group 7 of the same year coded up road sign recognition. These projects involved forms of pattern recognition that could have sufficed for face tracking and lip tracking, but were not nearly as efficient as the algorithms we uncovered. Therefore, though other projects exist that address the problem of face tracking, we spent a significant amount of our time implementing AVQ instead.

There have been a few past projects involving phoneme detection. Past groups had implemented dynamic time warping code for speech processing. The Hidden Markov Model, while more accurate, was never implemented due to its complexity, time frame, and algorithm size with respect to the C67 EVM.

The Algorithms

Human speech is composed of 39 basic audible units that are called phonemes[1] (Fig 1).

Phoneme	Example	Phoneme	Example	Phoneme	Example
/AA/	odd	/AE/	at	/AH/	hut
/AO/	ought	/AW/	cow	/AY/	hide
/B/	be	/CH/	cheese	/D/	dee
/DH/	thee	/EH/	Ed	/ER/	hurt
/EY/	ate	/F/	fee	/G/	green
/HH/	he	/IH/	it	/IY/	eat
/JH/	gee	/K/	key	/L/	lee
/M/	me	/N/	knee	/NG/	ping
/OW/	oat	/OY/	toy	/P/	pee
/R/	read	/S/	sea	/SH/	she
/T/	tea	/TH/	theta	/UH/	hood
/UW/	two	/V/	vee	/W/	we
/Y/	yield	/Z/	zee	/ZH/	seizure

Figure 1: The 39 phonemes

These phonemes (like the sound "sh"), when strung together, form words. Phoneme research is an area that is fairly well-defined territory in audio processing. When speaking, humans create a similar basic visible unit called visemes (Fig. 2). While the total number and exact kinds of visemes remains in debate, it is generally agreed that there are fewer

[1]From the CMU Pronouncing Dictionary, http://www.speech.cs.cmu.edu/cgi-bin/cmudict

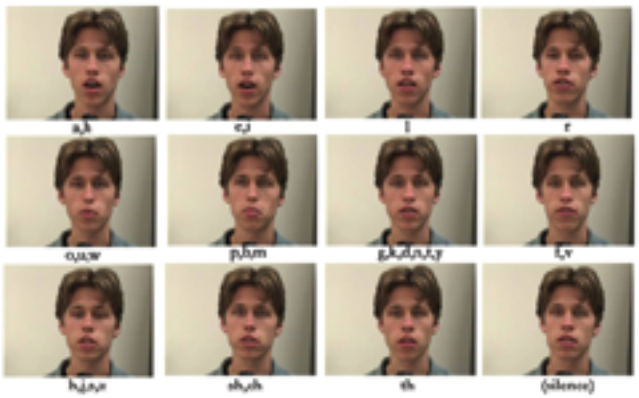

Figure 2: The 12 visemes

visemes than phonemes.

Using adaptive vector quantization, we track the face across the video frames. Once we know where the face is, we then use adaptive vector quantization again on the lower half of where the face is to find the lips. We then compare the features of the lips in the video frame to a set of stored features to determine which viseme the system is "seeing".

Our initial intention was to use dynamic time warping in order to determine the phonemes present in the audio stream, but were unable to make dynamic time warping detect single phonemes (see later section for details on the dynamic time warping algorithm and its shortcomings in relation to our application). In place of dynamic time warping, we use silence detection code to separate full words from silence. This simplification makes sense, given that the duration and length of silence varies between the kind of sentences that a speaker is saying. This means that the duration, number, and position of silences in human speech creates a relatively unique string that we could use much like we planned to use the string of

phonemes. Luckily, most speakers have a specific lip position associated with silence, meaning we can create a unique string of video and a unique stream of audio in our matching procedure.

Adaptive Vector Quantization

In order to determine which viseme was present in a frame, our synchronizer had to find the location of the face and lips in the image. This was accomplished by the implementation of a technique called Adaptive Vector Quantization[2]. We chose the algorithm due to its minimal training requirements and accuracy. We were unable to find any code for the above process, so we wrote our own implementation. The face tracking process begins with training on a single full-color 640 x 480 pel frame. (Fig. 3)

Figure 3: Input frame

[2] "Face and lip tracking using chromatic based AVQ," S. Lucey, S. Sridharan, and V. Chandran, Technical Report

This training frame is then downsampled to an 80 x 60 pel image (Fig. 6). The smaller image size allows the face tracking training process to be completed in less time, albeit with less accuracy. Additionally, the downsampling helps to make the nose and mouth less visible to the face tracking code. Each pixel (i, j) in the downsampled image is assigned a vector based on four equations (Fig. 4).

$$r = \frac{R}{R+G+B} \tag{1}$$

$$g = \frac{G}{R+G+B} \tag{2}$$

$$SD(i,j) = \sqrt{\frac{\sum_{p=i-3}^{i+3} \sum_{q=j-3}^{j+3} [\frac{R}{G}(i,j) - AI(P,q)]^2}{49}} \tag{3}$$

$$DI(i,j) = \frac{R}{G}(i,j) - AI(i,j) \tag{4}$$

where

$$AI(i,j) = \frac{\sum_{p=i-3}^{i+3} \sum_{q=j-3}^{j+3} \frac{R}{G}(p,q)}{49}$$

Figure 4: Vector dimensionality equations

These vectors are compared to a set of four code vectors hardcoded into the algorithm. Each pixel in the image is classified into one of four regions depending on minimum distance, based on the minimum squared error (Fig. 5), from the code vectors.

Then, for each region, the average of each vector (based on a pixel in the image) is taken.

$$d(x, y) = |x - y|^2$$

Figure 5: Minimum squared error (MSE)

This average becomes the new code vector. This process is repeated for 5 times, after which our code vector set has mostly converged. This set returned by the training code, becomes the set of code vectors to which all further input frames will be compared.

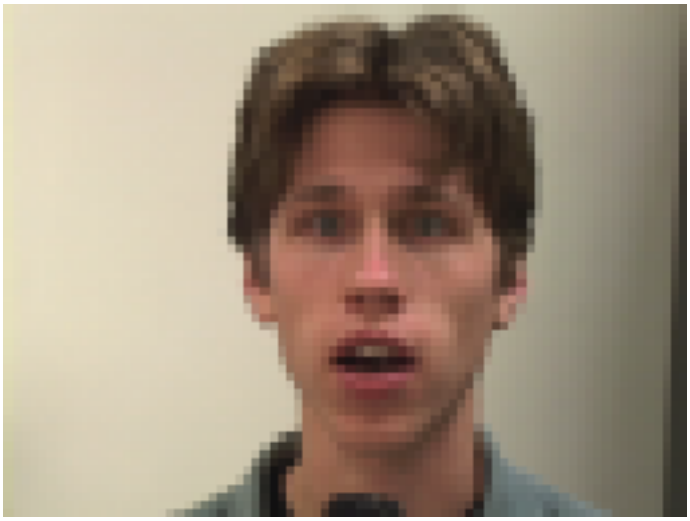

Figure 6: Downsampled version of input frame

After determining the code vectors, each frame thereafter requires limited processing. Again, we input a 640 x 480 pixel image and downsample it to 60 x 80 pixels. Each pixel in this image is also assigned a vector based on the the equations in figure 4. A new grayscale value is chosen for this pixel based upon the code vector to which it is nearest (Fig. 7).

The output of the vector quantization process makes finding the face and lips very simple. The image is scanned pel by pel until the first nearly black pel is found. This is the beginning of the face. Using facial heuristics, such as the fact that a person's lips are generally located

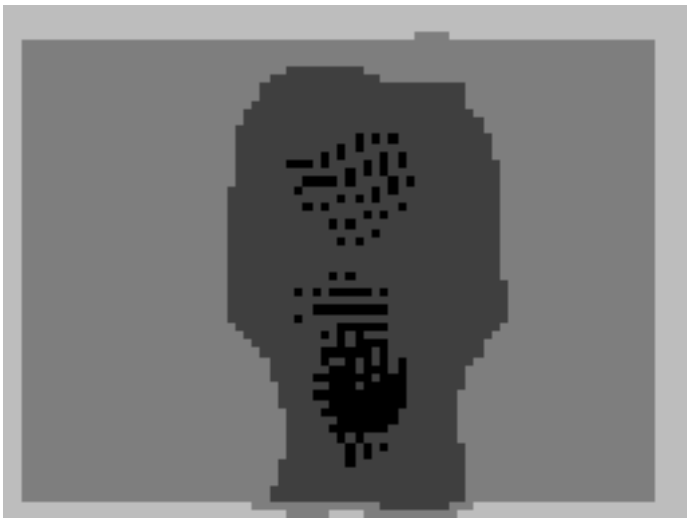

Figure 7: Input frame after downsampling and quantization

in the lower half of the facial area, the fullsize (640 x 480) image is cropped such that the lips are separated from the rest of the face (Fig. 8).

Figure 8: Input frame after lip cropping

The first image of cropped lips is used to train the lip tracking portion of the code. This code is the same AVQ code as used to track the face, however it starts with a different

set of initial code vectors. For lip tracking the initial code vectors are chosen by picking the first pel in the image and the next three pels a distance of ten pixels diagonally from the previous pel. The training process is repeated a nominal number of times. For each successive frame, the codevectors determined by this training sequence are used to quantize the images similarly to the face tracking algorithm (Fig. 9).

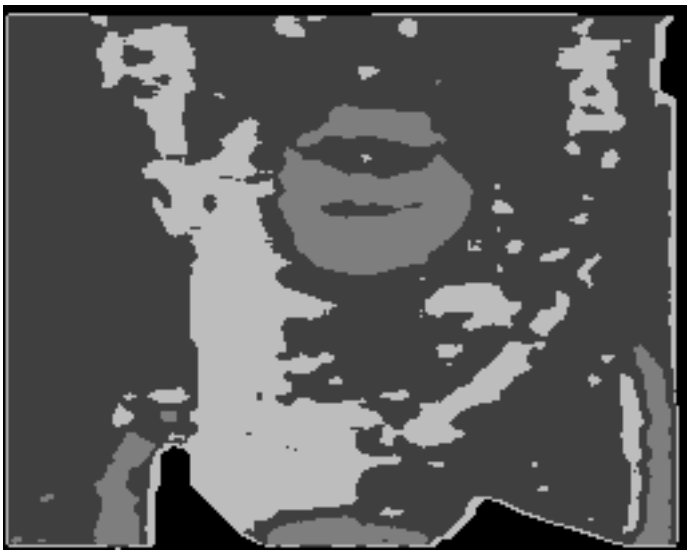

Figure 9: Cropped and quantized input frame

The lips can be easily tracked based on a vector consisting of the width, height, and area of the lips found by the lip tracking algorithm.

Dynamic Time Warping

For the audio side of our solution, we initially thought that we would use some sort of speech recognition system to generate a sequence of phonemes. This sequence of phonemes would then be compared to the sequence of visemes generated by the viseme-recognition

portion of the project.

Of the various speech recognition algorithms available, the two leading algorithms are Dynamic Time Warping (DTW) and the Hidden Markov Model (HMM). The Hidden Markov Model, which is what is used in the CMU Sphinx [3] speech system, requires a very large database and is computationally expensive. Dynamic Time Warping, on the other hand, requires a much smaller database at the expense of recognition accuracy. Initially, we chose the Dynamic Time Warping algorithm for the audio portion of our solution.

Human speech is an extremely time dependent process, in that the same utterance may have a different duration each time it is uttered. In order to determine the difference between two utterances, and thus determine which database member most closely matches the utterance, they must be aligned in time[4].

We can imagine the Dynamic Time Warping process on a grid. The horizontal axis represents the features of the given utterance, and the vertical axis represents the database utterance. The goal is to find the shortest path through the grid, which represents the distance between the two samples(Fig. 10).

In order to find the the minimum distance we impose the following constraints:

- Paths cannot go backwards in time
- Every frame in the input must be used in a matching path
- Local distance scores are combined by adding to give a global distance.

We will use Dynamic Programming to find the minimum distance. If $D(i, j)$ is the

[3]The CMU Speech group is at http://www.speech.cs.cmu.edu/
[4]http://www.dcs.shef.ac.uk/ stu/com326/

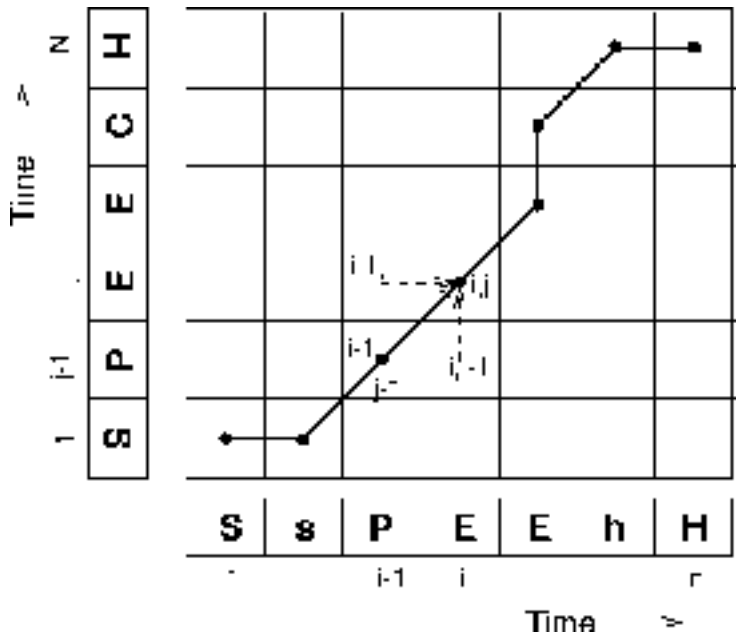

Figure 10: Time alignment path for DTW

distance up to the point (i, j) and the local distance at (i, j) is given by $d(i, j)$, then

$$D(i,j) = min\left[D(i-1,j-1), D(i-1,j), D(i,j-1)\right] + d(i,j)$$

We now have an efficient way of calculating $D(n, m)$ where the point (n, m) is the upper right hand corner of the matrix. In this way, differences in the duration of features within an utterance do not affect the recognition of the sequence.

Dynamic Time Warping represents an efficient way to recognize single words, and is widely and effectively used in that context. In order to use Dynamic Time Warping for a sequence of words, the input audio stream must be separated into segments, each of which represent an individual character. In many applications, silence detection is used to determine word boundaries. We realized that silence detection would give us just as good an

indication of the beginning and ending of words as would actually performing the dynamic time warping.

Silence Detection

The silence detection algorithm is derived from one used by many DTW silence detection systems. Whenever the input signal drops below a specified threshold, a word is considered to have ended, and whenever it rises above that threshold, a word is considered to have begun. Because the threshold is fixed, differing sound levels may cause certain word boundaries to be missed. By experimentation, we determined that the algorithm was invariant to threshold changes of plus or minus 10 dB. That is to say, that we picked -80dB as out threshold, but -90dB or -70dB would have worked equally well.

The Test and Training Files

The training data consists of a series of words that contain most of the visemes present in human speech. Our training video consists of the following words:

> One, two, three, four, five, six, seven, eight, nine, ten, eleven, twelve, thirteen, fourteen, fifteen, sixteen, seventeen, eighteen, nineteen, twenty, thirty, forty, fifty, sixty, seventy, eighty, ninety, hundred, thousand, million, billion, January, February, March, April, May, June, July, August, September, October, November, December Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, Morning, noon, afternoon, night, midnight, evening, AM, PM, now, next, last, yesterday, today, tomorrow, ago, after, before, from, for, through, until, till, that, this, day, month, week, year [5]

These words contain all of the phoneme set and all of the viseme set—no matter whose definition of viseme we use. We then used Final Cut Pro to create still TIFF images and de-interlace the images for use in traditional whole-frame algorithms. We chose a set of 12 visemes, based on Faruquie's work in the field[6]. This gave us a training set (Fig. 2).

[5] Taken from Carnegie Mellon University's Advanced Multimedia Processing group, "Audio-Visual Speech Processing," http://amp.ece.cmu.edu/projects/AudioVisualSpeechProcessing/

[6] "Audio Driven Facial Animation for Audio-Visual Reality", T. A. Faruquie, A. Kapoor*, R. Kate*, N. Rajput, L.V. Subramaniam, 2001 IEEE International Conference on Multimedia

Viseme feature data is unique to each person's face, so each test set is matched to a training set. These images of Ross, obviously, will only work in our system if we have a test set which is also of Ross. We did not create a sound training set, since the silence detection code is largely based on energy analysis which is not person-specific.

The test data consisted of Ross saying a sentence. Had we gotten more of the project working, it would have been useful to have Ross say a few different sentences under a varying set of lighting/set conditions. In order to bring the test data into our host program at a reasonable size, we used Final Cut Pro to convert the video to a non-interlaced format and a smaller resolution (see "Video Formats Used").

The Video Formats

All of our test and training data was shot on DVCAM tape, later captured to Quicktime, and eventually converted to a non-interlaced video format with a smaller resolution and lower audio sample rate. Natively, DVCAM records an interlaced, intra-frame compressed format at 720 pixels by 480 pixels and non-square pixels. While an intra-frame compressed format is not problematic, interlaced video is. For our application, non-square pixels are not a problem, provided that all of our test and training data is consistent: if we train on non-square pixels, we have to test on non-square pixels and vice-versa.

In order to simplify our project's video ingest, we used an external program to convert the captured video from 720 by 480 to 640 by 480, with square pixels. We saved this file as an AVI using the Cinepak codec. AVI, much like Quicktime, is largely used as a wrapper for a variety of video formats on Windows PCs. AVI has the added advantage, however, of being supported by Visual C++ Studio.

Interfacing with AVI's codecs is reasonably straightforward. Following the directions we found on the web[7] which had code available to access AVI Files and display them, but we modified the code in order to pass each frame to the adaptive vector quantization code.

[7] http://homespages.msn.com/RedmondAve/darrennix

Using Microsoft's API to interface with AVI files has the advantage of being moderately easy to use, especially in terms of using strange codecs. The Windows libraries manage all video codec implementation details, which allowed us to operate on a more abstracted level—notably at the frame and stream/file level. Unfortunately, in allowing us a high level of abstraction, it obscured some of the frame-level details that are pertinent. Specifically, we might be able to improve the way that we operate our vector coding by having access to the color space data, but the Windows implementation of this code was so complex that it became difficult to find how to access this information.

The Implementation Details

Unfortunately, we were unable to make the project work on the EVM. We have working implementations of the Adaptive Vector Coding and the Silence Detection, and we implemented AVI video ingest on the PC-side. Though all of these components work, our last push to get the code working on the EVM was unsuccessful. This seemed to be due to a variety of issues, including but not limited very limited time schedule. We had spent so much time fine-tuning our vector coding algorithm and our silence detection implementation that we only left about a week and half to implement our code on the EVM.

Unfortunately, the EVM did not co-operate in our rigorous time scheduling. The first issue that we encountered was a byte length/word length standard on the host PCs differing from the Linux boxes on which we wrote the original code. The EVM consistently gave us the wrong output—especially in the first 10 lines—and it seemed to be due to a difference between what Linux thought a bitmap's word size was versus the Windows API's definition of the same.

The second issue that we hit head-on was that the EVM simply was not fast enough to do all of the Adaptive Vector Coding itself. Codebook generation—easily the longest part of the initialization phase—would have taken an inordinate time on the EVM, so it

became necessary to segment our processing: initialization and some vector coding on the host PC, the rest of the vector coding and all of the sound processing on the EVM. Given the restricted time schedule—when we scheduled initially, we seem to have forgotten that all of our other classes would demand a significant amount of effort as well—rewriting our code to reflect this split became impossible to do before the demo.

We fell confident, however, that had we been able to implement our individual algorithms on the EVM, they would have worked very well in concert. The adaptive vector coding was very good at finding the test subject's head, and it worked equally well for the lips on the cropped image. The silence detection code, similarly, was able to cut off words nearly at nearly the first possible audio frame, meaning that our accuracy for knowing when our test subject was speaking was very good.

Having done so much leg work in the area of head/lip tracking, we feel that if another group attempted this project, they would have no problem completing before the deadline. In retrospect, it would have probably been a good idea to further restrict our project in order to make sure it was not such a multi-faceted project. While it seemed interesting to create a project that worked nearly as it would in a real-life situation, it would have been a better idea to create a project that would work as a single part of a larger system. We could, for instance, assume that the head had already been found or that we only get an image of the lips. Having received this image, we would have to calculate the features, etc, in order to find the viseme.