Implementation of Snake Method for Lip Tracking

Shilpi Nayak and Vanna Bushong

April 1, 2012

CS 269, UCLA

***Abstract* – This project presents a method for tracking lips with the implementation of a dynamic programming snake. The program detects faces and mouths in a video frame and initializes the snake in close proximity to the mouth. In the current version of the program, the snake will do a fairly accurate job outlining the lips in the first two or three frames. Further development is needed, however, to improve speed and accuracy as the video continues. Future work on this program could lead to a useful tool for expression recognition in applications such as political campaign videos.**

I. Introduction

Lip tracking in video has proved to be no simple task for the many researchers who have attempted it. Several methods have been implemented, including the jumping snake algorithm by Eveno *et al.* [1], pattern matching by Barnard *et al.* [2], and color extraction by Nakata and Ando [3]. Our lip tracking application focuses on the original active contours, "snakes", proposed by Kass. *et al.* [4] implemented using dynamic programming, with a combination of edge and color extraction for finding the lip boundary.

This paper outlines our approach and many of the trial and error steps we took in developing the end result. As research always goes, some of our methods were successful, while others were not, so we will explain what worked and what did not. The initial problem was to identify the faces and mouths in the video, followed by the task of initializing a snake in close proximity to the mouth. The snake implementation required a calculation of the energy of the image at each pixel, which we defined using canny edge detection and red hue extraction. We constrained the snake to be within the boundary of the box surrounding the mouth at every frame. Finally, we attempted to add a “bubble” of energy inside the lip boundary to help prevent the snake from jumping to the edges of the teeth when the mouth is open. For this to work, we had to assume that the initial frame is captured when the lips are closed and the snake is constrained such that it cannot shrink substantially more than the initial shape.

The resulting program will track the lips of one person in an input video. Our current implementation may not be as robust as we had hoped, but it does a fairly good job of outlining the lips in the initial frames. With further adjustments, this program could be a very legitimate lip tracking technique that may be used for various applications.

II. Face and Mouth Detection

The first step in tracking lips is to know if a person's face and mouth can actually be seen in a video frame. Our program uses a function from the OpenCV library for detecting faces – cvHaarDetectObjects – which is an implementation of the Viola-Jones algorithm for face detection [5].

The Viola-Jones algorithm as implemented in OpenCV essentially works by training a classifier with hundreds of sample images of the target object, in this case, a face. These are known as positive examples. Then the classifier is trained with negative examples - images of the same size without a face. Once the classifier is trained, it can be applied to a section of an image the same size as the training images and determine if there is likely to be a face in this area. The classifier can be moved across the image and resized to find faces of various sizes.

By applying this function in our program, we were able to draw a rectangle around every face detected in a video frame. For each face detected, we also drew a smaller box around the mouth, which we refer to as the "mouth box." The coordinates of the mouth box were based on the coordinates returned from the box around the face. The best calculation we arrived at for the mouth box was a width of 40% of the face and 15% of the height.

We began by drawing these boxes on the frame to ensure correctness. Once we had a mouth box of an appropriate size, we could use it as a starting point for initializing a snake.

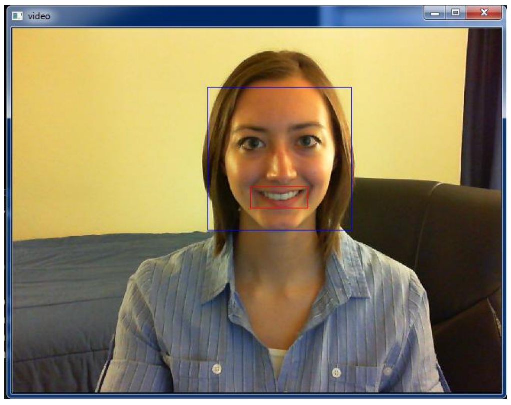
Fig. 1. Face and mouth detection.

III. Implementation of Dynamic Programming Snake for Tracking Lip Boundary

Once a face and mouth are detected, it is possible to initialize a snake fairly close to the lip boundary. A snake, also known as an active contour, is a deformable spline that tries to minimize its energy as it is pulled by image forces towards object contours. The original snake model was presented by Kass. *et al.* [4] in 1987 and continues to be implemented in various forms in computer vision projects to this day.

The energy functional used by the snake to find contours can be written as

$$E^*_{snake} = \int_0^1 E_{snake}\big(v(s)\big)ds$$

$$= \int_0^1 E_{int}\big(v(s)\big) + E_{image}\big(v(s)\big)$$

$$+ E_{con}\big(v(s)\big)ds \qquad (1)$$

where E_{int} is the internal energy of the snake, E_{image} represents the image forces, and E_{con} is the force of the external constraints.

The equation is then minimized using iterative gradient descent and solved using finite differences. For the purposes of this paper, we will not go into the details of how it is solved, but will focus on the image forces and how we calculated them for lip tracking purposes.

For our project, we decided to use a dynamic programming implementation of the original snake. Special thanks to Garett Ridge and his team of fellow students at UCLA for providing a version of the dynamic programming snake that was incredibly useful in building our own [6].
Their dynamic programming snake searches for *globally* optimal solutions as it steps across each snake pixel (snaxel). It works by finding the best choice of neighboring locations for each new snaxel to move to so that the energy of the entire snake is optimized. After deriving the best decision for the entire snake, all snaxels move simultaneously until the snake attains the minimum energy. A table of prior energies and location is maintained to store the minimum energy and optimal position.

To perform lip tracking, we made several modifications to this version of the snake. First of all, since our application deals with video, not still images, we wanted the snake to run automatically, without user interaction. So we eliminated the mouse interaction and instead initialized the snake via the mouth box boundary as previously mentioned.

A second modification was necessary for the deblurring process that Ridge used. Since we are dealing with video, there is not enough time to blur and deblur every frame while keeping the video continuous, as was possible for a still image. Also, from where we initialize the snake, the next minimum lies in the lip boundary so we need not bother about any other low energy feature. So we completely removed the blurring code and relied on our own image energy calculation for iterating the snake.

The next adjustment was made to the snake itself. Even after initializing the snake to the mouth box, we had trouble keeping it closed in a circle. Our solution was to force the snake to stay closed by drawing an additional line between the last snaxel in the array and the first.

Finally, we rewrote the external energy function so that any pixel outside of the mouth box had a maximum energy value. This prevented the snake from jumping up to the nose or down to the chin where the gradients were often stronger. For the pixels within the box, we defined an energy cost based on calculations of the image gradient, canny edges, and pseudo hue extraction [1]. The following section describes how we obtained these values.

IV. Calculation of Pixel Energies

The essential component of making a snake move is the energy of the pixels around it. For lip tracking, we needed to assign lower energy values to the pixels on the outer boundary of the lips. To do this, we performed three different types of calculations on the image and eventually used two of these values in our external energy function.

We first converted our original image frame from RGB color to a grayscale image and calculated the gradient of that image. We used the sobel edge detector to find the gradient in both directions and the strength of the edge.

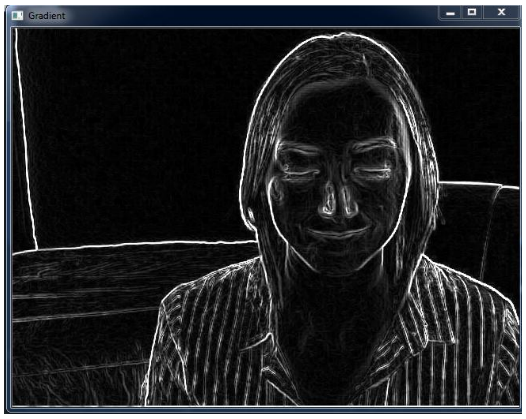

Fig. 2. A sample frame with a gradient applied.

It was immediately clear that gradient values alone would not be enough to keep the snake aligned to the mouth throughout the video. To make the boundary even stronger, we calculated another frame using canny edge detection. OpenCV has a convenient function for finding canny edges, which we used after first blurring the grayscale version of the original frame. The cvCanny function works by first calculating gradient of the image followed by thresholding and suppressing the non-maximal region of the image.

We realized that since the canny function was already calculating a gradient, it would not be necessary to include a separate value for the gradient we had found on the grayscale image before.

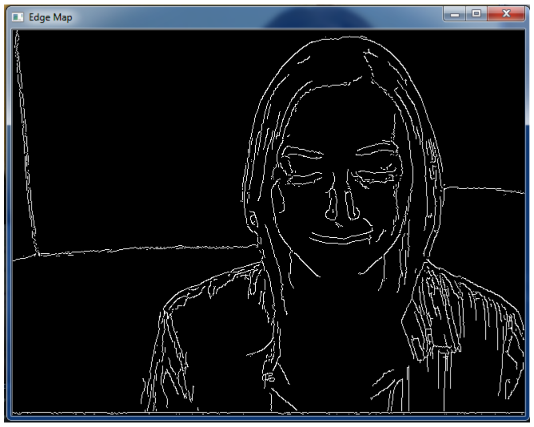

Fig. 3. A frame with canny edges detected.

The gradient and canny edge images were great for finding the edges of facial features, but neither took into account one of the most distinguishing aspects of lips: color. Naturally, we wanted to assign lower costs to pixels with a higher red component. This was achieved by first extracting the red, green, and blue component of each pixel. Then a brightness value Y was calculated using the following:

$$Y = (0.299 * B) + (0.587 * G) + (0.114 * R) \tag{2}$$

This is in accordance to ITU-R BT.601 standard for brightness in MPEG and JPEG algorithms.

The color components were then divided by Y and multiplied by 100 for scaling purposes.

$$\begin{aligned} C_r &= R/Y * 100 \\ C_g &= G/Y * 100 \\ C_b &= B/Y * 100 \end{aligned} \tag{3}$$

Based on the research by Nakata and Ando [3], the pixel most likely to match lips will have a red component above 125, a green component between 80 and 95, and a brightness value below 180. For all pixels that fell within these constraints, we set their hue value using the following:

$$H_p = \{[R/(.5 * R + .5 * G)] * 255 - Y\} * 200 \tag{4}$$

Any pixels outside of the constraints were simply assigned a black value. This gave us an extracted hue image like the following.

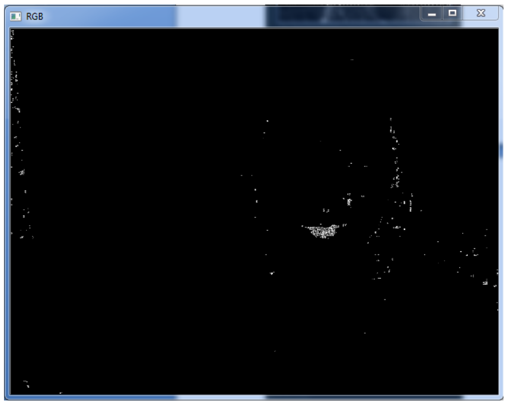

Fig. 4. A frame with red hues extracted.

The final step was to calculate the gradient of this new image using the same method as for the original grayscale image. The resulting pixels were used as another input to the external energy function.

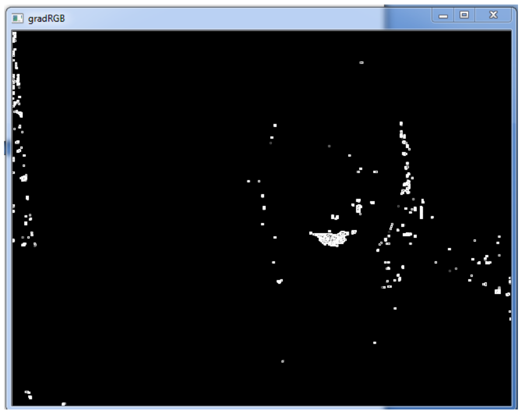

Fig. 5. The hue-extracted frame with a gradient applied.

In the process of developing this program, we attempted many different weights and combinations of weights for the pixel values used to calculate external energy. In the end, the combination that worked best is given here:

$$E_{image} = (w_{line} * P_{hue}) - \left(w_{canny} * P_{canny}^2\right) - (w\_hue * P_{gradient\_hue}^2) \quad (5)$$

where

$$w\_line = 250$$
$$w\_canny = 100$$
$$w\_hue = 500$$

and P_{hue} is the current pixel of the hue-extracted frame, P_{canny} is the current pixel of the frame with canny edges detected, and $P_{gradient\_hue}$ is the current pixel of the hue-extracted frame with a gradient applied.

This energy value is calculated for each snaxel and used as an input to the energy function called as the snake iterates.

V. Constraints

Early in development, it became apparent that we would need to place additional constraints on the snake to solve a couple of initial problems. First, the snake needed to stay within the mouth box every frame. Second, we needed to prevent it from collapsing on itself when the mouth is open and the teeth create a stronger edge than the lips.

To solve the first problem, we applied a shift to the entire snake at the beginning of each frame. By storing the position of the mouth box at the previous frame and calculating its change in x and y coordinates at the next frame, we could apply the same adjustment to every snaxel. This does not change the shape of the snake, but simply moves it to be within the mouth box. This method was very successful in keeping the snake close to the mouth, even when the head moved a significant distance across the screen from one frame to the next.

The second problem was quite a bit more challenging, and we have not yet found a solution that entirely works. The assumption was made that the speaker would begin the video with his or her mouth closed. We attempted to place what we called an energy “bubble” inside of the lips, so the snake would not be attracted to the center portion of the mouth once the speaker began talking. This was done by finding the minimum and maximum x and y points on the initial snake and using those points as top, bottom, and side coordinates of an ellipse. The ellipse was shrunk to be completely within the mouth in every frame, and every pixel within it was assigned the maximum energy value, which we defined as 10,000,000.

VI. Results

The results of our approach are mixed – in some ways we were very successful, while in others we simply have not yet been able to make it work. On the positive side, we were successful in detecting faces and mouths, and we have a moving snake that stays in close proximity to the mouth every frame. In many cases, when the snake runs its first two or three iterations, it does an accurate job outlining the lips.

Our problems arise as the video continues and the mouth changes shape more and more. The snake has difficulty keeping an accurate shape and tends to collapse on itself over time as the mouth is opened and closed. Forcing a higher energy level at the innermost part of the mouth should help with this problem, but this idea needs further development time to prove it could be a valid solution. Also, the current program does not run as fast as we would like.

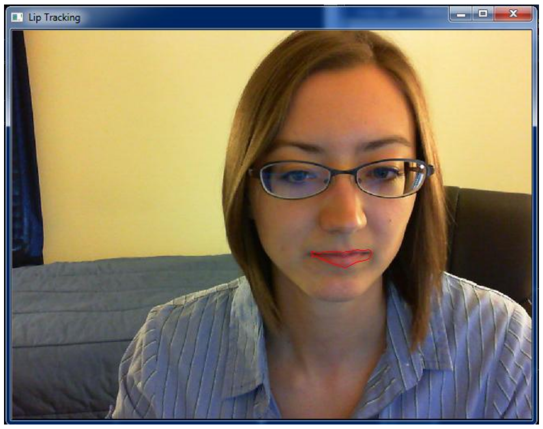

Fig. 6. Snake outlining lips when the mouth is closed.

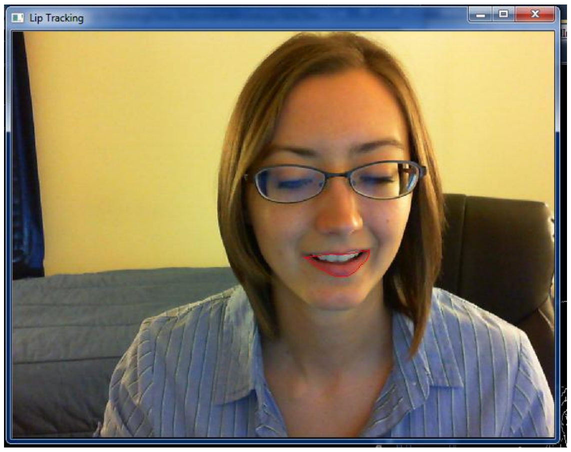

Fig. 7. Snake outlining lips when the mouth is open.

VII. Future Work

We started the project with an aim to develop a program which can be used towards facial expression analysis. Although there is a long way to go to build a program to detect expressions, next we will focus on interpreting the various lip movements combined with eye movements using a training system. The main objective is to be able to detect the alignment between the speech and the expressions of politicians in news channels and presidential campaigns.

VIII. Conclusion

In this project, we succeeded in tracking the lips in the starting few frames. We used face detection and then mouth detection in estimating the approximate location of the mouth relative to the face. We used the mouth box coordinates to initialize the snake close to the lips. The snake then converges to the lips using the dynamic programming approach.

The image energy used here is derived from a combination of energies from a psuedo hue image and a canny edge operation applied on an image. To extract the psuedo hue image, we filtered the pixels which have the color of lips detected by pseudo hue and luminance (brightness) of image. The gradient of the pseudo hue image is also used to calculate the total energy of the image. We also attempted to use constraints that stop the snake from collapsing more than a threshold by maximizing the energy of pixels which lie certainly inside the lip boundary. For this to work, we would have to assume that the lips in initial frame are in a closed position so we can extract the region which will always be inside the lip boundary.

We were successful to some extent, but there is much room for improvement. One possible way would be to consider the lip shape and plot the cubic curve to define the lip boundary after obtaining required points when the snake converges. We could also possibly apply training with all kinds of lip shapes to make the tracking more robust.

References

[1] N. Eveno, A. Caplier, and P. Coulon, "Accurate and Quasi-Automatic Lip Tracking," *IEEE Trans. Circuits Sys. Video Tech.*, vol. 14, no. 5, pp. 706-715, May 2004.

[2] M. Barnard, E. Holden, and R. Owens, "Lip tracking using pattern matching snakes,"

ACCV2002: The 5th Asian Conference on Computer Vision, Jan. 23-25, 2002.

[3] Y. Nakata and M. Ando, "Lipreading Method Using Color Extraction Method and Eigenspace Technique," *Systems and Computers in Japan*, vol. 35, no. 3, 2004.

[4] M. Kass, A. Witkin, and D. Terzopoulos, “Snakes: Active contour models,” *Int. J. Comput. Vis.*, vol. 1, no. 4, pp. 321–331, Jan. 1988.

[5] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," in *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*.

[6] C. Jiang, G. Ridge, and J. Fang, "Contour Tracking based on Intelligent Scissors and Snakes," *http://cs.ucla.edu/~garett/dp.pdf*