# LIP DETECTION AND ADAPTIVE TRACKING

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Benjamin Wang

January 2017

COMMITTEE MEMBERSHIP

TITLE: Lip Detection and Adaptive Tracking

AUTHOR: Benjamin Wang

DATE SUBMITTED: January 2017

COMMITTEE CHAIR: Jane Zhang, Ph.D.
Professor of Electrical Engineering

COMMITTEE MEMBER: John A. Saghri, Ph.D.
Professor of Electrical Engineering

COMMITTEE MEMBER: Wayne Pilkington, Ph.D.
Associate Professor of Electrical Engineering

ABSTRACT

Lip Detection and Adaptive Tracking

Benjamin Wang

Performance of automatic speech recognition (ASR) systems utilizing only acoustic information degrades significantly in noisy environments such as a car cabins. Incorporating audio and visual information together can improve performance in these situations. This work proposes a lip detection and tracking algorithm to serve as a visual front end to an audio-visual automatic speech recognition (AVASR) system.

Several color spaces are examined that are effective for segmenting lips from skin pixels. These color components and several features are used to characterize lips and to train cascaded lip detectors. Pre- and post-processing techniques are employed to maximize detector accuracy. The trained lip detector is incorporated into an adaptive mean-shift tracking algorithm for tracking lips in a car cabin environment. The resulting detector achieves 96.8% accuracy, and the tracker is shown to recover and adapt in scenarios where mean-shift alone fails.

Keywords: Lip Detection, Haar-like Features, Histograms of Oriented Gradients, Local Binary Patterns, Mean-Shift Tracking

# ACKNOWLEDGMENTS

I would like to thank my project advisor, Dr. Jane Zhang, for her support through the entire process of researching and defending my thesis. I would also like to thank Dr. Saghri and Dr. Pilkington for their time serving on my thesis defense committee.

I would also like to thank my family for motivating me to pursue and complete my MSEE.

Finally, I would like to thank Cassie Ding, for staying by my side through all the struggles, and pushing me to do my best.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

# 1. INTRODUCTION

## 1.1. BACKGROUND

Automatic speech recognition (ASR) has achieved over 99% accuracy using acoustic information only, but degrades significantly in poor SNR conditions. Since human speech perception is a multimodal process that includes visual observations, speech recognition systems can be improved by incorporating this visual information [1].

A system that uses both audio and visual information to recognize speech is known as audio visual automatic speech recognition, or AVASR. Research into AVASR has been motivated by voice recognition features in cars. Car cabins are typically acoustically noisy, which severely affects the robustness of audio-only voice recognition systems [2].

This report intends to create a visual front end for an AVASR system that will detect and track a subject's lip in a noisy car cabin. The front end should accurately and automatically detect lips in a car cabin environment. Once acquired, it should reliably track the lip's movement. Various color components and features will be studied to determine the optimal combination for detection and tracking.

## 1.2. ORGANIZATION

Creating the visual front end system is divided into three steps — color analysis, lip detection, and tracking. Color analysis determines the best color spaces to perform lip tracking. Lip

detection trains a lip detector using different features and compares the color spaces suggested by the color analysis step. Tracking implements the results from the color analysis and lip detection into an adaptive mean-shift tracker that will detect and track lips in a given video.

## 2. RELATED WORK

Research in computer vision and object detection has been popular in recent years. In particular, machine learning techniques have shown to be useful for object detection. These techniques seek to convert data into information by extracting patterns from data training sets, which can be supervised (labeled) or unsupervised (unlabeled). Popular machine learning algorithms include K-means, random trees, boosting, and neural networks [3].

Once an object's initial position is determined, tracking algorithms attempt to localize the target over time. Tracking algorithms include single and multiple-hypothesis methods. Single-hypothesis methods such as mean-shift and Kalman filters evaluate a single candidate at a given time. In contrast, multiple-hypothesis algorithms such as grid sampling and particle filters can evaluate multiple candidates simultaneously. The strengths of these different methods can be combined to create a hybrid particle filter mean-shift tracker [4] .

Different techniques have also been explored as the subject of other Master's Thesis projects, and the system presented in this report benefits from these previous works.

Husain in [5] explores different color spaces and illumination compensation techniques to create a robust face detector. The resulting face detection algorithm is then implemented with a lip localization algorithm based on hybrid gradients.

Crow in [6] explores tracking lips using the mean-shift tracking algorithm. Several parameters are investigated, including color space and kernel size/shape.

# 3. COLOR ANALYSIS

## 3.1. AVICAR FRAMES

AVICAR is a collection of videos of individuals reading from several scripts inside a vehicle. A total of 100 subjects are recorded – 50 males and 50 females – spanning Latino, European, and Asian ethnicities to represent various skin tones. Each of the subjects is recorded from four different camera angles under five different noise conditions, e.g. different vehicle speeds and window positions. The purpose of the database is to assist audiovisual research [1].

Color analysis was performed on select frames from AVICAR to determine the best color space to detect and track lips. The ideal color space would produce a large contrast between the background, the subject's face, and the subject's lips.

The analysis was performed by first capturing frames in AVICAR where the subject's lips are visible from all four camera angles. Figure 1 shows an example frame used in the color analysis which measures 720 pixels wide by 480 pixels high. The frame is separated into four frames, each measuring 360 pixels wide and 240 pixels high.

**Figure 1: Example Frames from AVICAR**

After separating the frames, the subject's face was manually cropped from the image. Figure 2 shows an example of separated frame with the face removed. These frames were used to analyze the background color content. For the purposes of this analysis, the subjects' hair and bodies were treated as backgrounds.

**Figure 2: Separated Frames with Faces Removed**

The subject's lips are then manually removed from the face image. Figure 3 shows an example of a face with its lips cropped from the image, which was used to analyze face color content. Figure 4 shows the isolated lips, which was used to determine lip color

content. Care was taken to minimize the number of face pixels in the isolated lips.

**Figure 3: Faces with Lips Removed**

**Figure 4: Isolated Lips**

A total of 25 AVICAR frames were used for this analysis. Since each frame contains four camera angles, the analysis included 100 images each for lips, faces, and backgrounds.

### 3.2. COLOR SPACES

Nine color spaces were analyzed: RGB, xyY, YUV, HSV, nRGB, CIELAB, YCbCr, and c1c2c3. Each color space contains three color components, but components containing only luminance information were ignored, so a total of 19 color components were analyzed.

Different illumination conditions can introduce changes in shadows and highlights. For video tracking, color features should remain constant under these different illumination

conditions, a property known as photometric invariance. Thus, color components that discount illumination artifacts are of interest.

The first color space analyzed was RGB. RGB does not separate luminance and chromatic information, which makes it sensitive to illumination changes in the environment and typically unsuitable for feature extraction. Grayscale, which represents only the luminance information in the image, was also included. Conversion from RGB to grayscale was performed using MATLAB's rgb2gray() function. Inclusion of RGB and grayscale in this analysis was for comparison purposes only.

The second color space analyzed was xyY, which is defined by normalizing XYZ. XYZ is modeled by the spectral power distribution emitted, and it is weighted by a standard human observer's sensitivity to different light wavelengths. In other words, XYZ is derived from the spectral sensitives of cones in humans, thus it represents all colors a human observer can perceive [4]. xy, which represent chromaticity, is defined by normalizing X and Y. The mapping from RGB to XYZ to xy is given by Equation 1. The Y component of xyY represents luminance, so it was not included for analysis.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.607 & 0.174 & 0.200 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.111 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$x = \frac{X}{X + Y + Z}$$

$$y = \frac{Y}{X + Y + Z}$$

**Equation 1: xy Mapping**

The third space analyzed was YUV. YUV is a linear space that separates luminance information Y and chrominance information U and V. The mapping from RGB to YUV is given by Equation 2. Conversion to YUV was performed using MATLAB's xyz2uvL() function.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.113 \\ -0.147 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

**Equation 2: YUV Mapping**

The fourth space analyzed was HSV. The V component represents luminance, so it is not included in this analysis. The hue H and saturation S components are insensitive to illumination changes. Hue corresponds to the pure pigment and saturation corresponds to the amount of white pigment. Conversion from RGB to HSV is given in Equation 3 and was performed using MATLAB's rgb2hsv() function.

$$\delta = max(R, G, B) - min(R, G, B)$$

$$h = \begin{cases} \dfrac{G-B}{\delta}, & R = max(R,G,B) \\ 2 + \dfrac{B-R}{\delta}, & G = max(R,G,B) \\ 4 + \dfrac{R-G}{\delta}, & B = max(R,G,B) \end{cases}$$

$$H = 60h$$

$$V = \max(R,G,B)$$

$$S = \begin{cases} \dfrac{max(R,G,B) - min(R,G,B)}{\max(R,G,B)}, & max(R,G,B) \neq 0 \\ 0, & max(R,G,B) = 0 \end{cases}$$

**Equation 3: HSV Definition**

The fifth space analyzed was normalized RGB, or rgb. Note that nRGB is redundant since $b = 1 - r - g$. Hence, only two components are required to uniquely define each point in the nRGB plane. It is defined by normalizing each component of RGB as shown in Equation 4.

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

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

$$b = \frac{B}{R+G+B}$$

**Equation 4: Normalized RGB**

The sixth space analyzed was CIELAB, also known as L*a*b*. L* represents lightness, a* represents red-green, and b* represents

yellow-blue. Conversion for L*a*b* was performed using MATLAB's rgb2lab() function.

The seventh space analyzed was YCbCr. YCbCr is similar to YUV in that it is linear and separates luminance and chrominance information. Conversion to YCbCr was performed using MATLAB's rgb2ycbcr() function.

The eighth space analyzed was the c1c2c3 color model. The components of c1c2c3 denote the body reflection vector and are defined from RGB components in Equation 5. The components are invariants for the dichromatic reflection model.

$$c_1 = \tan^{-1}\left(\frac{R}{max(G,B)}\right)$$

$$c_2 = \tan^{-1}\left(\frac{G}{max(R,B)}\right)$$

$$c_3 = \tan^{-1}\left(\frac{B}{max(R,G)}\right)$$

**Equation 5: c1c2c3 Color Model**

3.3. COLOR HISTOGRAMS

Figure 5 through Figure 13 show the resulting histograms of each color component. Each plot contains the histograms of the sample lips and faces. All histograms were normalized so the values sum to one, making them consistent with probability distributions. Apart from RGB and gray, the histogram bin ranges were normalized to values in $[0,1]$. Table 1 compiles the results of the

histograms, showing the means and standard deviations of the lips and faces for each color component.

**Table 1: Color Analysis Summary**

| | Lips | | Face | | Comparison | |
|---|---|---|---|---|---|---|
| | μ | σ | μ | σ | Δμ | SMD |
| **R** | 120.978 | 55.478 | 137.993 | 58.206 | 14.1% | 0.299 |
| **G** | 108.311 | 49.821 | 130.960 | 53.509 | 20.9% | 0.439 |
| **B** | 99.146 | 51.371 | 115.615 | 54.487 | 16.6% | 0.311 |
| **gray** | 111.086 | 51.370 | 131.313 | 54.752 | 18.2% | 0.381 |
| **x** | 0.377 | 0.021 | 0.371 | 0.017 | 1.7% | 0.337 |
| **y** | 0.376 | 0.021 | 0.384 | 0.019 | 2.3% | 0.443 |
| **u** | 0.222 | 0.012 | 0.216 | 0.008 | 2.7% | 0.610 |
| **v** | 0.333 | 0.008 | 0.334 | 0.007 | 0.4% | 0.158 |
| **H** | 0.829 | 0.170 | 0.789 | 0.288 | 4.8% | 0.181 |
| **S** | 0.208 | 0.113 | 0.185 | 0.094 | 11.0% | 0.223 |
| **r** | 0.372 | 0.029 | 0.359 | 0.021 | 3.3% | 0.500 |
| **g** | 0.333 | 0.019 | 0.344 | 0.017 | 3.5% | 0.648 |
| **a*** | 0.513 | 0.019 | 0.501 | 0.016 | 2.5% | 0.738 |
| **b*** | 0.526 | 0.024 | 0.535 | 0.025 | 1.7% | 0.363 |
| **Cb** | 0.474 | 0.019 | 0.467 | 0.021 | 1.5% | 0.356 |
| **Cr** | 0.523 | 0.018 | 0.515 | 0.017 | 1.6% | 0.471 |
| **c1** | 0.835 | 0.045 | 0.805 | 0.037 | 3.5% | 0.728 |
| **c2** | 0.730 | 0.045 | 0.763 | 0.035 | 4.5% | 0.826 |
| **c3** | 0.672 | 0.076 | 0.684 | 0.059 | 1.7% | 0.171 |

The ideal color components should produce a large separation between the lip and face histograms. Two metrics were used to measure the effectiveness of each color component for feature extraction. The first is the percent difference between the lip and face mean. The second is the squared mean difference normalized by variance given in Equation 6. Normalizing to the variance of the histograms eliminates the effect of a large "spread."

$$SMD = \sqrt{\frac{(\mu_L - \mu_F)^2}{\sigma_L \sigma_F}}$$

**Equation 6: Squared Mean Difference Normalized by Variance**

For example, comparing only mean difference suggests that all components of RGB produce a high contrast between faces and lips. However, it does not consider the large standard deviation that is apparent in Figure 5. Normalizing by variance takes this into account and shows RGB is not as effective as other color components.

Visually, the RGB histograms in Figure 5 show a large variance and the lips and face histograms show a large overlap.

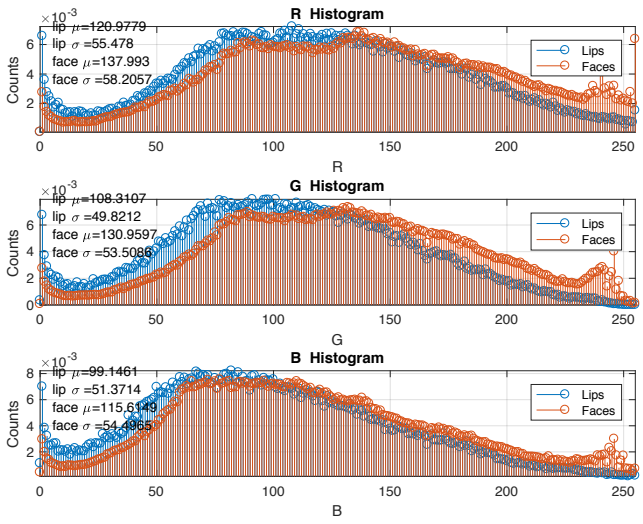


**Figure 5: RGB Histograms**

The grayscale histogram shown in Figure 6 shows similar overlap to RGB as expected.

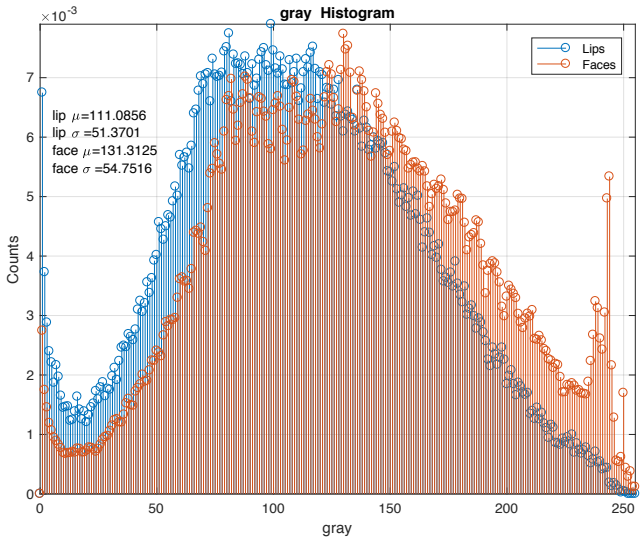


**Figure 6: Grayscale Histogram**

The xy histograms shown in Figure 7, show smaller variances than RGB, but still large overlap.

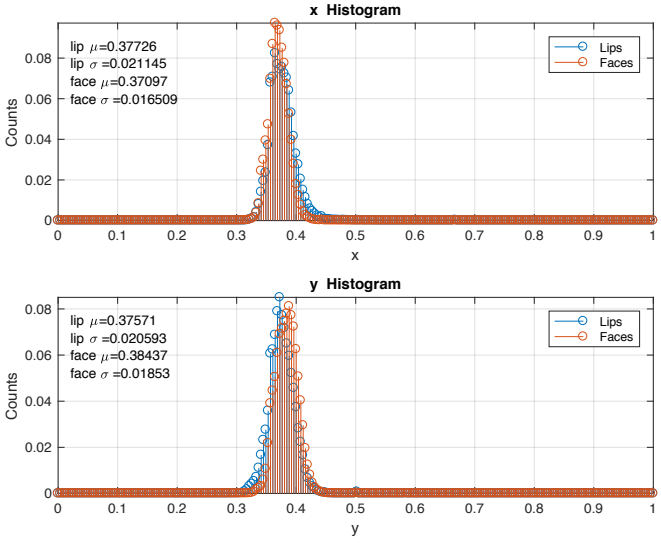


**Figure 7: XY Histograms**

The uv histograms shown in Figure 8 also show a large overlap.

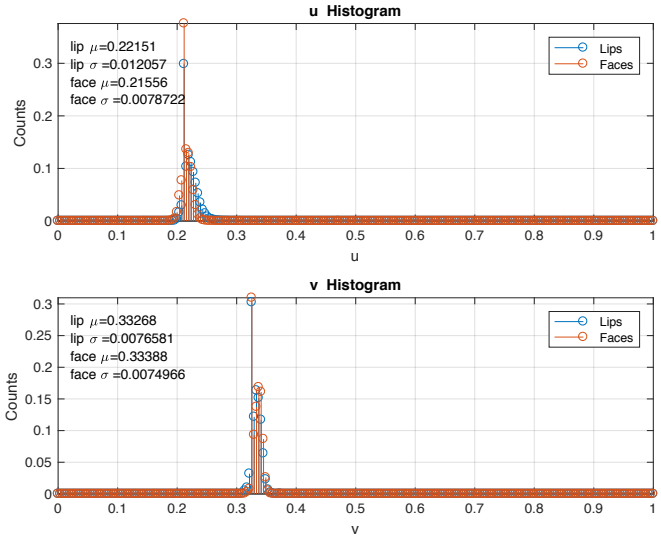


**Figure 8: UV Histograms**

Hue in Figure 9 shows a slight separation between lips and faces, but Saturation still shows a large overlap.

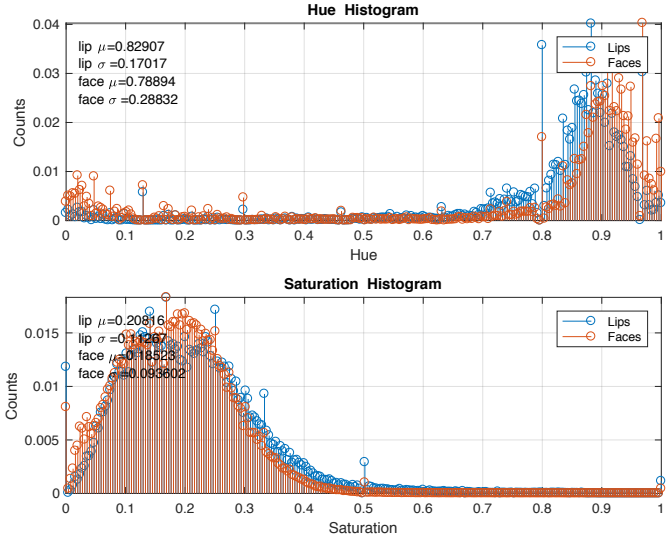


**Figure 9: HS Histograms**

Normalized green shown in Figure 10 also shows large overlap, but performed well in the SMD metric.

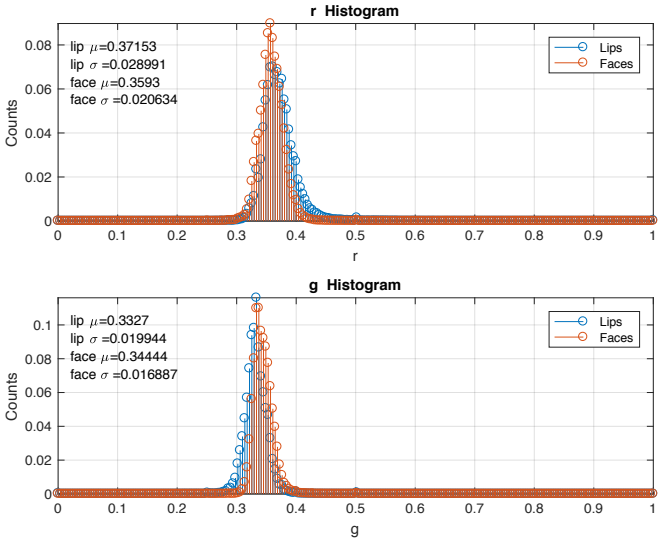


**Figure 10: Normalized RG Histograms**

a* shown in Figure 11 was also a top performer using the SMD metric.

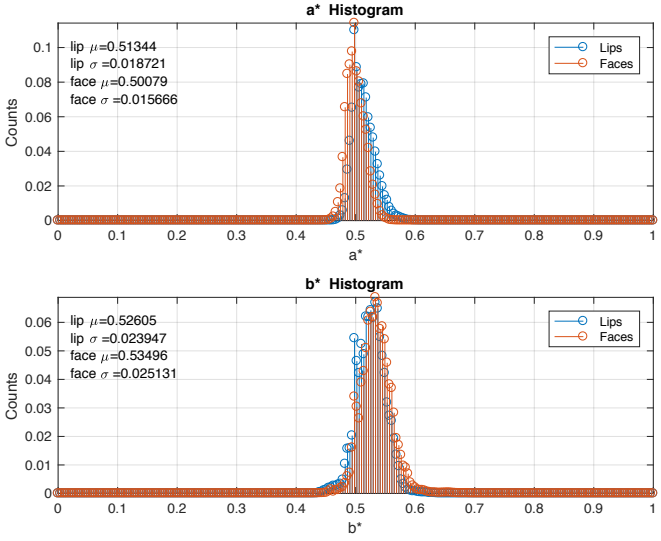


**Figure 11: a*b* Histograms**

Cb and Cr shown in Figure 12 did not perform as well as a* or normalized g.

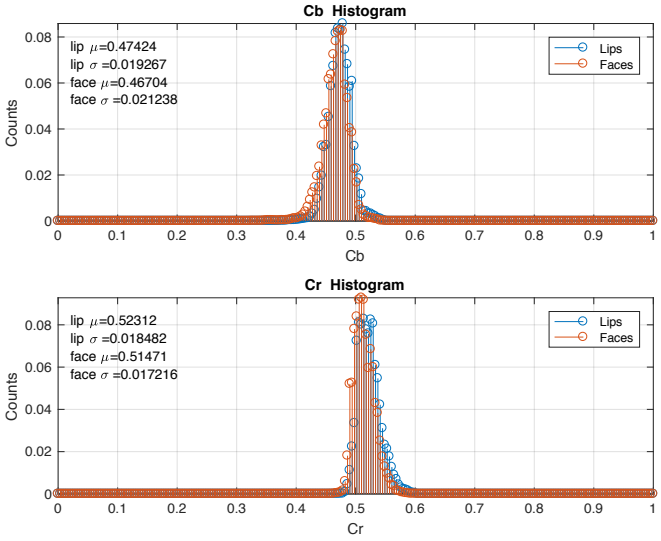


**Figure 12: CbCr Histograms**

c1 and c2 in Figure 13 showed some separation between the face and lip histograms, and their variances are relatively small. As a result, c1 and c2 have relatively high SMDs.

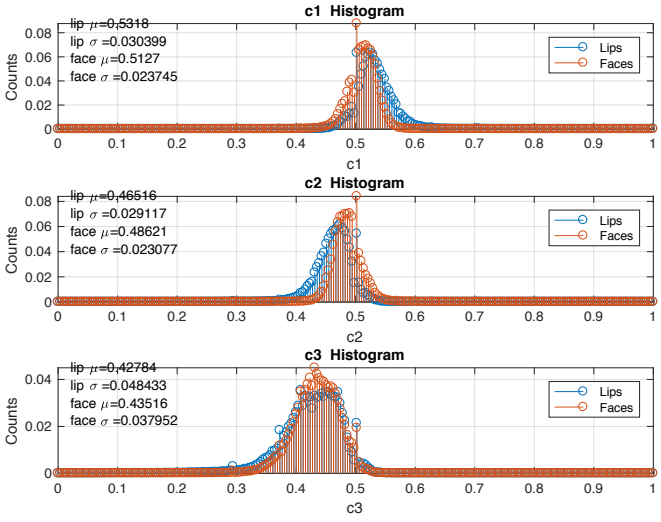


**Figure 13: c1c2c3 Histograms**

In general, the overlaps in the histograms show lips and skin are similar in color. Using the SMD metric, normalized green, a*, c1, and c2 outperformed the other color components. These four components will be further analyzed for lip detection in the next section.

# 4. LIP DETECTION

## 4.1. BACKGROUND

This section tries to determine the best feature and color component to use for a lip detector. The lip detectors were made by training cascaded classifiers. Cascade classifiers are made up of several stages, which are each made up of weak learners. A stage begins by sliding windows of various sizes (constant aspect ratio) across the image and classifying each window position as positive or negative. Positive classification means the desired object may be contained within the window at that position, while negative classification means the object was not found at that window position. Negative positions are eliminated, while positive positions are passed to the next stage.

The next stage repeats this process on the positive regions passed by the previous stage. The purpose of this process is to eliminate negative regions, and hence negative samples, quickly. If the classifier is used on a negative sample, an early stage will likely reject all regions scanned, thus ending the classification process. A positive detection occurs if the final stage classifies a region as positive. As such, this type of classifier is also known as a rejection cascade. Figure 14 shows the basic structure of a face detector rejection cascade; a lip detector rejection cascade would follow the same structure.

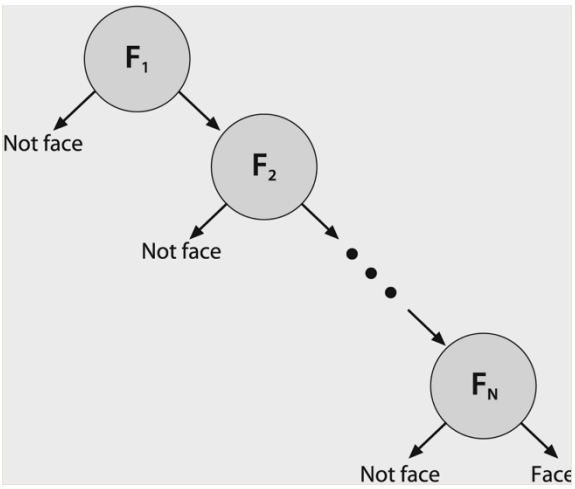


**Figure 14: Structure of Rejection Cascade**

Because of the cascaded nature, a high false-positive is preferable over a high false-negative for each stage. If a stage produces a false-negative, then the classifier cannot recover from this mistake since negative regions are not passed to the next stage. False-positives, however, will be further examined and can be corrected by subsequent stages. A larger number of stages will typically reduce the false positive rate, thus improving detector accuracy.

A feature type is chosen to characterize the objects to be detected, and the classifiers use this feature to classify a given window as lip or not lip. Training was performed using five different color components and three feature types, producing 15 detectors total. The color components chosen were gray, c1, c2, a*, and g. The chrominance components c1, c2, a*, and g were shown to produce the largest contrast between face and lips in the Color Analysis section. Gray represents the

luminance intensity values, and its inclusion will be explained later in this section.

Classifiers were trained using three types of features: Haar-like, Histograms of Oriented Gradients (HOG), and Local Binary Patterns (LBP). The next three sections give a background of these features.

## 4.2. HAAR-LIKE FEATURES

Haar-like features were first explored. The features are defined as difference of sums of the pixels within rectangular regions. Figure 15 shows examples of Haar-like features whose values are calculated as the pixel values in the light rectangles subtracted from the dark rectangles. Although there are only three types of Haar-like features, the entire set of rectangle features includes roughly 180,000 different features. Since only rectangular features are used, orientation is limited to horizontal, vertical, and diagonal.

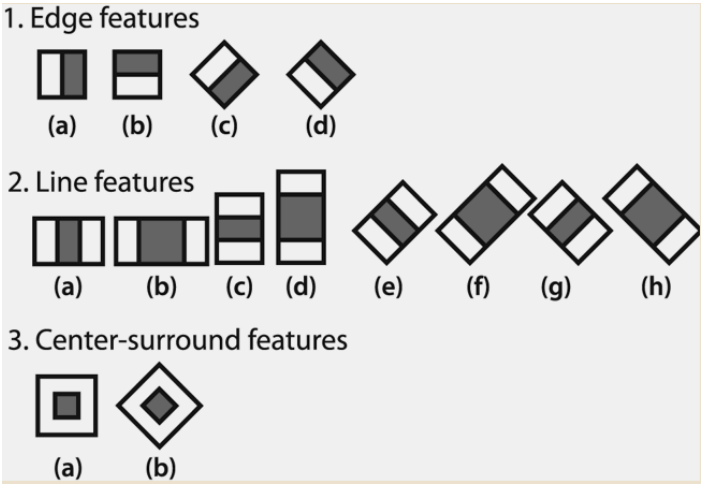


**Figure 15: Example of Haar-like Rectangular Features [3]**

Note this feature set differs from a Haar basis in that its elements contain linear dependencies, a property is known as overcomplete. A true complete basis contains only linearly independent elements.

Figure 16 shows an example of Haar-like features applied to a human face. These features take advantage of the difference in intensity values that is typically present between eyes, eyebrows, and nose.

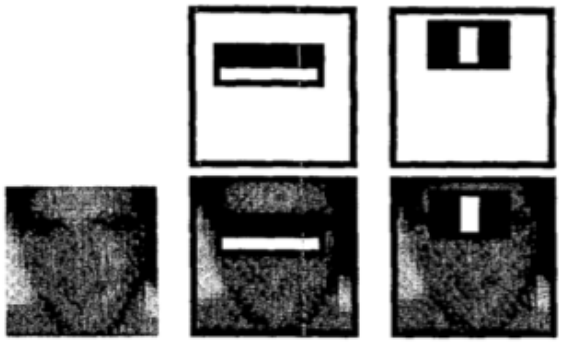

**Figure 16: Examples of Haar-Like Features Applied to Faces [7]**

Intuitively, the rectangular features tend to perform well with block-like features, but they tend to struggle with objects distinguished by their outline, e.g. tree branches and mugs.

### 4.3. HISTOGRAMS OF ORIENTED GRADIENTS (HOG)

Rather than color histograms, targets can also be represented using orientation histograms, which are constructed using the image gradient. To calculate the gradient, partial derivatives of the image must first be calculated using Sobel operators or similar methods. Equation 7 shows 3×3 Sobel kernels that are

convolved with the image $I(x,y)$ to estimate the first partial derivatives in the $x$ and $y$ directions.

$$\frac{\partial}{\partial x} I(x,y) \approx \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -2 \end{bmatrix} * I(x,y)$$

$$\frac{\partial}{\partial y} I(x,y) \approx \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * I(x,y)$$

**Equation 7: 3×3 Sobel Operators**

The image gradient $\nabla I(x,y)$ is then defined as the vector given in Equation 8, along with the gradient magnitude and orientation. Note this definition is identical to the gradient of a two-dimensional continuous function.

$$\nabla I(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} I(x,y) & \frac{\partial}{\partial y} I(x,y) \end{bmatrix}$$

$$|\nabla I(x,y)| = \sqrt{\left(\frac{\partial}{\partial x} I(x,y)\right)^2 + \left(\frac{\partial}{\partial y} I(x,y)\right)^2}$$

$$\angle \nabla I(x,y) = \tan^{-1}\left(\frac{\frac{\partial}{\partial y} I(x,y)}{\frac{\partial}{\partial x} I(x,y)}\right)$$

**Equation 8: Image Gradient Vector, Magnitude, and Orientation**

Histograms of Oriented Gradients (HOG) intend to characterize local object appearance and shape using the distribution of local intensity gradients. The orientation histograms are formed by

binning the image gradient magnitude and orientation, like a two-dimensional color histogram.

The image is divided into smaller cells and gradients are calculated for each cell. Because of the local nature of the characterization, the authors in [8] found adding a buffer around the edge helped with classifier performance.

Figure 17 shows HOG features on a lip training sample as a grid of rose plots. The length of the petal of each rose indicates the contribution of that edge direction. Note the buffer around the lips still contains information about the lip's edge.

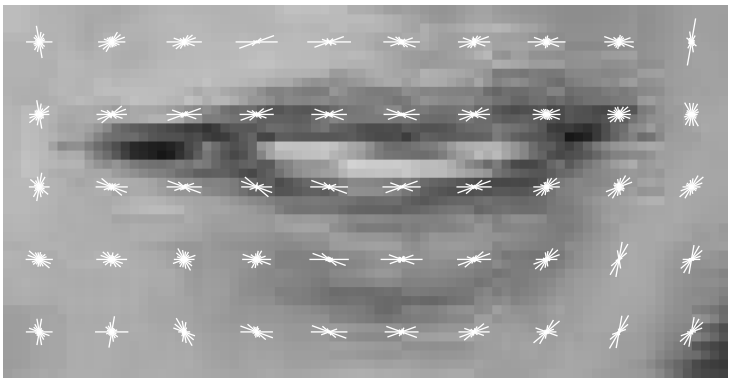

**Figure 17: Visualization of HOG**

### 4.4. LOCAL BINARY PATTERNS (LBP)

Local Binary Patterns were initially intended to characterize textures of materials such as wood, cotton, and sand by modeling their patterns. This is done by analyzing the grayscale values of circularly symmetric neighborhood sets distance $R$ from a given center pixel as shown in Figure 18. Interpolation is used to

determine the grayscale values of locations that do not fall at the center of a pixel.

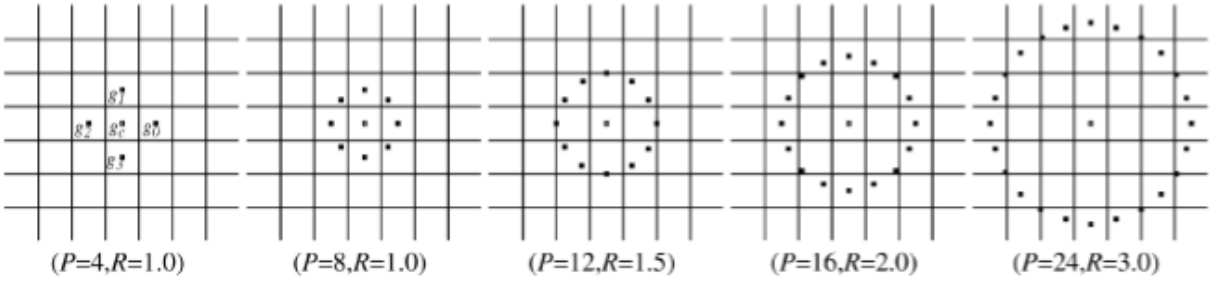


**Figure 18: Circularly Symmetric Neighbor Sets [9]**

Consider the case where $P = 4$ and $R = 1.0$ shown on the left side of Figure 18. The texture $T$ of this neighborhood in a grayscale image can be defined by the vector in Equation 9. The components of $T$ are the differences between the center pixel and its neighborhood pixels.

$$T = \langle g_0 - g_c, g_1 - g_c, g_2 - g_c, g_3 - g_c \rangle$$

**Equation 9: Texture Operator**

To make the texture immune to luminance shifts, only the signs of the differences are considered as shown in Equation 10, where $s(x)$ is the unit step function ($s(x) = 1$ when $x \geq 0$, $s(x) = 0$ otherwise). Mean illumination variances should not change the sign of the differences between these intensity values.

$$T = \langle s(g_0 - g_c), s(g_1 - g_c,), s(g_2 - g_c), s(g_3 - g_c) \rangle$$

**Equation 10: Scale Invariant Texture Operator**

The texture is now defined by binary values based on whether the local neighborhood intensity increases or decreases compared to the center pixel, hence the name Local Binary Pattern. Finally, the LBP at radius $R$ with $P$ number of pixels is given by Equation 11. This is simply the weighted sum of the texture components, which converts the binary string to its decimal equivalent.

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p$$

**Equation 11: Local Binary Pattern Definition**

LBP can be further adapted to achieve rotational invariance, but is outside the scope of this report.

## 4.5. TRAINING

Training was performed using a method known as adaptive boosting, or AdaBoost. Each stage in a rejection cascade is composed of several weak classifiers that each use a single feature decision stump. As shown in Equation 12, a weak classifier using feature $f$ produces a positive or negative vote depending on the feature's value $v$ and the chosen threshold $t$.

$$f_i = \begin{cases} -1, & v_i < t_i \\ 1, & v_i \geq t_i \end{cases}$$

**Equation 12: Decision Stump**

The boosting algorithm then builds a strong classifier as a weighted sum of the weaker classifiers. The decision of a strong classifier $F$ composed of $n$ features is then given by Equation 13,

where $w$ is the weight calculated by the boosting algorithm and $sgn(x)$ is the signum function.

$$F = sgn(w_1 f_1 + \cdots + w_n f_n)$$

**Equation 13: Classification Output**

Training using the boosting algorithm requires a large set of positive and negative samples, where the number of negative samples is typically double the number of positive samples. The positive samples contain the object that the detector is being trained to find (lips), while negative samples do not contain the intended target, e.g. eyes, noses, car cabin.

The positive samples consisted of lips cropped from frames where faces and lips could be clearly seen. A total of 75 frames were used, where each frame included four camera angles, thus producing 300 positive samples total. Example positive samples are shown in Figure 19. Note that the positive samples include a border of face pixels. This extra buffer allows features like HOG to characterize the outline of the lips.

**Figure 19: Positive Samples**

The images that remained after the lips were cropped from the positive samples were used as negative samples. Additionally, frames in AVICAR where lips were occluded were also used as

negative samples. Examples of the negative samples used are shown in Figure 20 and Figure 21. A total of 700 negative samples were used.

**Figure 20: Negative Samples with Cropped Lips**

**Figure 21: Negative Samples with Occluded Lips**

To produce a meaningful and fair comparison between color spaces, the false positive rates and true positive rates were set to 50% and 99.5% for each stage of all detectors. Similarly, the object training size was set to 32 pixels tall and 58 pixels wide, the average size of the positive samples.

Training was performed using MATLAB's trainCascadeObjectDetector function. The trainer was allowed to maximize the number of stages with the given samples up to 20 stages. On a 2.6 GHz Intel Core i7 computer, the HOG and LBP classifiers trained at approximately 1 minute per stage. The Haar classifiers required considerably more time at approximately 5 minutes per stage.

### 4.6. TEST SETUP

The resulting detectors were tested on 200 test images. All test images were also taken from AVICAR frames and show a single face and lip. They resemble, but are different, from the positive test images used for training.

To eliminate false positives (FP) in the background, a face detector is first run on the image. Face detection is currently very reliable using the Viola-Jones method. MATLAB's implementation of a Viola-Jones face detector correctly located 97% of faces in the test images. Face bounding boxes were added manually to the remaining images. The detected face becomes the new region of interest (ROI) for the lip detector, i.e. the detector will only look for lips within the ROI.

Initial tests showed eyes were often misclassified as lips since they share a similar shape. To further reduce FP, the face bounding box is cut in half so the detector ROI is focused on the bottom half of the subject's face.

Figure 22 shows examples of true positive (TP) outputs from a detector. The "face" bounding boxes are produced by MATLAB's

built-in face detector. The "face bottom" bounding box is the final ROI used by the lip detector. The shaded green region represents the true lip location that was manually determined for each test image. The "lip" bounding boxes are the detections returned by the lip detector.

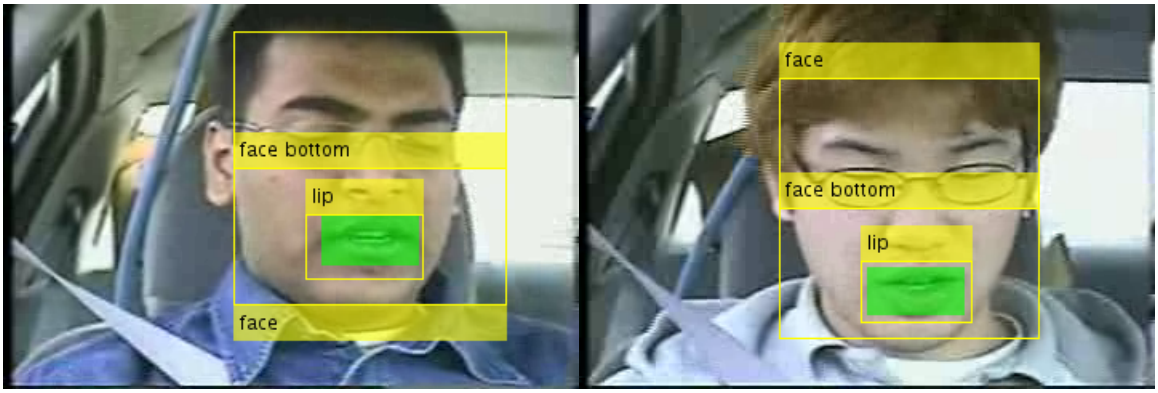


**Figure 22: True Positive Detections**

Overlap ratios were used to gauge the accuracy of each returned detection, specifically the overlap between the detector output (yellow lip bounding box) and the true lip location (green shaded region) in each output image. Equation 14 shows the union and minimum methods, $r_{union}$ and $r_{min}$, for calculating overlap ratios between two regions $A$ and $B$.

$$r_{union} = \frac{area(A \cap B)}{area(A \cup B)}$$

$$r_{min} = \frac{area(A \cap B)}{min\big(area(A), area(B)\big)}$$

**Equation 14: Union and Minimum Overlap Ratios**

Note that both are maximized when $A = B$ and minimized when their intersection is empty. $r_{union}$ is maximized only when $A = B$, while $r_{min}$ is maximized whenever $A \subseteq B$ or $B \subseteq A$, i.e. whenever one region is contained within the other.

Through visual inspection, an appropriate threshold for TP classification was determined to be $r_{min} \times r_{union} \geq 0.3$; a detection is classified as FP otherwise. When used on the images shown in Figure 23, this threshold accurately counts one TP classification and one FP classification for each image.

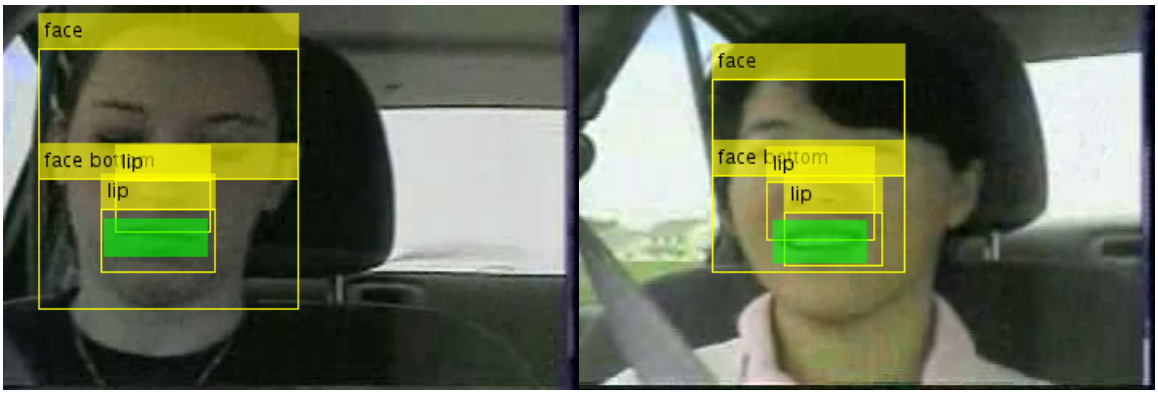


**Figure 23: One True Positive and One False Positive**

True negative (TN) and false negative (FN) counts are determined from the TP and FP counts for each image. A face bottom ROI can be segmented into lip and non-lip regions. Positive detections returned within the non-lip region will be classified as FP. It follows that if no FP detections are returned, no positive detections occurred within the non-lip region. In this case, one TN would be counted since the detector correctly returned negative detections within the non-lip region. Similarly, a FN

would be counted if no TP detections were returned for a given image since the detector incorrectly returned a negative detection for the true lip region.

To summarize, a given detection is classified as TP if $r_{min} \times r_{union} \geq 0.3$ and classified as FP otherwise. For a given image, a FN is counted when a TP detection is not returned, and a TN is counted when a FP detection is not returned.

The performance of the detectors will be judged by accuracy as defined by Equation 15 using the TP, TN, FP, and FN counts.

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

**Equation 15: Detector Accuracy**

These definitions allow quick, consistent, and automated classification of the returned detections. More importantly, it allows objective comparisons of performance between the different detectors.

## 4.7. BASELINE RESULTS

Table 2, Table 3, and Table 4 show the raw counts of each detector. These will be referred to as the baseline (base) results. "Detections" indicates the total number of positive detections returned by the classifier and is the sum of TP and FP.

For comparison, MATLAB's built in mouth detector was also used on the test images. MATLAB's detector was trained using only

grayscale images and Haar features, hence it's results are only shown in Table 3 with the other Haar detectors. Its performance is consistent with the gray Haar detector trained with AVICAR images.

The tables show a*, c1, and c2 detectors return many detections regardless of the feature used, in some cases returning over 400 detections. Since there are only 200 test images, many of these returned detections are likely false positives that will degrade accuracy. The raw counts will be analyzed in further detail later in this section.

**Table 2: Base HOG Detections**

| | gray | a | c1 | c2 | g |
|---|---|---|---|---|---|
| **Detections** | 185 | 418 | 402 | 228 | 157 |
| **TP** | 173 | 49 | 49 | 111 | 136 |
| **FP** | 12 | 369 | 353 | 117 | 21 |
| **TN** | 188 | 14 | 19 | 107 | 180 |
| **FN** | 27 | 151 | 152 | 91 | 64 |

**Table 3: Base Haar Detections**

| | gray | a | c1 | c2 | g | mouth |
|---|---|---|---|---|---|---|
| **Detections** | 320 | 264 | 340 | 351 | 245 | 306 |
| **TP** | 194 | 88 | 81 | 79 | 158 | 196 |
| **FP** | 126 | 176 | 259 | 272 | 87 | 110 |
| **TN** | 93 | 72 | 45 | 38 | 125 | 98 |
| **FN** | 6 | 112 | 119 | 121 | 42 | 4 |

**Table 4: Base LBP Detections**

| | gray | a | c1 | c2 | g |
|---|---|---|---|---|---|
| **Detections** | 254 | 325 | 428 | 338 | 201 |
| **TP** | 186 | 79 | 54 | 67 | 152 |
| **FP** | 68 | 246 | 374 | 271 | 49 |
| **TN** | 136 | 57 | 14 | 38 | 159 |
| **FN** | 14 | 121 | 147 | 133 | 48 |

On average, the trained detectors processed all 200 test images in 10.14 seconds, or 51 milliseconds per image. MATLAB's mouth detector required 11.92 seconds, or 60 milliseconds per image. MATLAB's documentation does not provide specifics on the mouth detector's training parameters, but the longer processing time is likely due to more stages within MATLAB's mouth detector cascade.

Figure 24 shows the calculated accuracy of all 16 detectors. The detectors using grayscale and normalized green outperform detectors trained using a*, c1, and c2 components.

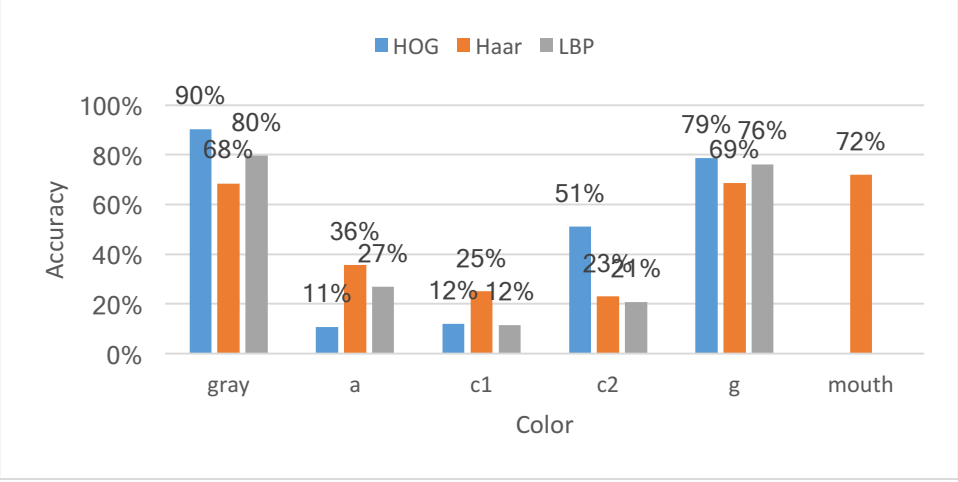


**Figure 24: Base Detector Accuracy**

Figure 25 compares TP counts of all 16 detectors. Gray and normalized green produced the highest TP counts, contributing to their higher accuracy.

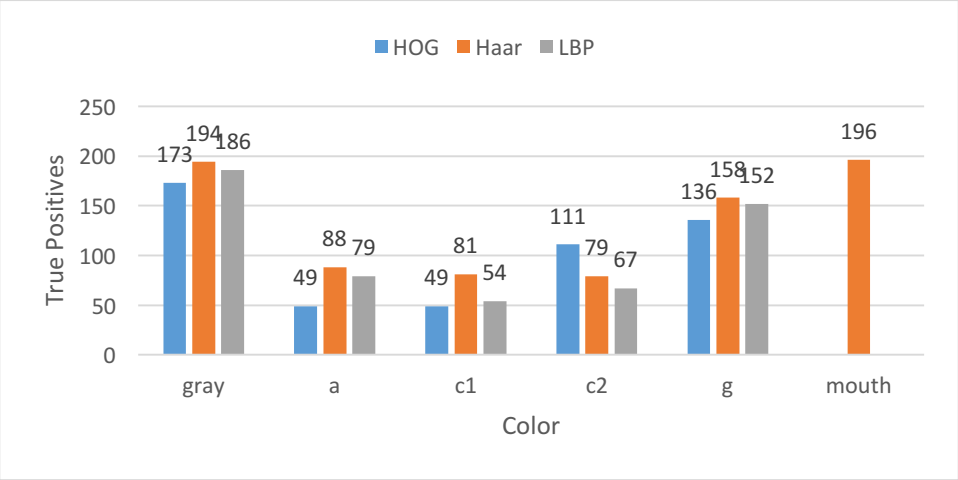


**Figure 25: Base True Positives Counts**

Figure 26 shows the FN counts of each detector. Again, gray and normalized green outperformed the other detectors.

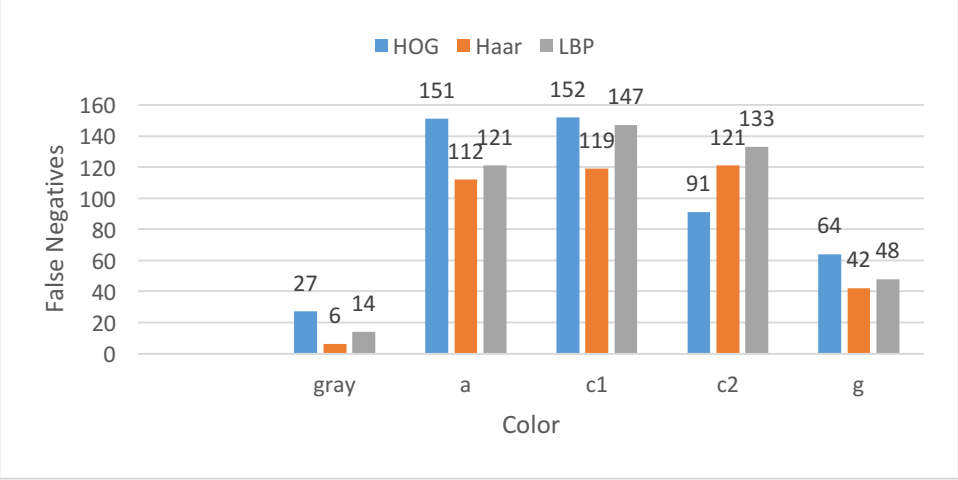


**Figure 26: Base False Negative Counts**

Figure 27 shows examples of false negatives returned by the gray HOG detector. Note the top-left image is counted as one FP since the detected box contains too many non-lip pixels. It also counts as a FN since a TP detection was not returned.

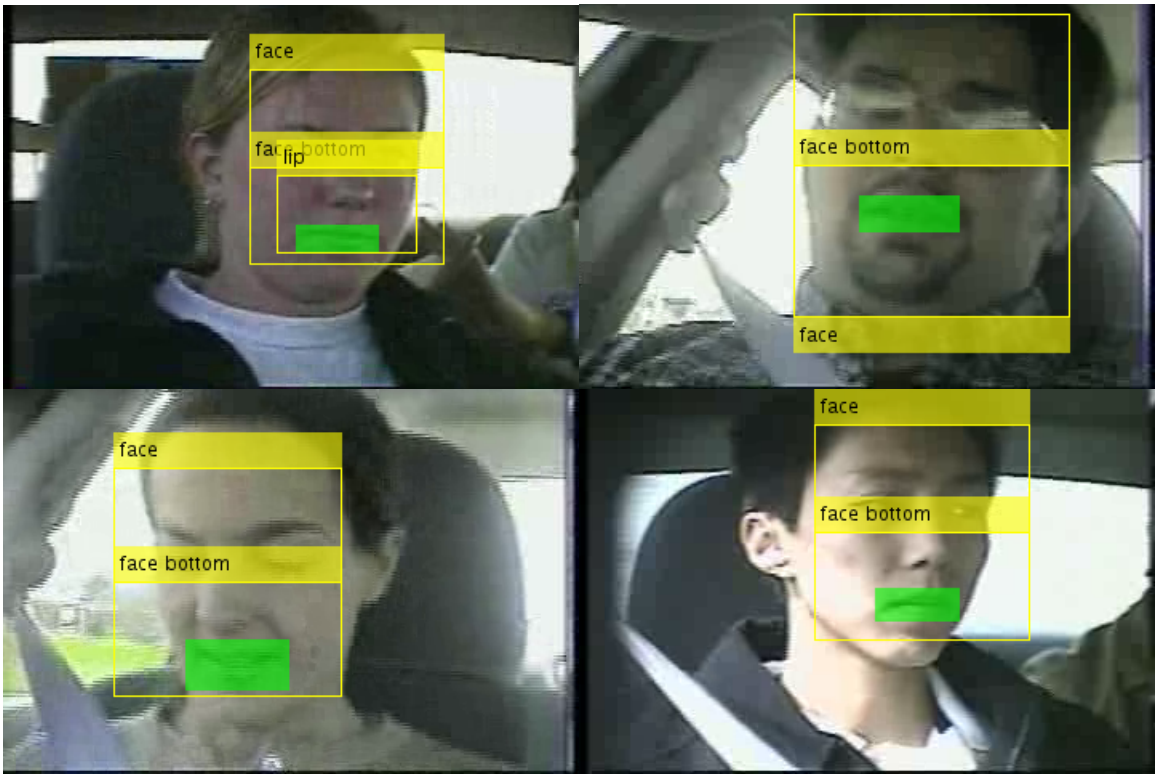


**Figure 27: False Negatives**

A FN count indicates the detector did not find matching features within the test image. The top right image in Figure 27 likely produced a FN since the subject's beard produced different features than the training samples. In addition, poor lighting in the frame reduces the contrast of intensity values around the subject's lips. Figure 28 shows the detections returned by the other HOG detectors for the same image – clockwise from top left: a*, c1, c2, and g.

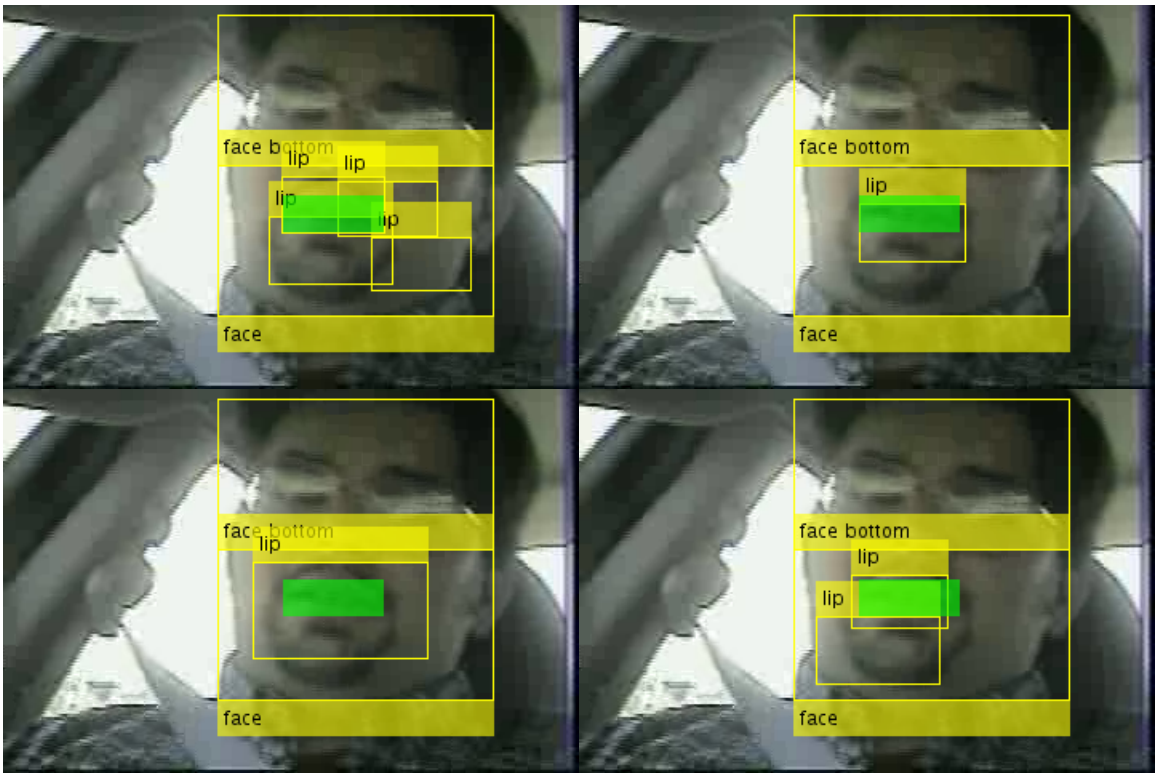


**Figure 28: Detections Returned by a*, c1, c2, and g**

Normalized green returns a FP since the bounding box includes too many non-lip pixels and does not meet the TP threshold established in the previous section. a*, c1, and c2 all return one TP detection each, but a* and c2 also returned other FP detections. These detections show that in poor lighting, chrominance components have an advantage over luminance components.

Despite the results in Figure 28, Figure 29 shows a*, c1, and c2 produced more FP counts that also contributed to their reduced accuracy.

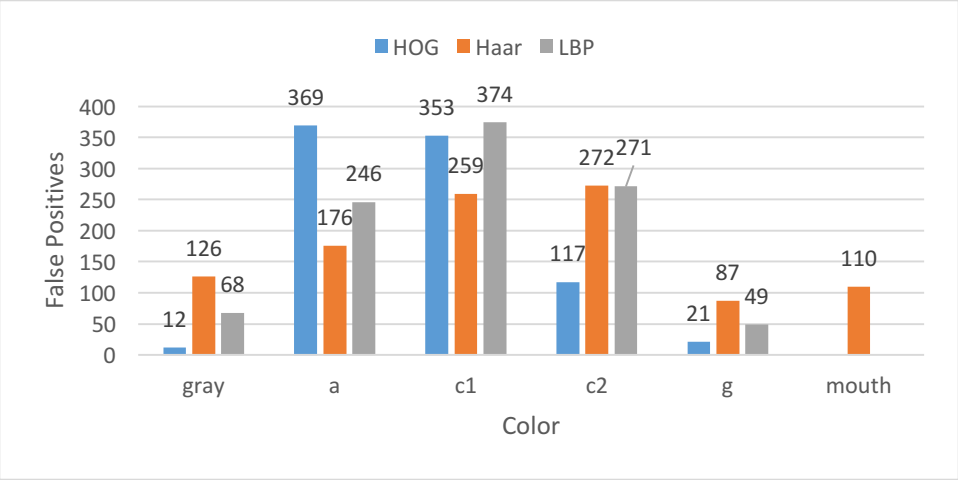


**Figure 29: Base False Positive Counts**

The color components from the previous section were considered because they included only chrominance information and were immune to illumination changes. The features being used by the classifiers rely on differences in intensity values to characterize lips. So, by design, these features are already immune to mean luminance changes.

Separating out the luminance information is useful for tracking, but it eliminates information required to generate features for detection. Figure 30 and Figure 31 illustrate the information that is lost between a test image in gray and c2. c2 clearly shows chrominance information since the subject's face is evenly illuminated. The contrast between skin and lips is also seen as expected. However, illumination differences created by the contours of the subject's face and mouth are not present in the c2 image. These illumination differences are features that help improve detector accuracy.

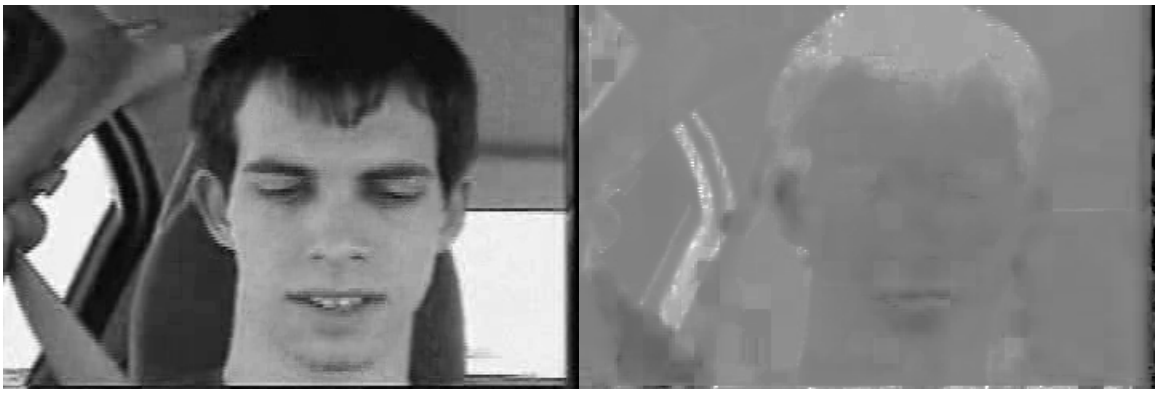

**Figure 30: Gray and C2 Test Image**

**Figure 31: HOG Features on Gray and C2 Lips**

An ideal feature should allow for repeatable detections. In other words, lips in different images should produce similar features. To calculate the consistency of features between images, the HOG vector was calculated for each positive sample used for training (300 images total). The positive samples were resized to 32×58 so the resulting HOG vectors all contained 648 components each. The consistency of the vector components was measured by calculating the standard deviation of each component among the 300 positive training samples. For example, Figure 32 shows the standard deviation of each vector component for gray (blue) and c2 (red).

The consistency of the vectors was measured by calculating the mean standard deviation of all the components. A higher mean standard deviation indicates a larger spread between the HOG vectors, i.e. less consistency.

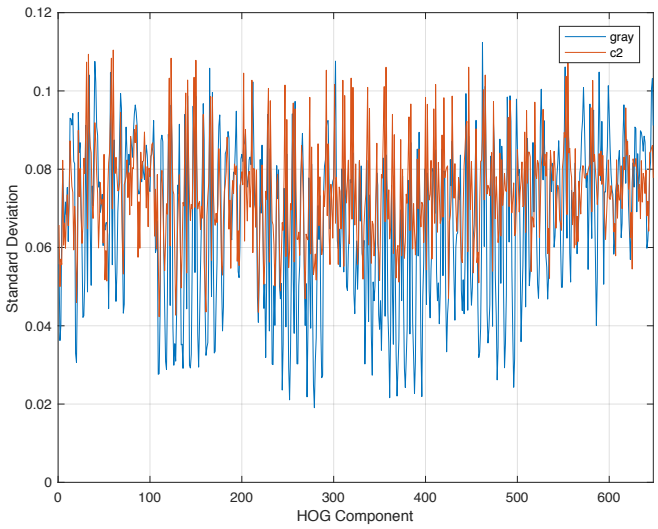


**Figure 32: Gray vs. c2 HOG Vectors**

Visually, c2 (red) appears to have higher standard deviation among its components than gray (blue) indicating less consistency between images. The overall consistency of the vectors was measured by taking the average of all the standard deviation values. Figure 33 plots the mean standard deviation of each color relative to gray.

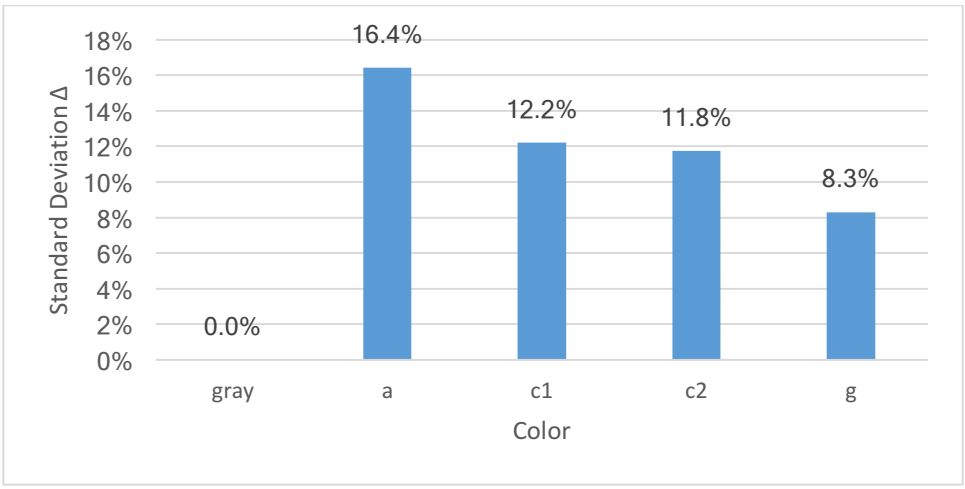


**Figure 33: Mean Standard Deviation of HOG Vector Components**

A higher percentage indicates less consistency between the HOG vectors. This analysis shows gray produces the most consistent HOG vectors, followed by g, c2, c1, and a*. The consistency of the HOG vectors corresponds exactly with the HOG detector accuracy shown in Figure 24 — lower standard deviation results in higher accuracy.

Thus, out of all the color components considered, gray is most effective for lip detection since it preserves the features needed to characterize lips and train the detector.

The following sections explore various ways to improve detector accuracy. Based on Equation 15, this can be done by increasing TP, reducing FP, and/or reducing FN.

Since TP and FN appear in the numerator, they create an upper bound on the accuracy the detector can achieve. The metric involving TP and FN is known as true positive rate (TPR), also known as hit rate, which is given in Equation 16. One hundred

percent TPR indicates the detector returned a TP detection for every test image. TP and FN counts are determined by the trained detector, so the first steps apply image enhancement techniques and adjust detector parameters to maximize TPR.

$$TPR = \frac{TP}{TP + FN}$$

**Equation 16: True Positive Rate**

Maximizing TPR increases the probability of detection, but also typically increases FP counts. Each FP detection reduces the accuracy from the upper bound established by TP and FN, so post-processing techniques are explored to minimize FP detections.

Figure 34 shows the TPR for each detector. Since the a*, c1, c2, and g detectors have significantly lower TPR compared to their gray counterparts, they are abandoned and will not be explored further for improvement.

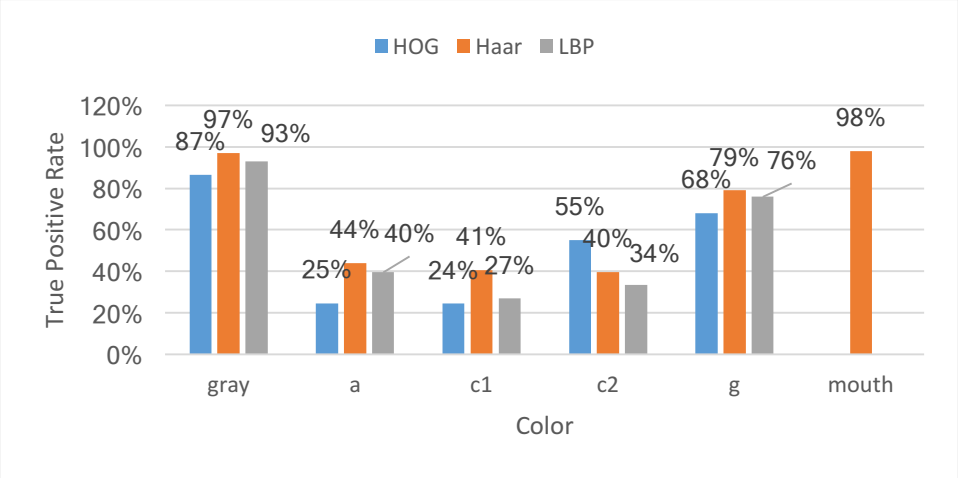


**Figure 34: Base True Positive Rate**

## 4.8. IMAGE ENHANCEMENT

Four pre-processing techniques were explored to enhance the features in the test image before the detector was applied to it – increasing local contrast, gamma correction, histogram equalization, and sharpening. As mentioned previously, the intention of applying these enhancements is to maximize TPR. All four image enhancement techniques were implemented using MATLAB's built-in functions.

For histogram equalization, discrete levels of 64, 128 and 256 were compared. Figure 35 shows the effect on TPR compared to the baseline after applying each of the pre-processing functions to the test images, and Figure 36 shows the effect on accuracy.

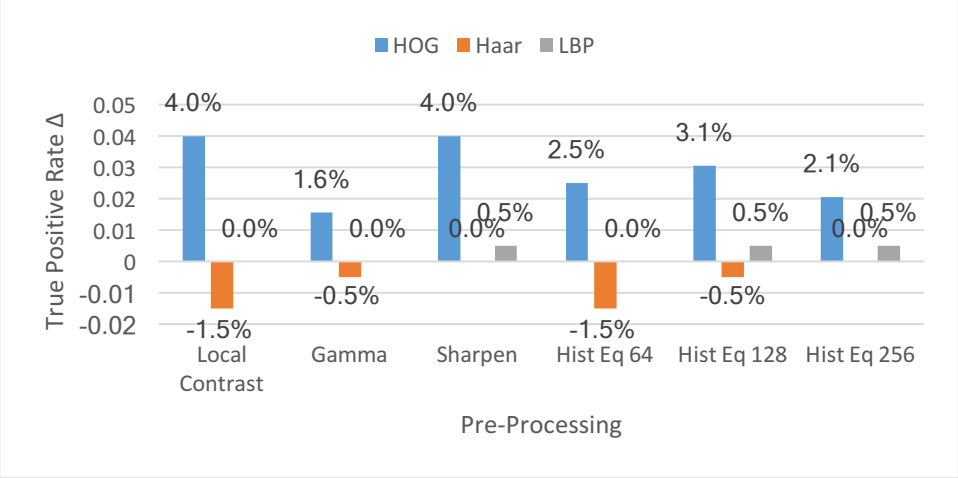


**Figure 35: TPR Δ after Pre-Processing**

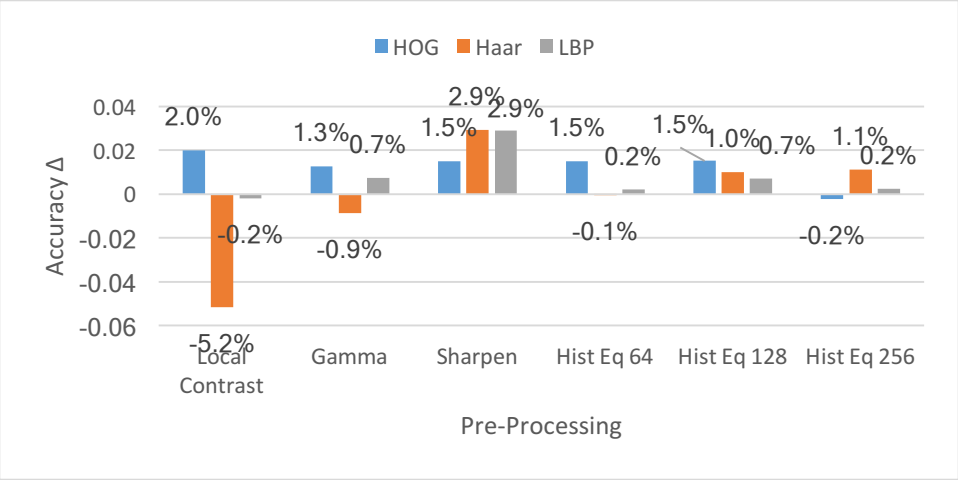


**Figure 36: Accuracy Δ after Pre-Processing**

The pre-processing functions had the largest positive effects on the HOG detector. The techniques had either a negative effect or no effect on the Haar detector. The LBP detector received either no effect or little positive effect. Out of these functions, sharpening produced the most positive results on the HOG detector. Intuitively, sharpening strengthens the edges in the images, which should strengthen HOG features. Although sharpening had no effect on the Haar TPR, it reduced the FP count, thereby increasing the Haar accuracy by 2.9% over baseline. Table 5 shows the raw counts of the gray detectors after sharpening.

**Table 5: Detections after Sharpen**

| | HOG | Haar | LBP |
|---|---|---|---|
| **Detections** | 195 | 306 | 244 |
| **TP** | 181 | 194 | 187 |
| **FP** | 14 | 112 | 57 |
| **TN** | 186 | 101 | 145 |
| **FN** | 19 | 6 | 13 |

Sharpening increases contrast only at edges detected in the image. The other functions increase contrast of the entire image by mapping to new values. So, combining contrast enhancement followed by sharpening was explored next. Figure 37 shows the change in TPR over baseline when enhancing contrast and sharpening the test image, and Figure 38 shows the effect on accuracy.

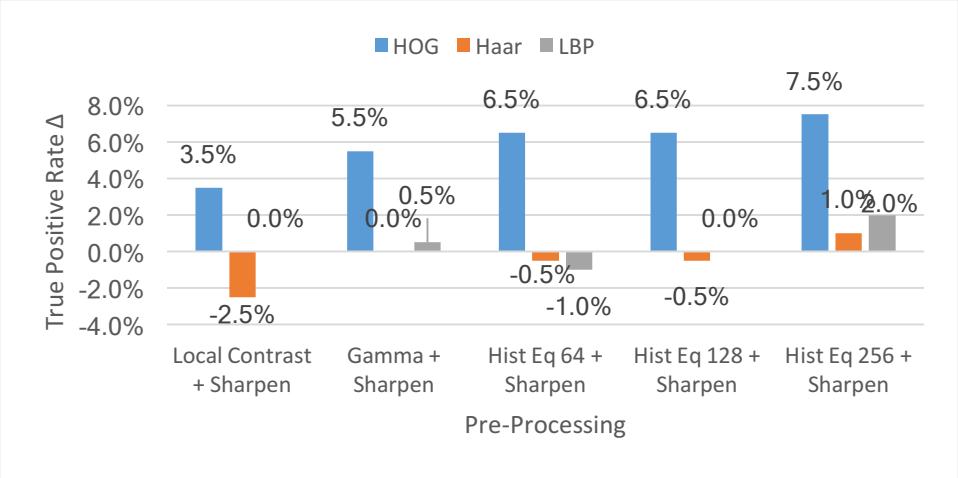


**Figure 37: TPR Δ after Contrast and Sharpen**

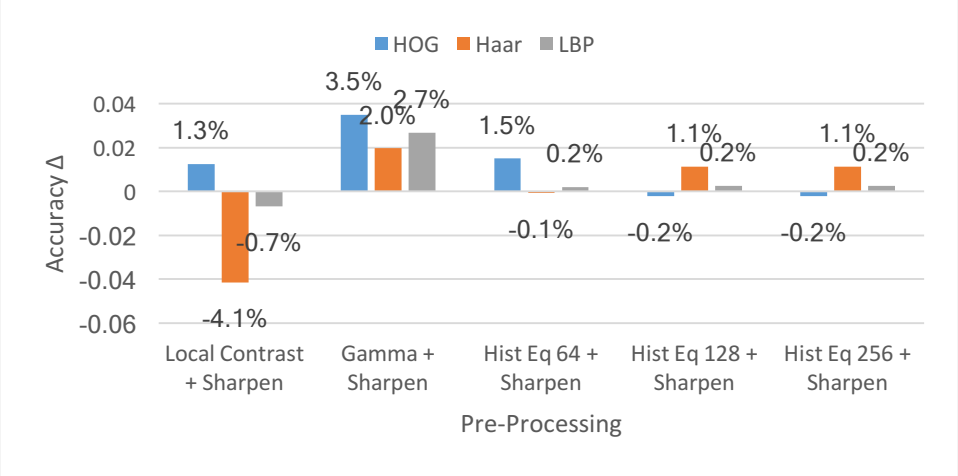


**Figure 38: Accuracy Δ after Contrast and Sharpen**

Histogram equalization with 256 values followed by sharpening produced the largest increase in TPR for all three features. Table 6 shows the raw counts of the gray detectors after this process.

**Table 6: Detections after HistEq 256 and Sharpen**

| | HOG | Haar | LBP |
|---|---|---|---|
| **Detections** | 202 | 293 | 256 |
| **TP** | 188 | 196 | 190 |
| **FP** | 14 | 97 | 66 |
| **TN** | 186 | 113 | 136 |
| **FN** | 12 | 4 | 10 |

4.9. MERGE THRESHOLD

A cascaded object detector performs detection by sliding a window across the image and classifying each window position as positive or negative. The result produces several overlapping positive detections around the true lip. The merge threshold of a cascaded object detector determines the number of overlapping detections required to be counted as positive detection. For example, a merge threshold of 4 requires all positive detections returned by a detector to have at least 4 overlapping detections. Figure 39 shows an example of varying merge thresholds and its effect on returned detections. The resulting bounding box is the average of the overlapping boxes.

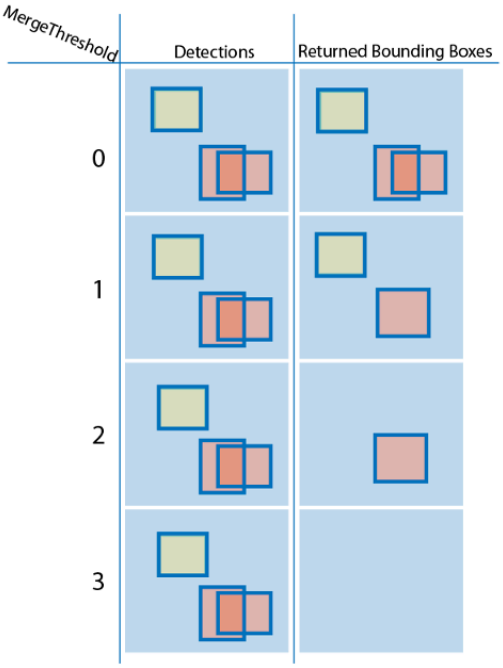


**Figure 39: Merge Threshold**

MATLAB uses a default merge threshold of 4, and that is the value used in the previous results. In general, a lower threshold will produce more detections. In contrast, increasing the threshold may reduce FP and increase TP counts. Figure 40 shows the resulting TPR as the merge threshold was varied between 1 and 4.

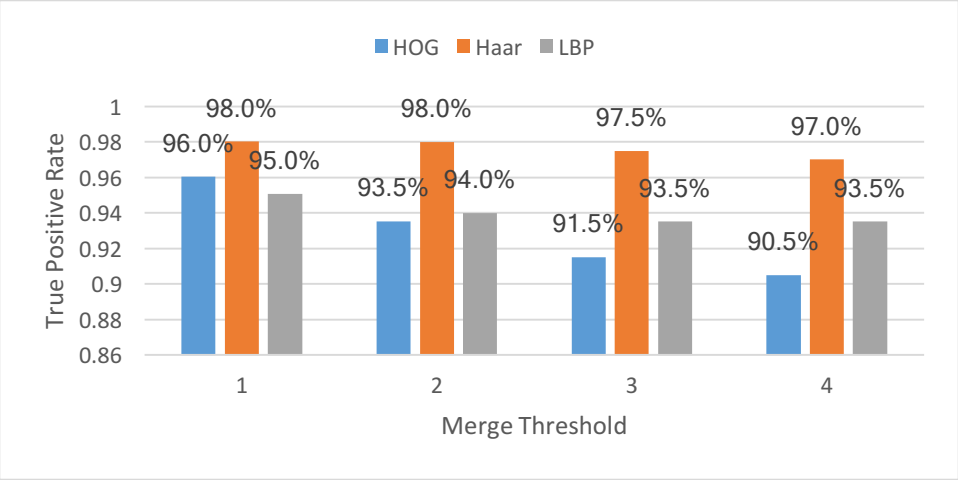


**Figure 40: TPR vs. Merge Threshold**

As expected, TPR increases as the threshold is lowered. However, lowering the threshold also significantly increases the FP count, thereby reducing accuracy. So, the following sections explore methods to reduce FP count. The resulting accuracy is a more effective comparison of different thresholds.

## 4.10. FILTERING BY SIZE

After maximizing TPR, further improvement in accuracy was achieved by reducing the FP rate. The gray HOG detector achieved a maximum 93.5% TPR using histogram equalization (256 values) followed by sharpening. This process produced 14 false positives total. Figure 41 shows four of the false positives remaining for the gray HOG detector.

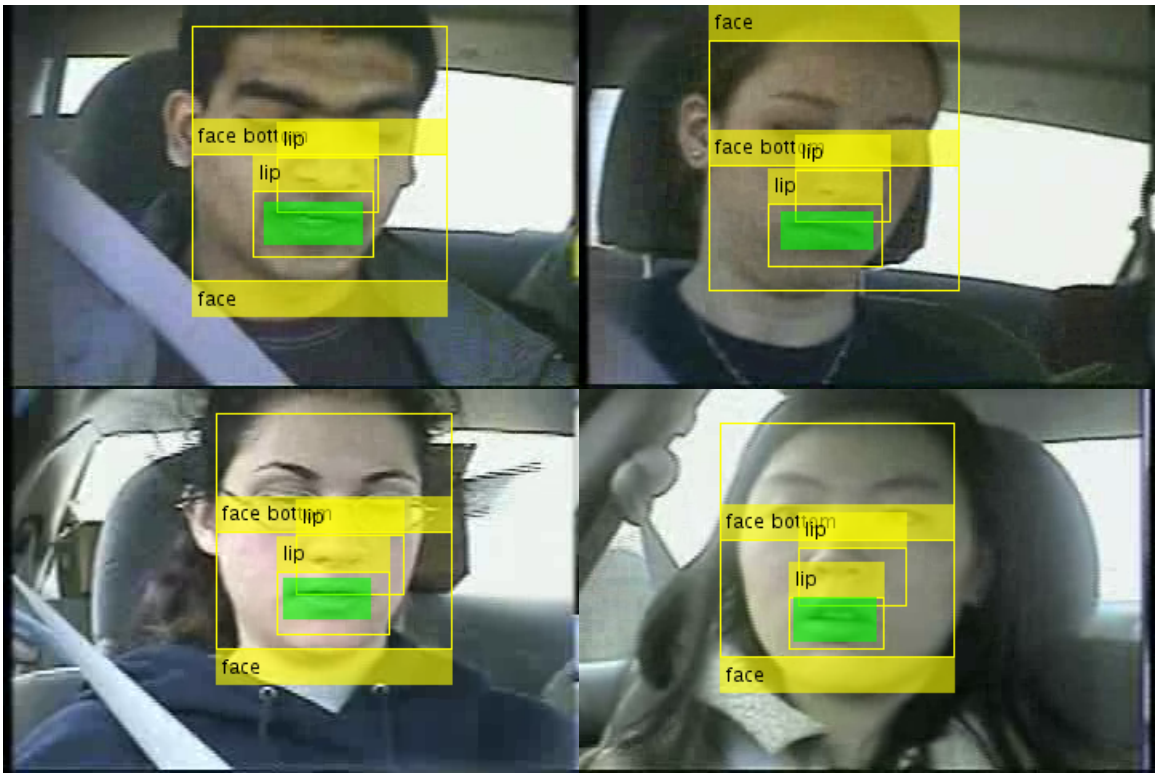


**Figure 41: False Positives after Image Enhancement**

Since the ROI has been limited to the bottom of the face, the lips occupy a relatively large area of the ROI. And within the bottom of the face, only the nose contains different intensity values needed for features. The rest of the skin pixels have relatively uniform intensity. In fact, all 14 remaining false positives were caused by the subject's nose. By selecting the widest lip detected, most false positives caused by the subject's nose can be eliminated since lips are typically wider than noses.

Figure 42 shows the only remaining false positives for the gray HOG detector after filtering the detections by size.

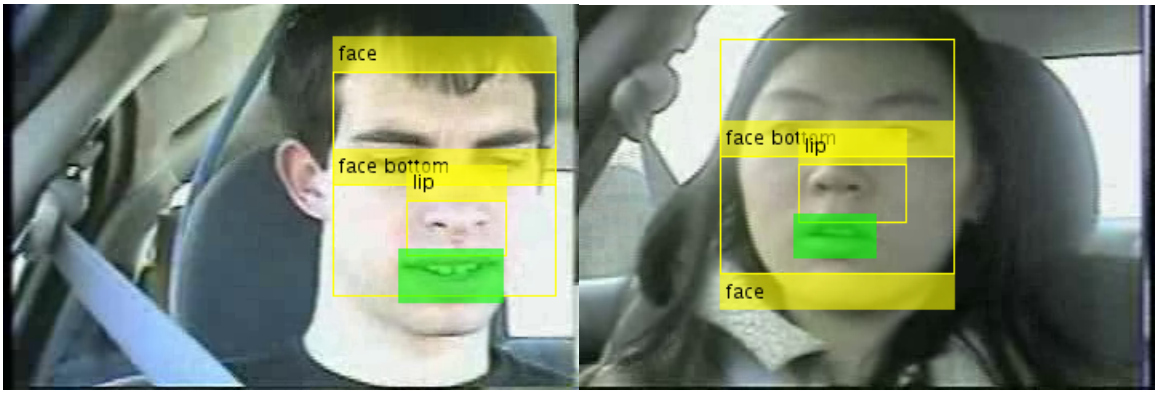


**Figure 42: Remaining False Positives after Size Selection**

In the left picture, the subject's nose was the only positive detection returned for this test image. Since the true lip was not detected, this false positive could not be eliminated through this selection process.

In the right picture, selecting by size did not select the correct detection. The output before size selection is shown in the bottom right of Figure 41. The detection caused by the nose in this image is wider than the true positive lip detection. The merge threshold was then varied between 1 and 16. Figure 43

shows the resulting accuracies as the threshold is adjusted. The maximum accuracy of each detector is labeled.

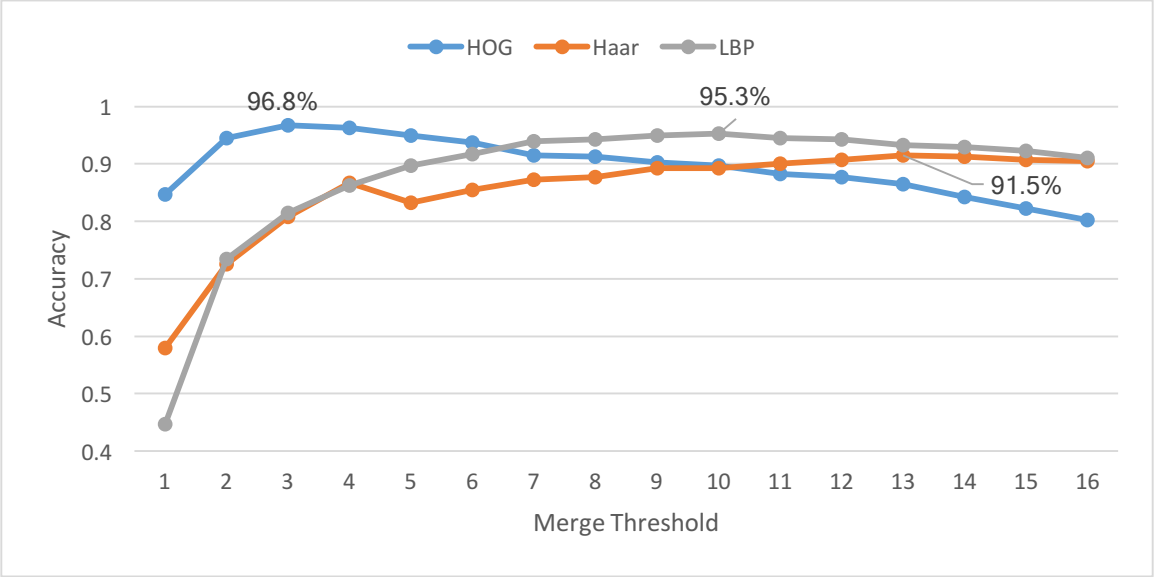


**Figure 43: HOG Accuracy vs. Merge Threshold**

The highest accuracy is achieved by the HOG detector at 96.8% with the threshold set to 3. However, this produced one other FP. Figure 44 shows the output image that caused the FP before the size selection was made. Again, in this case, the nose detection is wider than the lip detection. Since reducing the threshold increased the TP count, overall accuracy still increased.

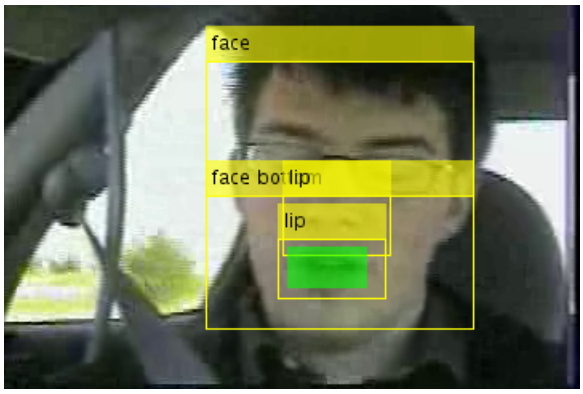


**Figure 44: Additional False Positive**

Table 7 shows the counts after histogram equalization, sharpening, setting merge threshold to 3, and selection by size.

**Table 7: Detections after Size Selection**

| | HOG | Haar | LBP |
|---|---|---|---|
| **Detections** | 193 | 199 | 194 |
| **TP** | 190 | 161 | 160 |
| **FP** | 3 | 38 | 34 |
| **TN** | 197 | 162 | 166 |
| **FN** | 10 | 39 | 40 |

## 4.11. BAYESIAN CLASSIFIER

A Bayesian classifier was briefly explored as a post-processing technique to reduce FP counts. Since the detector ROI only contains the bottom half of the subject's face, only two classes are considered – each pixel is classified as either lip or skin (face). As shown in Equation 17, Bayes' rule can relate the conditional probabilities $P(lip|color)$ and $P(color|lip)$.

$$P(lip|color) = \frac{P(lip)\,P(color|lip)}{P(color)}$$

**Equation 17: Bayes' Rule for Lips**

Combining Equation 17 with a similar relationship for skin pixels forms the decision function shown in Equation 18. For a given pixel color, if this inequality holds, the pixel is classified as lip; the pixel is classified as skin otherwise.

$$P(lip)\,P(color|lip) \geq P(skin)\,P(color|skin)$$

**Equation 18: Decision Function**

c1 and c2 color components were used in the Bayesian classifier since they showed the largest SMD separation between skin and lips. Figure 45 shows the lip c1c2 probability distribution function (PDF) formed by normalizing the c1c2 histogram. The conditional probabilities $P(color|lip)$ and $P(color|skin)$ are determined directly from their PDFs.

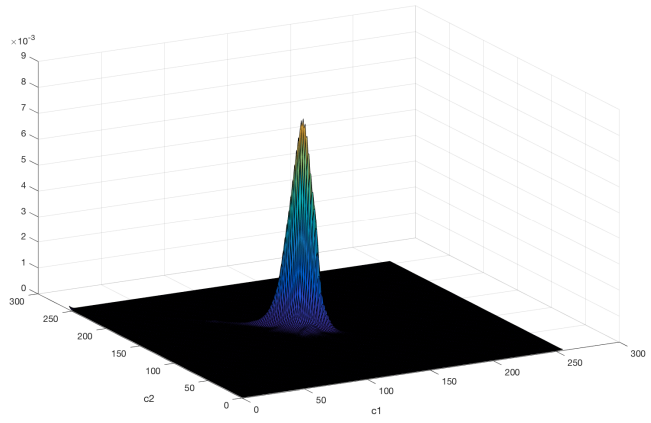


**Figure 45: Lip c1c2 PDF**

$P(lip)$ and $P(skin)$ can be interpreted as the areas each class typically occupies in the detector ROI. These values were estimated using the true lip locations marked in the 200 test

images. On average, the true lip occupied 13% of the face bottom; the remaining 87% is assumed to be skin. Thus, $P(lip)$ and $P(skin)$ were set to 0.13 and 0.87 in the Bayesian classifier.

Figure 46 shows two grayscale test images (left) and the pixels classified as lips by the Bayesian classifier (right). In the top image, the classifier masks most of the non-lip pixels except for the subject's ear. In the bottom image, the classifier removes most of the background, but includes some skin pixels from various facial features, including the subject's nose.

**Figure 46: Lip Pixels after Bayesian Classification**

Post-processing with the Bayesian classifier begin by applying the classifier to each detection returned by the trained lip detector. It classifies each pixel within a given detection as

either lip or skin based on the pixel's c1c2 color. The area of the lip pixels within each detection is then calculated using MATLAB's bwarea() function, which weighs connected groups of lip pixels higher than isolated pixels. Since each image contains one lip at most, the detection with the highest lip area is chosen.

Table 8 shows the counts after histogram equalization, sharpening, setting merge threshold to 3, and selection with the Bayesian classifier.

**Table 8: Detections after Bayesian Classifier**

| | **HOG** | **Haar** | **LBP** |
|---|---|---|---|
| **Detections** | 193 | 199 | 194 |
| **TP** | 190 | 165 | 155 |
| **FP** | 3 | 34 | 39 |
| **TN** | 197 | 166 | 161 |
| **FN** | 10 | 35 | 45 |

The resulting HOG detector achieves the same performance as filtering by size – 96.8% accuracy. The associated Haar and LBP detectors achieved 82.8% and 79.0% accuracy. The merge threshold was adjusted over a limited range to verify that setting the merge threshold to 3 places the HOG detector accuracy at a local maximum. Based on results in the previous section, the optimal merge threshold was not determined for the Haar and LBP detectors.

Similar to filtering by size, the HOG detector with the Bayesian classifier produced 3 FP counts. As mentioned before, the left

image in Figure 42 does not return a TP detection, so it produces a FP count regardless of the selection method used. The other two test images that produced FP counts after Bayesian classification are shown in Figure 47.

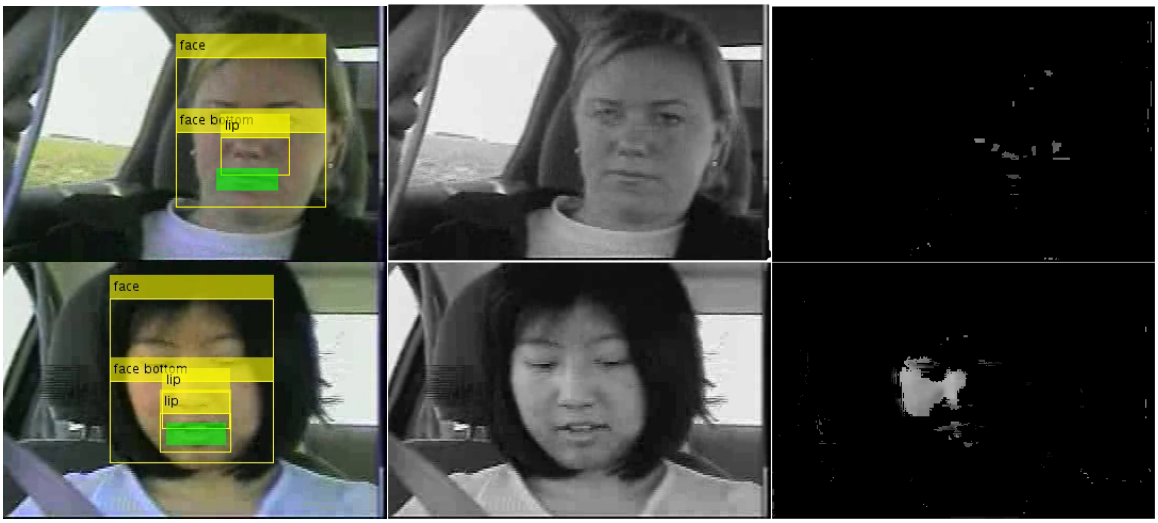


**Figure 47: False Positives after Bayesian Classifier**

Figure 47 shows the detections (left), grayscale test images (center), and the pixels classified as lips (right). Both images produce a FP count since the Bayesian classifier includes too many nose pixels in the lip class.

## 4.12. FINAL LIP DETECTOR

Filtering by size and Bayesian classification produced detectors with identical accuracy, but filtering by size requires less processing. Based on these results, the detector using grayscale, HOG features, 256 value histogram equalization, sharpening, merge threshold 3, and returning only the widest detection was selected to be used in the subsequent tracker.

Figure 48 shows the final flow chart of the lip detector function that will be used for tracking in the next section.

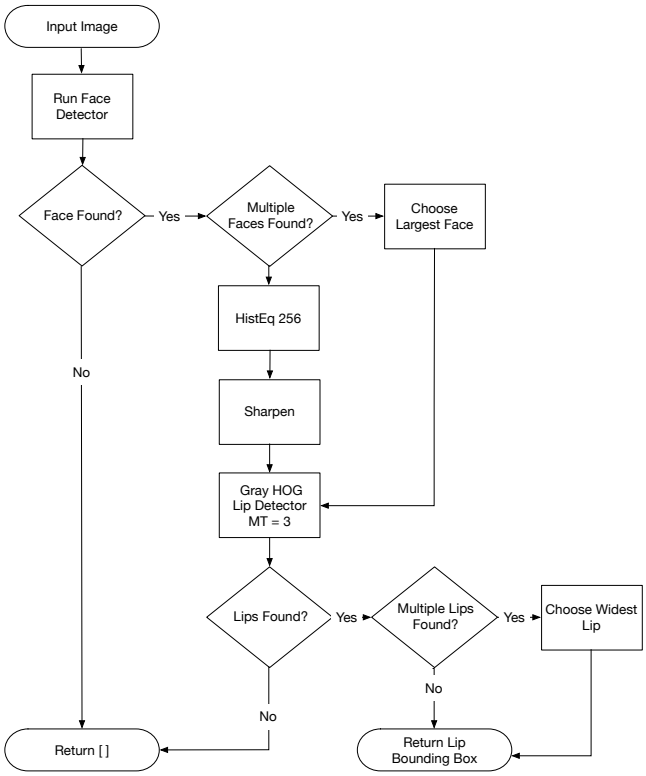


**Figure 48: Lip Detection Flow Chart**

# 5. MEAN-SHIFT TRACKING

## 5.1. BACKGROUND

Mean-shift (MS) tracking algorithm starts by defining a model to represent the object being tracked and an initial location to begin tracking. The features representing the model are chosen to be unique to the target and invariant as the target moves or the scene changes [10]. For this application, the tracker uses the c1 and c2 color components to perform tracking. The previous sections have shown c1 and c2 produce the largest contrast between face and lip pixels. Also, since c1 and c2 represent chrominance components, they are invariant to lighting changes.

The model's color histogram thus acts as its probability distribution. As the video progresses, the algorithm moves the search window toward the area that has a distribution that most closely matches the model's distribution. Similarity between two given distributions $p$ and $q$ with $m$ discrete values can be measured by the Bhattacharyya coefficient given in Equation 19.

$$\rho(p,q) = \sum_{a=1}^{m} \sqrt{p_a \, q_a}$$

**Equation 19: Bhattacharyya Coefficient**

Note that the Bhattacharyya coefficient can be interpreted as the inner product of the vectors $\langle \sqrt{p_1}, \cdots, \sqrt{p_m} \rangle$ and $\langle \sqrt{q_1}, \cdots, \sqrt{q_m} \rangle$. The coefficient is maximized when one is a scalar multiple of the other. If both vectors have unit magnitude, then the coefficient

is maximized if and only if $p = q$ as expected for a similarity measure. The "distance" $d$ between distributions can then be defined as shown in Equation 20 since the Bhattacharyya coefficient has a maximum value of one. MS tracking can be shown to iteratively minimize this distance between model and target.

$$d = \sqrt{1 - \rho(p, q)}$$

**Equation 20: Distance between Discrete Distributions**

The iterative movement of the tracking window is performed by calculating the MS vector, which estimates the gradient vector at the center of the tracking window. Since the gradient points in the direction of greatest change, its direction and magnitude determine the direction and distance to move the tracking window. As desired for a tracking algorithm, the window's movement is large when the gradient has large magnitude, i.e. the desired target is not centered in the window. The MS vector is recalculated after each window movement since moving the window will change the gradient vector. This process is repeated until the MS vector converges to zero, indicating the window has centered on a local maximum.

Intuitively, the tracking window seeks out and centers itself on its center of mass. This is illustrated in Figure 49, which shows an example of the MS algorithm applied to a two-dimensional array of data points. The window starts in a position with an uneven distribution of points. Each iteration moves the window towards its center of mass, i.e. towards the dense cluster of

points, until the window itself is centered on its center of mass.

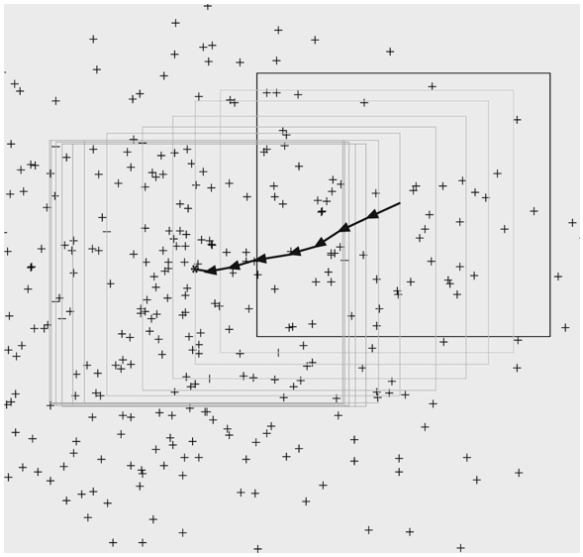

**Figure 49: Mean-Shift Performed on Array of Data Points [3]**

An advantage of MS tracking is its robustness to objects far from the tracking window's initial position. Every iteration only considers the data contained within the tracking window's current position. Hence objects outside of the window will not affect tracking performance. It is reasonable to assume that lips will not move outside of the tracking window between frames in a car environment.

## 5.2. ADAPTIVE TRACKING

In standard MS tracking, the model is static. The algorithm cannot adapt if the target's features change over time, thereby invalidating the initial model. Furthermore, since the tracking algorithm assumes the target is always visible, MS may lose its

target if the target becomes occluded or leaves the frame momentarily.

As an example, the MS algorithm using c1 and c2 color components was run on the first 2000 frames of an AVICAR video. c1 and c2 was used since these components produced the largest SMD. MS tracking using RGB was explored in [6], but was shown to be less effective than purely chrominance components. Figure 50 plots the similarity between model and target against frame number. The general trend of the similarity is decreasing as the video progresses, indicating that the model no longer matches the object within the tracking window.

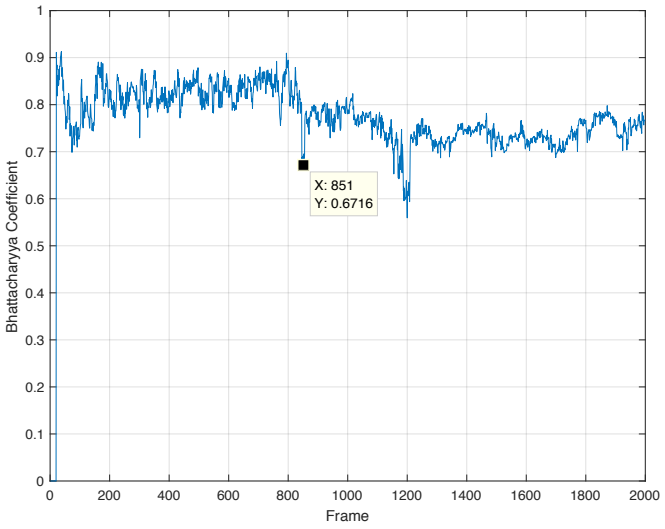


**Figure 50: MS Tracking Model and Target Similarity**

The event corresponding to the dip seen near frame 851 is shown in Figure 51. In these frames, the frame number is printed in the top left, and the lip box indicates the tracking window. The lips were initially tracked correctly, but beginning in frame

835, the subject quickly turns her head. The movement was quick enough to move her lips outside of the tracking window. This cause the tracking algorithm to converge on the subject's eye instead. As discussed previously, MS relies on a portion of the lips remaining within the window. Once the window was focused on her eye, the algorithm did not have a method to reacquire the lips.


**Figure 51: Lost Target**

To give the tracking algorithm the ability to adapt, similarity between the current target and model is calculated at a constant interval of frames via the Bhattacharyya coefficient. If the coefficient value falls below a preset threshold, that indicates the target being tracked differs too greatly from the model, indicating the target has likely been lost.

When this occurs, the script runs the lip detector trained in the previous section on each frame until a new lip is found. The

model is then replaced by this new lip and the MS tracking resumes using this new model.

Figure 52 plots the similarity measure when adaptive tracking is applied to the same video. In this example, the threshold for resetting the model was set to 0.75. Since the model is updated whenever the Bhattacharyya coefficient falls below the threshold, similarity remains more constant than before.

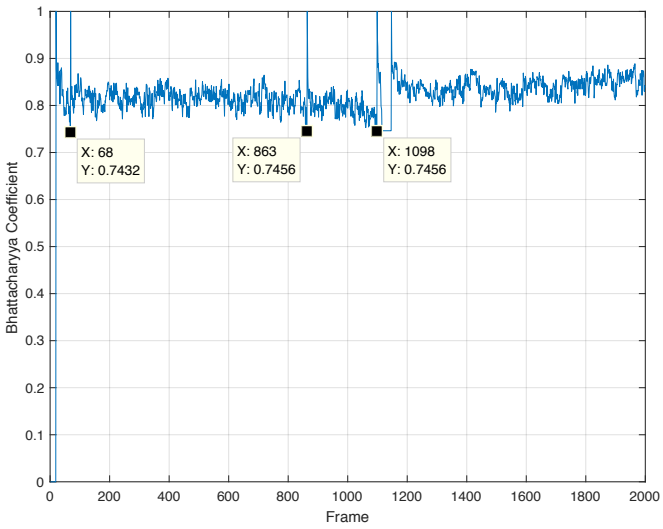


**Figure 52: Adaptive MS Tracking Model and Target Similarity**

Figure 53 shows the same frames as Figure 51 using adaptive MS tracking. In this case, the adaptive algorithm was able to reacquire the lips despite the subject's rapid movement.


**Figure 53: Target Recovered**

Figure 54 shows another case of recovery where the subject's face was turned completely to her side. Tracking is paused starting frame 1116 and the lip detector begins running on each frame. The lip detector does not return a detection until frame 1146, where the tracker then successfully reacquires and tracks the lips until the end of the video.


**Figure 54: Recovery from Head Turn**

## 5.3. RESULTS

The MS and adaptive MS scripts were tested on five videos from AVICAR, each containing 2000 frames and running 1 minute 7 seconds. Both tracking algorithms use an Epanechnikov kernel, a shift-vector threshold of 2, and a maximum of 10 shifts per iteration. Figure 55 shows the flow chart of the adaptive MS algorithm. The non-adaptive MS algorithm omits the similarity comparison at the start of each frame.

Both trackers begin by running the lip detector on each frame until lips are found. Since the same lip detector is used, both trackers begin with their tracking windows at the same frame and initial location, and hence the same model distribution. For all videos tested, the lip detector correctly acquired the subject's lips to create the initial model. The model distribution is formed using its c1 and c2 two-dimensional histogram.

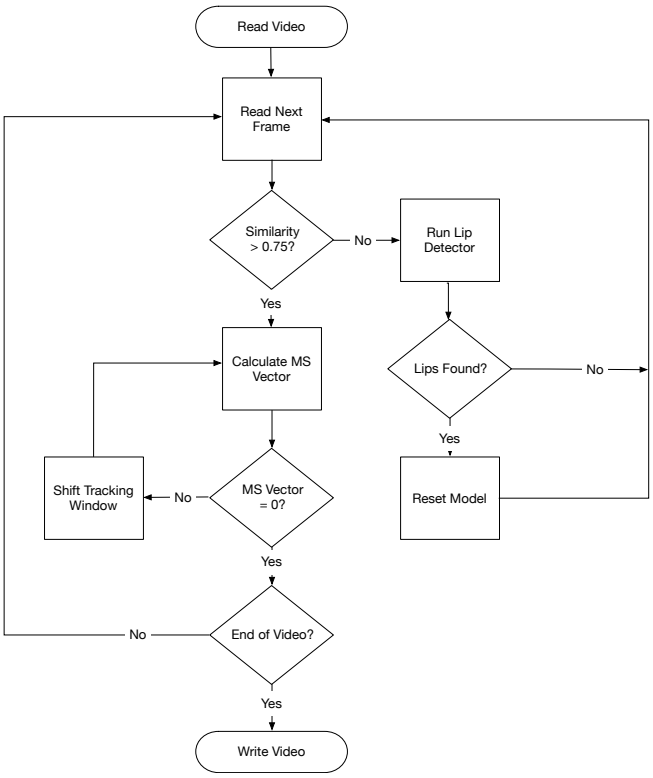


**Figure 55: Adaptive Mean Shift Flow Chart**

The resulting videos were manually scanned to determine the accuracy of the tracker. In four of the videos tested, the MS tracker eventually loses tracking of the lips, while the adaptive tracker can follow the lips until the end of the video. In both cases, the bounding box tends to hover around the true lip location when the subject is not moving. For the adaptive MS tracker, the bounding box is sometimes off-center from the lips, but includes at least 75% of the true lip pixels. In contrast, when the non-adaptive MS tracker loses the target, the bounding box does not include any true lip pixels.

One concern for the adaptive tracker is processing speed. In all tests, the trackers could process the videos faster than real-

time, i.e. less than the video's length of 67 seconds. However, the adaptive tracker adds several extra processes if the similarity falls below its threshold. This would suggest the adaptive tracker may require more processing time than MS tracking alone.

Table 9 shows the average processing time of each video by each tracker. At worst case, the adaptive tracker is 9.7% slower than normal MS. Surprisingly, the adaptive tracker performs faster than MS in most cases.

**Table 9: Average Processing Time**

| | Speed (seconds) | | |
|---|---|---|---|
| **Video** | **MS** | **Adaptive** | **Δ** |
| **1** | 47.8 | 52.4 | 9.7% |
| **2** | 53.8 | 53.3 | -0.8% |
| **3** | 61.3 | 52.7 | -14.0% |
| **4** | 51.8 | 50.6 | -2.4% |
| **5** | 55.8 | 60.2 | 8.0% |

Figure 56 shows the number of shift vectors calculated by each tracker for Video 4, which gives insight into their processing times. The number of shift vectors indicate how many times the tracking window is shifted for a given frame. Note that shift vectors are not calculated until frame 91, which is the first frame where lips are detected. The MS tracker averages 0.85 shifts per frame, while the adaptive tracker averages 0.73 shifts per frame for this video.

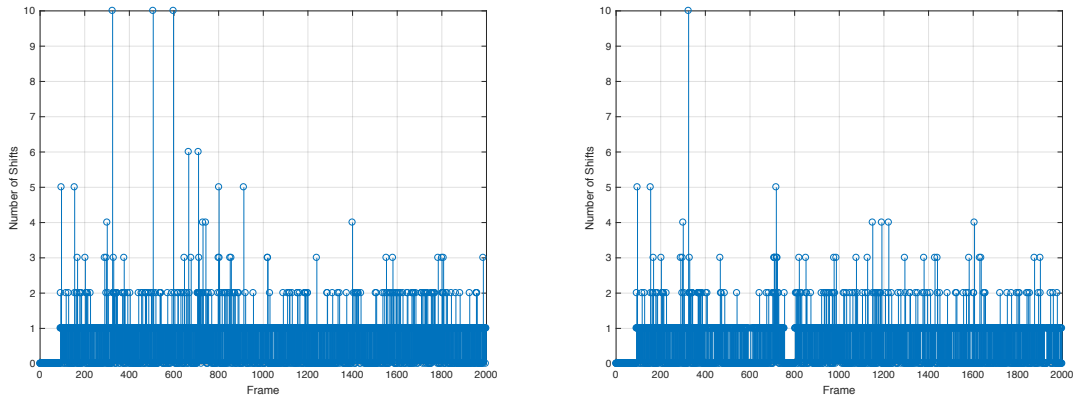


**Figure 56: MS (Left) and Adaptive MS (Right) Shift Vectors**

When the MS tracker loses its target, which occurs in this video, the tracking window requires more iterations to converge on a local maximum. Thus, the extra time required to converge causes the MS tracker to perform slower than the adaptive tracker. Also in this video, the subject turns her head starting frame 768, so no shift vectors are calculated by the adaptive tracker during this time.

# 6. CONCLUSION AND FUTURE WORK

We have maximized the lip detector accuracy using several pre- and post-processing techniques. For pre-processing, histogram equalization using 256 bins combined with image sharpening improved the HOG TPR by 7.5% over baseline with minimal impact on accuracy. Post-processing using size selection and Bayesian classification produced identical accuracy, but size selection was ultimately chosen for its simplicity.

We have also shown that features like HOG, Haar, and LBP rely on differences in intensity values to effectively describe targets. These features are illumination invariant by design. Thus, training cascaded classifiers using luminance values are more effective for lip detection than using chrominance components, since chrominance components eliminate details that can be used as features. Furthermore, using HOG to characterize lips created a lip detector 26% more accurate than using Haar features, and 13% more accurate than using LBP.

MS tracking, on the other hand, relies on features that produce a strong contrast with the background and are invariant to lighting changes. Out of all the color components studied, c2 was shown to produce the largest contrast between face and lip pixels using the SMD metric. c2 produced a higher contrast by 11% when compared to a* and 22% when compared with normalized g.

An adaptive MS tracker was created by combining the lip detector, the MS algorithm, and the Bhattacharyya coefficient for

similarity measure. Tests on AVICAR videos validated its ability to reacquire its target when normal MS fails. Comparisons of processing time also suggest that the adaptive algorithm can be used to real-time live tracking. The adaptive nature also allows the MS algorithm to converge on its target faster.

Future opportunities are available to examine more color components that were not included in this report. Furthermore, many other training algorithms exist beyond AdaBoost, such as support vector machine and multilayer perceptron neural networks. Classifiers using these methods may produce higher accuracy.

The Bayesian classifier was explored briefly in this report, but it has the potential to eliminate all false positives. Additional color components can be added to the lip and skin PDFs to improve the classifier accuracy.

There are also other tracking algorithms that may perform better than MS for lip tracking. Continually adaptive mean shift (CAMSHIFT) can automatically adjust tracking window size to account for subjects moving closer or further away. Multiple-hypothesis methods such as particle filters may also perform better under rapid movement.

# BIBLIOGRAPHY


[1] "AVICAR Project: Audio-Visual Speech Recognition in a Car," University of Illinois at Urbana-Champaign, [Online]. Available: http://www.isle.illinois.edu/sst/AVICAR/. [Accessed August 2016].

[2] R. Navarathna, P. Lucey, D. Dean, C. Fookes and S. Sridharan, "Lip Detection For Audio-Visual Speech Recognition In-Car Environment," in *10th International Conference on Information Science, Signal Processing and their Applications*, Brisbane, 2010.

[3] G. Bradski and A. Kaehler, Learning OpenCV Computer Vision with the OpenCV Library, Sebastopol, CA: O'Reilly Media, Inc., 2008.

[4] E. Maggio and A. Cavallaro, Video Tracking, Theory and Practice, Chichester: John Wiley & Sons Ltd, 2011.

[5] B. N. Husain, "FACE DETECTION AND LIP LOCALIZATION," California Polytechnic State University, San Luis Obispo, 2011.

[6] B. Crow, "Automated Location and Tracking of Facial Features in an Unconstrained Environment," California Polytechnic State University, San Luis Obispo, 2008.

[7] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* vol. 1, pp. 511-518, 2001.

[8] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," *2005 IEEE Computer Society Conference on*

*Computer Vision and Pattern Recognition,* vol. 1, pp. 886-893, 2005.

[9] T. Ojala, M. Pietikainen and T. Maenpaa, "Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 24, no. 7, pp. 971-987, July 2002.

[10] The Mathworks, Inc., "Face Detection and Tracking Using CAMShift," The MathWorks, Inc., 2016. [Online]. Available: https://www.mathworks.com/help/vision/examples/face-detection-and-tracking-using-camshift.html. [Accessed 26 10 2016].

[11] T. Gevers and A. W. Smeulders, "Color-Based Object Recognition," *Pattern Recognition,* no. 32, pp. 453-464, 1999.

[12] T. Gevers and A. W. M. Smeulders, "PicToSeek: Combining Color and Shape Invariant Features for Image Retrieval," *IEEE Transactions On Image Proessing,* vol. 9, no. 1, pp. 102-109, January 2000.

[13] D. Comaniciu and V. Ramesh, "Kernel-Based Object Tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 25, no. 5, pp. 564-577, May 2003.

[14] B. Lee, M. Hasegawa-Johnson, C. Goudeseune, S. Kamdar, S. Boryus, M. Liu and T. Huang, "AVICAR: Audio-Visual Speech Corpus in a Car Environment," in *International Conference on Spoken Language Processing*, Jeju Island, 2004.

[15] "Train a Cascade Object Detector," The MathWorks, Inc., 2016. [Online]. Available:

https://www.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html. [Accessed 01 10 2016].

[16] The MathWorks, Inc., "bboxOverlapRatio," The MathWorks, Inc., 2016. [Online]. Available: https://www.mathworks.com/help/vision/ref/bboxoverlapratio.html. [Accessed 17 10 2016].

# APPENDICES

## APPENDIX A: COLOR ANALYSIS MATLAB CODE

```
clearvars; close all; clc;

numFrames = 25;
numSubFrames = 4;

% Directories
dirFaces = './Faces without Lips/';
dirLips = './Lips/';

%% RGB
[faces_gray faces_R faces_G faces_B] = hist_RGB(dirFaces, numFrames,
numSubFrames, './countRGB_faces.mat');
[lips_gray lips_R lips_G lips_B] = hist_RGB(dirLips, numFrames,
numSubFrames, './countRGB_lips.mat');

bin_gray = 0:1:255;
bin_R = 0:1:255;
bin_G = 0:1:255;
bin_B = 0:1:255;

% Plot 1D histograms
figure(1);
    subplot(3,1,1);
        plotHist1(bin_R, lips_R, faces_R, 'R');
    subplot(3,1,2);
        plotHist1(bin_G, lips_G, faces_G, 'G');
    subplot(3,1,3);
        plotHist1(bin_B, lips_B, faces_B, 'B');

figure(2);
    plotHist1(bin_gray, lips_gray, faces_gray, 'gray');

%% xyY

[faces_xy faces_x faces_y] = hist_xy(dirFaces, numFrames, numSubFrames,
'./countxy_faces.mat');
[lips_xy lips_x lips_y] = hist_xy(dirLips, numFrames, numSubFrames,
'./countxy_lips.mat');

bin_x = (0:1:255) ./ 255;
bin_y = (0:1:255) ./ 255;

% Plot 1D histograms
figure(2);
    subplot(2,1,1);
        x = plotHist1(bin_x, lips_x, faces_x, 'x');
    subplot(2,1,2);
        y = plotHist1(bin_y, lips_y, faces_y, 'y');

%% uvL

[faces_uv faces_u faces_v] = hist_uv(dirFaces, numFrames, numSubFrames,
'./countuv_faces.mat');
```

```
[lips_uv lips_u lips_v] = hist_uv(dirLips, numFrames, numSubFrames,
'./countuv_lips.mat');

bin_u = (0:1:255) ./ 255;
bin_v = (0:1:255) ./ 255;

% Plot 1D histograms
figure(3);
    subplot(2,1,1);
        u = plotHist1(bin_u, lips_u, faces_u, 'u');
    subplot(2,1,2);
        v = plotHist1(bin_v, lips_v, faces_v, 'v');

%% HSV

[faces_HS faces_H faces_S] = hist_HS(dirFaces, numFrames, numSubFrames,
'./countHS_faces.mat');
[lips_HS lips_H lips_S] = hist_HS(dirLips, numFrames, numSubFrames,
'./countHS_lips.mat');

bin_H = (0:1:255) ./ 255;
bin_S = (0:1:255) ./ 255;

% Plot 1D histograms
figure(4);
    subplot(2,1,1);
        hue = plotHist1(bin_H, lips_H, faces_H, 'Hue');
    subplot(2,1,2);
        sat = plotHist1(bin_S, lips_S, faces_S, 'Saturation');

%% Normalized RGB

% Create histograms for nRGB faces, and lips
[faces_rg faces_r faces_g] = hist_nRGB(dirFaces, numFrames,
numSubFrames, './countrg_faces.mat');
[lips_rg lips_r lips_g] = hist_nRGB(dirLips, numFrames, numSubFrames,
'./countrg_lips.mat');

bin_r = (0:1:255) ./ 255;
bin_g = (0:1:255) ./ 255;

% Plot 1D histograms
figure(5);
    subplot(2,1,1);
        r = plotHist1(bin_r, lips_r, faces_r, 'r');
    subplot(2,1,2);
        g = plotHist1(bin_g, lips_g, faces_g, 'g');

%% L*a*b*

% Create histograms for Lab backgrounds, faces, and lips
[faces_ab faces_a faces_b] = hist_Lab(dirFaces, numFrames,
numSubFrames, './countab_faces.mat');
[lips_ab lips_a lips_b] = hist_Lab(dirLips, numFrames, numSubFrames,
'./countab_lips.mat');

bin_a = (0:1:255) ./ 255;
bin_b = (0:1:255) ./ 255;
```

```
% Plot 1D histograms
figure(6);
    subplot(2,1,1);
        a = plotHist1(bin_a, lips_a, faces_a, 'a*');
    subplot(2,1,2);
        b = plotHist1(bin_b, lips_b, faces_b, 'b*');

%% YCbCr

% Create histograms for YCbCr backgrounds, faces, and lips
[faces_CbCr faces_Cb faces_Cr] = hist_YCbCr(dirFaces, numFrames,
numSubFrames, './countCbCr_faces.mat');
[lips_CbCr lips_Cb lips_Cr] = hist_YCbCr(dirLips, numFrames,
numSubFrames, './countCbCr_lips.mat');

bin_Cb = (0:1:255) ./ 255;
bin_Cr = (0:1:255) ./ 255;

% Plot 1D histograms
figure(7);
    subplot(2,1,1);
        cb = plotHist1(bin_Cb, lips_Cb, faces_Cb, 'Cb');
    subplot(2,1,2);
        cr = plotHist1(bin_Cr, lips_Cr, faces_Cr, 'Cr');

%% c1c2c3

[faces_c1 faces_c2 faces_c3] = hist_c1c2c3(dirFaces, numFrames,
numSubFrames, './countc1c2c3_faces.mat');
[lips_c1 lips_c2 lips_c3] = hist_c1c2c3(dirLips, numFrames,
numSubFrames, './countc1c2c3_lips.mat');

bin_c1 = (0:1:255) ./ 255;
bin_c2 = (0:1:255) ./ 255;
bin_c3 = (0:1:255) ./ 255;

% Plot 1D histograms
figure(8);
    subplot(3,1,1);
        c1 = plotHist1(bin_c1, lips_c1, faces_c1, 'c1');
    subplot(3,1,2);
        c2 = plotHist1(bin_c2, lips_c2, faces_c2, 'c2');
    subplot(3,1,3);
        c3 = plotHist1(bin_c3, lips_c3, faces_c3, 'c3');

%% l1l2l3

[faces_l1 faces_l2 faces_l3] = hist_l1l2l3(dirFaces, numFrames,
numSubFrames, './countl1l2l3_faces.mat');
[lips_l1 lips_l2 lips_l3] = hist_l1l2l3(dirLips, numFrames,
numSubFrames, './countl1l2l3_lips.mat');

bin_l1 = linspace(0, 2/3, 256);
bin_l2 = linspace(0, 2/3, 256);
bin_l3 = linspace(0, 2/3, 256);

% Plot 1D histograms
```

```
figure(9);
    subplot(3,1,1);
        l1 = plotHist1(bin_l1, lips_l1, faces_l1, 'l1');
    subplot(3,1,2);
        l2 = plotHist1(bin_l2, lips_l2, faces_l2, 'l2');
    subplot(3,1,3);
        l3 = plotHist1(bin_l3, lips_l3, faces_l3, 'l3');


function [countTotal_gray,countTotal_R,countTotal_G,countTotal_B] =
hist_RGB(dir, numFrames, numSubFrames, saveFile)

    % Initialize matrices
    countTotal_gray = zeros(256,1);
    countTotal_R = zeros(256,1);
    countTotal_G = zeros(256,1);
    countTotal_B = zeros(256,1);

    % Black pixel count
    numBlackPixels = 0;

    for i=1:numFrames
        for j=1:numSubFrames

            % Read image file
            inFile = [dir int2str(i) '_' int2str(j) '.png'];
            im_RGB = imread(inFile);

            im_gray = rgb2gray(im_RGB);
            im_R = im_RGB(:,:,1);
            im_G = im_RGB(:,:,2);
            im_B = im_RGB(:,:,3);

            % Increment black pixel count
            % Black pixels counted by condition R + G + B == 0
            numBlackPixels = numBlackPixels + sum(sum(sum(im_RGB,3) ==
0));

            % Increment histograms
            [count_gray, ~] = imhist(im_gray);
            [count_R, ~] = imhist(im_R);
            [count_G, ~] = imhist(im_G);
            [count_B, ~] = imhist(im_B);

            countTotal_gray = countTotal_gray + count_gray;
            countTotal_R = countTotal_R + count_R;
            countTotal_G = countTotal_G + count_G;
            countTotal_B = countTotal_B + count_B;

        end
    end

    % Ignore black pixels
    countTotal_gray(1) = 0;
    countTotal_R(1) = countTotal_R(1) - numBlackPixels;
    countTotal_G(1) = countTotal_G(1) - numBlackPixels;
    countTotal_B(1) = countTotal_B(1) - numBlackPixels;
```

```
    % Save histograms to file
    save(saveFile,'countTotal_gray','countTotal_R','countTotal_G','coun
tTotal_B');

end


function [countTotal_xy countTotal_x countTotal_y] = hist_xy(dir,
numFrames, numSubFrames, saveFile)

    countTotal_xy = zeros(256, 256);

    for i=1:numFrames
        for j=1:numSubFrames

            % Read image file
            inFile = [dir int2str(i) '_' int2str(j) '.png'];
            im_RGB = imread(inFile);

            % Convert to Lab
            im_XYZ = xyz2double(applycform(im_RGB,
makecform('srgb2xyz')));
            im_xyY = applycform(im_XYZ, makecform('xyz2xyl'));
            im_x = im_xyY(:,:,1);
            im_y = im_xyY(:,:,2);
            im_Y = im_xyY(:,:,3);

            % 2D Histogram
            [numRows,numCols] = size(im_x);
            for m=1:numRows
                for n=1:numCols
                    x = round(255*im_x(m,n) + 1);
                    y = round(255*im_y(m,n) + 1);
                    % Ignore black pixels
                    if (im_Y(m,n) ~= 0)
                        % Increment histogram
                        countTotal_xy(x,y) = countTotal_xy(x,y) + 1;
                    end
                end
            end
        end
    end

    % Calculate individual counts for x and y
    countTotal_x = sum(countTotal_xy,2);
    countTotal_y = sum(countTotal_xy).';

    % Save histograms to file
    save(saveFile, 'countTotal_xy', 'countTotal_x', 'countTotal_y');

end


function [countTotal_uv countTotal_u countTotal_v] = hist_uv(dir,
numFrames, numSubFrames, saveFile)

    countTotal_uv = zeros(256,256);
```

```
    for i=1:numFrames
        for j=1:numSubFrames

            % Read image file
            inFile = [dir int2str(i) '_' int2str(j) '.png'];
            im_RGB = imread(inFile);

            % Convert to Lab
            im_XYZ = xyz2double(applycform(im_RGB,
makecform('srgb2xyz')));
            im_uvL = applycform(im_XYZ, makecform('xyz2uvl'));
            im_u = im_uvL(:,:,1);
            im_v = im_uvL(:,:,2);
            im_L = im_uvL(:,:,3);

            % 2D Histogram
            [numRows,numCols] = size(im_u);
            for m=1:numRows
                for n=1:numCols
                    x = round(256*im_u(m,n) + 1);
                    y = round(256*im_v(m,n) + 1);
                    % Ignore black pixels
                    if (im_L(m,n) ~= 0)
                        % Increment histogram
                        countTotal_uv(x,y) = countTotal_uv(x,y) + 1;
                    end
                end
            end
        end
    end

    % Calculate individual counts for a and b
    countTotal_u = sum(countTotal_uv,2);
    countTotal_v = sum(countTotal_uv).';

    % Save histograms to file
    save(saveFile, 'countTotal_uv', 'countTotal_u', 'countTotal_v');

end


function [countTotalHS countTotalH countTotalS] = hist_HS(dir,
numFrames, numSubFrames, saveFile)

    countTotalHS = zeros(256,256);

    for i=1:numFrames
        for j=1:numSubFrames

            % Read image file
            inFile = [dir int2str(i) '_' int2str(j) '.png'];
            im_RGB = imread(inFile);

            % Convert to HSV
            im_HSV = rgb2hsv(im_RGB);
            im_HSV = uint8(255*im_HSV)+1;
            im_H = im_HSV(:,:,1);
            im_S = im_HSV(:,:,2);
```

```
            im_V = im_HSV(:,:,3);

            % 2D Histogram
            [numRows,numCols] = size(im_H);
            for m=1:numRows
                for n=1:numCols
%                     hue = round(255*im_H(m,n) + 1);
%                     sat = round(255*im_S(m,n) + 1);
                    hue = im_H(m,n);
                    sat = im_S(m,n);
                    % Ignore black pixels
                    if (im_V(m,n) ~= 1)
                        countTotalHS(hue,sat) = countTotalHS(hue,sat) +
1;
                    end
                end
            end

        end
    end

    % Shift hue by 0.2 (52 bins)
%     countTotalHS = circshift(countTotalHS,52);

    % Calculate individual counts for hue and saturation
    countTotalH = sum(countTotalHS,2);
    countTotalS = sum(countTotalHS).';

    % Save histograms to file
    save(saveFile, 'countTotalHS', 'countTotalH', 'countTotalS');

end


function [countTotalrg countTotalr countTotalg] = hist_nRGB(dir,
numFrames, numSubFrames, saveFile)

    countTotalrg = zeros(256,256);

    for i=1:numFrames
        for j=1:numSubFrames

            % Generate image name and read image file
            inFile = [dir int2str(i) '_' int2str(j) '.png'];
            im_RGB = imread(inFile);

            % Convert to nRGB
            im_rgb = RGB_to_nRGB(im_RGB);
            im_r = im_rgb(:,:,1);
            im_g = im_rgb(:,:,2);
            im_b = im_rgb(:,:,3);

            % 2D Histogram
            [numRows,numCols] = size(im_r);
            for m=1:numRows
                for n=1:numCols
                    % Change range to [1,256]
                    r = round(255*im_r(m,n) + 1);
```

```
                    g = round(255*im_g(m,n) + 1);
                    b = round(255*im_b(m,n) + 1);
                    % Ignore black pixels
                    if (~(im_r(m,n) == 0 && im_g(m,n) == 0 && im_b(m,n)
== 0))
                        % Increment histogram
                        countTotalrg(r,g) = countTotalrg(r,g) + 1;
                    end
                end
            end
        end
    end

    % Calculate individual counts for r and g
    countTotalr = sum(countTotalrg,2);
    countTotalg = sum(countTotalrg).';

    % Save histograms to file
    save(saveFile, 'countTotalrg', 'countTotalr', 'countTotalg');

end


function [countTotal_ab countTotal_a countTotal_b] = hist_Lab(dir,
numFrames, numSubFrames, saveFile)

    countTotal_ab = zeros(256,256);

    for i=1:numFrames
        for j=1:numSubFrames

            % Read image file
            inFile = [dir int2str(i) '_' int2str(j) '.png'];
            im_RGB = imread(inFile);

            % Convert to Lab
            im_Lab = applycform(im_RGB, makecform('srgb2lab'));
            im_L = im_Lab(:,:,1);
            im_a = im_Lab(:,:,2);
            im_b = im_Lab(:,:,3);

            % 2D Histogram
            [numRows,numCols] = size(im_a);
            for m=1:numRows
                for n=1:numCols
                    a = im_a(m,n);
                    b = im_b(m,n);
                    % Ignore black pixels
                    if (im_L(m,n) ~= 0)
                        % Increment histogram
                        countTotal_ab(a,b) = countTotal_ab(a,b) + 1;
                    end
                end
            end
        end
    end

    % Calculate individual counts for a and b
```

```
    countTotal_a = sum(countTotal_ab,2);
    countTotal_b = sum(countTotal_ab).';

    % Save histograms to file
    save(saveFile, 'countTotal_ab', 'countTotal_a', 'countTotal_b');

end


function [countTotal_c1, countTotal_c2, countTotal_c3] ...
    = hist_c1c2c3(dir, numFrames, numSubFrames, saveFile)

    % Initialize matrices
    countTotal_c1 = zeros(256,1);
    countTotal_c2 = zeros(256,1);
    countTotal_c3 = zeros(256,1);
    countTotal_c1c2 = zeros(256,256);


    for i=1:numFrames
        for j=1:numSubFrames

            % Read image file
            inFile = [dir int2str(i) '_' int2str(j) '.png'];
            im_RGB = imread(inFile);

            % Convert to c1c2c3
            im_c1c2c3 = RGB_to_c1c2c3(im_RGB);
            im_c1 = im_c1c2c3(:,:,1);
            im_c2 = im_c1c2c3(:,:,2);
            im_c3 = im_c1c2c3(:,:,3);

            % Increment histograms
            [count_c1,~] = imhist(im_c1);
            [count_c2,~] = imhist(im_c2);
            [count_c3,~] = imhist(im_c3);
            [count_c1c2,~,~] = histcounts2(im_c1,im_c2,[256 256], ...
                'BinMethod','integers', ...
                'XBinLimits',[-0.5 255.5], ...
                'YBinLimits',[-0.5 255.5]);

            countTotal_c1 = countTotal_c1 + count_c1;
            countTotal_c2 = countTotal_c2 + count_c2;
            countTotal_c3 = countTotal_c3 + count_c3;
            countTotal_c1c2 = countTotal_c1c2 + count_c1c2;

        end
    end

    % Ignore black pixels
    countTotal_c1(1) = 0;
    countTotal_c2(1) = 0;
    countTotal_c3(1) = 0;
    countTotal_c1c2(1,1) = 0;

    % Save histograms to file
    save(saveFile, 'countTotal_c1', 'countTotal_c2', 'countTotal_c3',
'countTotal_c1c2');
```

```
end


function stats = plotHist1(x, y1, y2, color)

% Plot 1D Histograms
%
% Inputs
%   x = histogram bins
%   y1 = lips counts
%   y2 = face counts
%   y3 = background counts
%   color = string name of color space

    % Normalize data
    y1 = y1 ./ sum(y1);
    y2 = y2 ./ sum(y2);

    % Calculate means
    mean1 = sum(x * y1);
    mean2 = sum(x * y2);

    % Calculate standard deviations
    sd1 = sqrt(sum((x - mean1).^2 * y1));
    sd2 = sqrt(sum((x - mean2).^2 * y2));

    % Calculate axis limits
    xmin = min(x);
    xmax = max(x);
    ymin = min([y1;y2]);
    ymax = max([y1;y2]);

    stats = [mean1, sd1, mean2, sd2];

    % Stem plot
    stem(x, [y1 y2]);
    axis([xmin xmax ymin ymax]);
    title([color '  Histogram']); xlabel(color); ylabel('Counts');
    legend('Lips','Faces');
    txstr(1) = {['lip \mu=',num2str(mean1)]};
    txstr(2) = {['lip \sigma =',num2str(sd1)]};
    txstr(3) = {['face \mu=',num2str(mean2)]};
    txstr(4) = {['face \sigma =',num2str(sd2)]};
    text(0.02*max(x), 0.75*max([y1;y2]), txstr);
    grid on;

end
```

## APPENDIX B: LIP DETECTION MATLAB CODE

```
% trainLipDetect.m

% maximum number of stages to use for each feature type
hogStages = 20;
lbpStages = 20;
haarStages = 20;

%% Train Lip Detector RGB

% folder = 'RGB';
% disp(folder);
%
% % load lip positive samples
% % created using Training Image Labeler app
% name = 'ROI_RGB.mat';
% file = fullfile(folder, name);
% load(file);
%
% % set folder path containing negative samples
% negFolder = fullfile(folder, 'Negative Images');
%
% % train object detectors
% % max 12 stages
% trainCascadeObjectDetector('lipDetector_RGB_HOG.xml', ROI,
negFolder, ...
%     'FeatureType', 'HOG', 'NumCascadeStages', hogStages,
'TruePositiveRate', 0.995, ...
%     'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');
%
% % max 12 stages
% trainCascadeObjectDetector('lipDetector_RGB_LBP.xml', ROI,
negFolder, ...
%     'FeatureType', 'LBP', 'NumCascadeStages', lbpStages,
'TruePositiveRate', 0.995, ...
%     'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');
%
% % max 10 stages
% trainCascadeObjectDetector('lipDetector_RGB_Haar.xml', ROI,
negFolder, ...
%     'FeatureType', 'Haar', 'NumCascadeStages', haarStages,
'TruePositiveRate', 0.995, ...
%     'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

%% Train Lip Detector grey

folder = 'gray';
disp(folder);

% load lip positive samples
% created using Training Image Labeler app
name = 'ROI_gray.mat';
file = fullfile(folder, name);
load(file);
```

```
% set folder path containing negative samples
negFolder = fullfile(folder, 'Negative Images');

% train viola-jones object detector
% max 12 stages
trainCascadeObjectDetector('lipDetector_gray_HOG.xml', ROI,
negFolder, ...
    'FeatureType', 'HOG', 'NumCascadeStages', hogStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

% max 12 stages
trainCascadeObjectDetector('lipDetector_gray_LBP.xml', ROI,
negFolder, ...
    'FeatureType', 'LBP', 'NumCascadeStages', lbpStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

% max 11 stages
trainCascadeObjectDetector('lipDetector_gray_Haar.xml', ROI,
negFolder, ...
    'FeatureType', 'Haar', 'NumCascadeStages', haarStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');


%% Train Lip Detector a

folder = 'a';
disp(folder);

% load lip positive samples
% created using Training Image Labeler app
name = 'ROI_a.mat';
file = fullfile(folder, name);
load(file);

% set folder path containing negative samples
negFolder = fullfile(folder, 'Negative Images');

% train viola-jones object detectors
% max 17 stages
trainCascadeObjectDetector('lipDetector_a_HOG.xml', ROI, negFolder, ...
    'FeatureType', 'HOG', 'NumCascadeStages', hogStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

% max 19 stages
trainCascadeObjectDetector('lipDetector_a_LBP.xml', ROI, negFolder, ...
    'FeatureType', 'LBP', 'NumCascadeStages', lbpStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');
```

```
% max 13 stages
trainCascadeObjectDetector('lipDetector_a_Haar.xml', ROI,
negFolder, ...
    'FeatureType', 'Haar', 'NumCascadeStages', haarStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

%% Train Lip Detector C1

folder = 'c1';
disp(folder);

% load lip positive samples
% created using Training Image Labeler app
name = 'ROI_c1.mat';
file = fullfile(folder, name);
load(file);

% set folder path containing negative samples
negFolder = fullfile(folder, 'Negative Images');

% train viola-jones object detectors
% max 12 stages
trainCascadeObjectDetector('lipDetector_c1_HOG.xml', ROI,
negFolder, ...
    'FeatureType', 'HOG', 'NumCascadeStages', hogStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

% max 20 stages
trainCascadeObjectDetector('lipDetector_c1_LBP.xml', ROI,
negFolder, ...
    'FeatureType', 'LBP', 'NumCascadeStages', lbpStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

% max 15 stages
trainCascadeObjectDetector('lipDetector_c1_Haar.xml', ROI,
negFolder, ...
    'FeatureType', 'Haar', 'NumCascadeStages', haarStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

%% Train Lip Detector C2

folder = 'c2';
disp(folder);

% load lip positive samples
% created using Training Image Labeler app
name = 'ROI_c2.mat';
file = fullfile(folder, name);
load(file);
```

```
% set folder path containing negative samples
negFolder = fullfile(folder, 'Negative Images');

% train viola-jones object detector
% max 19 stages
trainCascadeObjectDetector('lipDetector_c2_HOG.xml', ROI,
negFolder, ...
    'FeatureType', 'HOG', 'NumCascadeStages', hogStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

% max 18 stages
trainCascadeObjectDetector('lipDetector_c2_LBP.xml', ROI,
negFolder, ...
    'FeatureType', 'LBP', 'NumCascadeStages', lbpStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

% max 18 stages
trainCascadeObjectDetector('lipDetector_c2_Haar.xml', ROI,
negFolder, ...
    'FeatureType', 'Haar', 'NumCascadeStages', haarStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

%% Train Lip Detector g

folder = 'g';
disp(folder);

% load lip positive samples
% created using Training Image Labeler app
name = 'ROI_g.mat';
file = fullfile(folder, name);
load(file);

% set folder path containing negative samples
negFolder = fullfile(folder, 'Negative Images');

% train viola-jones object detector
% max 14 stages
trainCascadeObjectDetector('lipDetector_g_HOG.xml', ROI, negFolder, ...
    'FeatureType', 'HOG', 'NumCascadeStages', hogStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');

% max 16 stages
trainCascadeObjectDetector('lipDetector_g_LBP.xml', ROI, negFolder, ...
    'FeatureType', 'LBP', 'NumCascadeStages', lbpStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');
```

```
% max 13 stages
trainCascadeObjectDetector('lipDetector_g_Haar.xml', ROI,
negFolder, ...
    'FeatureType', 'Haar', 'NumCascadeStages', haarStages,
'TruePositiveRate', 0.995, ...
    'FalseAlarmRate', 0.5, 'NegativeSamplesFactor', 2,
'ObjectTrainingSize', 'Auto');


% detectLipsScript.m

%% Load c1c2 PDFs

% load c1c2c3 lip and face histograms
lipCounts = load('../Color Analysis/countc1c2c3_lips.mat');
faceCounts = load('../Color Analysis/countc1c2c3_faces.mat');

% normalize lip histogram to form lip PDF
lipPDF = lipCounts.countTotal_c1c2;
lipPDF = imgaussfilt(lipPDF,0.5);
lipPDF = lipPDF ./ sum(sum(lipPDF));

% normalize face histogram to form face PDF
facePDF = faceCounts.countTotal_c1c2;
facePDF = imgaussfilt(facePDF,0.5);
facePDF = facePDF ./ sum(sum(facePDF));

% threshold
c1c2Thresh = 0.8739 / 0.1261;
% c1c2Thresh = 1.5;

%% Loop through all colors and features

for colorCell = {'gray'}%{'gray','a','c1','c2','g','mouth'}

% convert color cell to string
color = cell2mat(colorCell);

% display current color
disp(color);

if strcmp(color,'mouth')
    featList = {'Haar'};
else
    featList = {'HOG','Haar','LBP'};
end

for featCell = featList

% convert feature cell to string
feature = cell2mat(featCell);

% display current feature
disp(['    ' feature]);

% location of test images
testDir = fullfile(color, 'Test Images');
testDirRGB = fullfile('RGB', 'Test Images');
```

```
% name of trained lip detector XML file
lipDetectorXML = ['lipDetector_' color '_' feature '.xml'];

% location to write bounding box images
testResultsDir = fullfile(color, 'Test Results', feature);

%% Create Detector Objects

% create lip detector object from XML file
if strcmp(color,'mouth')
    lipDetector = vision.CascadeObjectDetector('Mouth');
else
    lipDetector = vision.CascadeObjectDetector(lipDetectorXML);
end
lipDetector.UseROI = true;
lipDetector.MergeThreshold = 3; % default 4

% OLD CODE
% create face detector object
% faceDetector = vision.CascadeObjectDetector('FrontalFaceCART');

%% Run Lip Detector on Test Images

% load true locations of lips in test images
load('truePositives.mat');

% load locations of faces in test images
load('faces.mat');

% create test results struct
testResults = struct('file',[],'bbox',[],'numLips',[],...
    'overlapRatioUnion',[],'overlapRatioMin',[]);

% start timer
tic;

% run lip detector on test images
for i = 1:50
for j = 1:4

    % calculate index
    index = (i-1)*4 + j;

    % read test image for current color space
    inFile = fullfile(testDir, [int2str(i) '_' int2str(j) '.png']);
    image = imread(inFile);

    % if using mouth detector, convert rgb image to grayscale
    if strcmp(color,'mouth')
        image = rgb2gray(image);
    end

    % histogram equalization
    image = histeq(image,256);

    % enhance local contrast of test image
%     image = localcontrast(image);
```

```
    % apply gamma equalization to test image
%     image = imadjust(image);

    % sharpen test image
    image = imsharpen(image);

    % read RGB image for annotations
    inFileRGB = fullfile(testDirRGB,[int2str(i) '_' int2str(j)
'.png']);
    imageRGB = imread(inFileRGB);

    % OLD CODE
%     run face detector on image
%     bboxFace = step(faceDetector,imageRGB);

    % read face bbox coordinates and draw face bbox
    bboxFace = faces(index).bbox;
    annotatedFrame = insertObjectAnnotation(imageRGB, ...
                                            'rectangle', ...
                                            bboxFace, ...
                                            'face');

    % trim face bounding box to bottom half of face
    % [x y width height]
    bboxFaceBottom = bboxFace;
    bboxFaceBottom(:,2) = bboxFace(:,2) + floor(bboxFace(:,4)/ 2);
    bboxFaceBottom(:,4) = ceil(bboxFace(:,4) / 2);
    annotatedFrame = insertObjectAnnotation(annotatedFrame, ...
                                            'rectangle', ...
                                            bboxFaceBottom, ...
                                            'face bottom');

    % detect lips in face bottom and draw lip bbox
    bboxLip = step(lipDetector,image,bboxFaceBottom);

    % if lips are found, calculate probability of true detection
    if (~isempty(bboxLip))

        % number of detections
        numDet = size(bboxLip,1);

        % initialize lip detections
        bboxLipTemp = zeros(numDet,5);
        bboxLipTemp(:,1:4) = bboxLip;

        % loop through all detections
        for k = 1:numDet

            bbox = bboxLip(k,:);
            width = bbox(3);
            height = bbox(4);
            boxArea = height * width;

            % crop image to isolate detected area
            crop = imcrop(imageRGB,bbox);
            crop = RGB_to_c1c2c3(crop);
            crop(:,:,1) = imgaussfilt(crop(:,:,1),3);
```

```
            crop(:,:,2) = imgaussfilt(crop(:,:,2),3);

            % initialize lip and face probability map
            pLip = zeros(height,width);
            pFace = zeros(height,width);

            % calculate probability of lip or face of each pixel
            for row = 1:height
                for col = 1:width

                    % c1 and c2 value of pixel
                    c1 = crop(row,col,1) + 1;
                    c2 = crop(row,col,2) + 1;

                    % populate lip and probability map
                    pLip(row,col) = lipPDF(c1,c2);
                    pFace(row,col) = facePDF(c1,c2);

                end
            end

            %
            pLip = 0.13 * pLip;
            pFace = (1 - 0.13) * pFace;

            % classify lip pixels
            lipPixels = pLip >= pFace;

            % calculate number of lip pixels
            lipArea = bwarea(lipPixels);

            % assign percentage of lip pixels in box area
%             bboxLipTemp(k,5) = lipArea ./ boxArea;
            bboxLipTemp(k,5) = lipArea;

        end

        % sort detections by percentage of lip pixels
        bboxLipTemp = flipud(sortrows(bboxLipTemp,5));

        % save detection with highest percentage
        bboxLip = bboxLipTemp(1,1:4);

        % only save if area is above 1%
%         if (bboxLipTemp(1,5) > 0)
%             bboxLip = bboxLipTemp(1,1:4);
%         else
%             bboxLip = int16.empty(0,4);
%         end

    end

    % if lips are found choose the largest one
%     if (~isempty(bboxLip))
%
%         % sort lips by size largest to smallest
%         bboxLip = flipud(sortrows(bboxLip,3));
%
```

```
%         % choose largest detected lip
%         bboxLip = bboxLip(1,:);
%
%     end

    % mark image with lip bounding boxes
    annotatedFrame = insertObjectAnnotation(annotatedFrame, ...
        'rectangle', ...
        bboxLip, ...
        'lip');

    % mark true lip location
    trueBBox = truePositives(index).objectBoundingBoxes;
    annotatedFrame = insertShape(annotatedFrame, ...
                                 'FilledRectangle', ...
                                 trueBBox, ...
                                 'Color', ...
                                 'green');

    % write resulting image to file
    outFile = fullfile(testResultsDir, [int2str(i) '_' int2str(j)
'.png']);
    imwrite(annotatedFrame, outFile, 'png');

    %% save detection results

    % file name
    testResults(index).file = outFile;

    % lip bbox coordinates
    testResults(index).bbox = bboxLip;

    % number of lips detected
    testResults(index).numLips = size(bboxLip,1);

    % union overlap ratio
    testResults(index).overlapRatioUnion ...
        = bboxOverlapRatio(trueBBox,bboxLip,'Union');

    % min overlap ratio
    testResults(index).overlapRatioMin ...
        = bboxOverlapRatio(trueBBox,bboxLip,'Min');

end
end

% stop timer
toc;

% save test results
outFile = fullfile(color,['testResults_' color '_' feature '.mat']);
save(outFile,'testResults');

end
end


% detectLipsTestResults.m
```

```
% color components
% gray, a, c1, c2, g, or mouth
color = 'gray';
disp(['color component = ' color]);

% number of test images
numTestImages = 200;

% overlap ratio threshold for true positive
thresh = 0.3;

% list of features tested
if strcmp(color,'mouth')
    featCell = {'Haar'};
else
    featCell = {'HOG','Haar','LBP'};
end
numFeats = length(featCell);

% average of overlap ratios for each feature
overlapRatioUnionAvg = zeros(1,numFeats);
overlapRatioMinAvg = zeros(1,numFeats);

% total number of lips detected for each feature
numLipsTotal = zeros(1,numFeats);

% true positives for each test image/feature
truePos = zeros(numTestImages,numFeats);

% false positives for each test image/feature
falsePos = zeros(numTestImages,numFeats);

% cycle through all features
for featIndex = 1:numFeats

    % convert color and feat to strings
    feature = cell2mat(featCell(featIndex));

    % name and path of saved test results file
    testResultsName = ['testResults_' color '_' feature '.mat'];
    testResultsPath = fullfile(color,testResultsName);

    % load test results for current color and feature
    load(testResultsPath);

    % extract number of lips detected and sum total
    numLips = cell2mat({testResults.numLips});
    numLipsTotal(featIndex) = sum(numLips);

    % cycle through all test images
    for testIndex = 1:numTestImages

        % read union overlap ratio(s)
        union = testResults(testIndex).overlapRatioUnion;

        % read min overlap ratio(s)
        min = testResults(testIndex).overlapRatioMin;
```

```
        % count true/false positives
        truePos(testIndex,featIndex) = sum(union .* min >= thresh);
        falsePos(testIndex,featIndex) = sum(union .* min < thresh);

    end

end

% divide ratio sums by number of lips detected to caculate average
overlapRatioUnionAvg = overlapRatioUnionAvg ./ numLipsTotal;
overlapRatioMinAvg = overlapRatioMinAvg ./ numLipsTotal;

% count number of true postives and false positives
truePosTotal = sum(truePos);
falsePosTotal = sum(falsePos);

% count number of true negatives and false negatives
trueNegTotal = sum(falsePos <= 0);
falseNegTotal = sum(truePos <= 0);

% calculate miss rate
% missRate = falseNegTotal ./ (falseNegTotal + truePosTotal);

%% convert matrices to tables

% total number of lips detected for each feature
numLipsTotal = array2table(numLipsTotal, ...
                           'RowNames', {'Lips Detected'}, ...
                           'VariableNames', featCell);

% total number of true positives using the union method
truePosTotal = array2table(truePosTotal, ...
                           'RowNames', {'True Positives'}, ...
                           'VariableNames', featCell);

% total number of false positives using the union method
falsePosTotal = array2table(falsePosTotal, ...
                            'RowNames', {'False Positives'}, ...
                            'VariableNames', featCell);

% total number of false negatives (no lips detected in test image)
trueNegTotal = array2table(trueNegTotal, ...
                           'RowNames', {'True Negatives'}, ...
                           'VariableNames', featCell);

% total number of false negatives (no lips detected in test image)
falseNegTotal = array2table(falseNegTotal, ...
                            'RowNames', {'False Negatives'}, ...
                            'VariableNames', featCell);

% combine data into a single results table
results = [numLipsTotal;
           truePosTotal;
           falsePosTotal;
           trueNegTotal;
           falseNegTotal];
```

```
% display results table
disp(results);
```

## APPENDIX C: LIP TRACKING MATLAB CODE

```
% meanShiftTrackerLips.m

%% Global Varibales

% trained lip detector XML file
lipDetectorXML = 'lipDetector_gray_HOG.xml';

thresh = 2;          % threshold for mean-shift
maxIteration = 10;  % maximum number of iterations (non-convergent
cases)

%% Create VideoWriter and VideoReader objects

% create video with bounding box around object
boundingBoxVideo = VideoWriter('boundingBox4.avi');
open(boundingBoxVideo);

% read video
video = VideoReader('mov4.avi');
videoH = video.Height;
videoW = video.Width;
videoNumFrames = int64(video.Duration .* video.FrameRate);

%% Find first candidate

% create face detector objects
faceDetector = vision.CascadeObjectDetector();
faceDetector.MergeThreshold = 4;

% create lip detector object
lipDetector = vision.CascadeObjectDetector(lipDetectorXML);
lipDetector.UseROI = true;  % use ROI feature
lipDetector.MergeThreshold = 8; % default 4

% create bounding box variables
bboxFrame = [];
bboxFaceBottom = [];
bboxLip = [];
bboxSize = 0;

% initialize counters
frameNum = 0;    % current frame number

% find first frame with lips
while (frameNum < 500 && isempty(bboxLip))
    frameRGB = readFrame(video);
    frameNum = frameNum + 1;

    % detect lips in current frame
    [bboxLip,bboxFrame] =
detectLips(faceDetector,lipDetector,frameRGB);

    % write frame to video
    writeVideo(boundingBoxVideo,bboxFrame);

end
```

```
% show first frame with lips
figure(1);
imshow(bboxFrame);
title(['Detected Lip(s) in frame ' num2str(frameNum)]);

%% Define model coordinates

% model bounding box [x y width height]
modelBbox = bboxLip;
modelX = modelBbox(1);
modelY = modelBbox(2);
modelW = modelBbox(3);
modelH = modelBbox(4);
modelCenter = [modelX modelY] + round([modelW modelH] / 2);

% initialize candidate bounding box [x y width height]
candBbox = modelBbox;
candX = candBbox(1);
candY = candBbox(2);
candW = candBbox(3);
candH = candBbox(4);

% set candidate local center coordinates
candLocalCenter = round([candW candH] / 2);

% max and min coordinates before candidate is out of frame
maxCandX = videoW - candW;
maxCandY = videoH - candH;
minCandX = 1;
minCandY = 1;

% generate distance matrix used as LUT
distanceMatrix = createDistanceMatrix(modelH,modelW);

% create model PDF
model = imcrop(frameRGB, modelBbox);
modelPDF = createPDF(model);

% candidate PDF starts off the same as model PDF
candPDF = modelPDF;

% shift vector saves shift value for each frame and iteration
shiftVector = zeros(maxIteration,videoNumFrames);

% Bhattacharyya coefficient for each frame
bhatt = zeros(videoNumFrames,1);

%% loop through each video frame

% start timer
tic;

while hasFrame(video)

    % read current frame
    frameRGB = readFrame(video);     % read frame
    frameNum = frameNum + 1;         % increment frame number
```

```
    % initialize iteration count
    iteration = 0;
    shift = thresh + 1;

    % perform mean-shift until threshold is met or max iterations
reached
    while (shift > thresh) && (iteration < maxIteration)

        % increment interation count
        iteration = iteration + 1;

        % crop frame to isolate candidate
        cand = imcrop(frameRGB, candBbox);

        % generated candidate PDF
        candPDF = createPDF(cand);

        % generate weight function from model and candidate PDFs
        weightFunction = createWeight(modelPDF,candPDF);

        % convert candidate c1c2c3 values to integers
        candC1C2C3 = RGB_to_c1c2c3(cand) + 1;
        candC1 = candC1C2C3(:,:,1);
        candC2 = candC1C2C3(:,:,2);

        % numerator and denominator of mean-shift vector
        numerator = [0,0];
        denominator = 0;

        % calculate mean-shift vector numerator and denominator
        for i=1:modelH
            for j=1:modelW

                weight = weightFunction(candC1(i,j), candC2(i,j));
                distance = [distanceMatrix(i,j,1),
distanceMatrix(i,j,2)];

                numerator = numerator + (weight .* distance);
                denominator = denominator + weight;

            end
        end

        % calculate mean shift vector
        if denominator == 0
            meanShiftVector = [0,0];
            disp('undefined mean-shift vector');
        else
            meanShiftVector = numerator ./ denominator;
            meanShiftVector = round(meanShiftVector .*
candLocalCenter);
        end

        % calculate shift and update shift vector
        shift = sqrt(meanShiftVector * meanShiftVector.');
        shiftVector(iteration,frameNum) = shift;
```

```
        % determine new location from mean-shift vector
        candX = candX + meanShiftVector(2);
        candX = max(minCandX, candX);
        candX = min(maxCandX, candX);
        candY = candY + meanShiftVector(1);
        candY = max(minCandY, candY);
        candY = min(maxCandY, candY);
        candBbox = [candX candY candW candH];

    end

    % calculate Bhattacharyya coefficient and update vector
    bhatt(frameNum) = bhattCoeff(modelPDF,candPDF);

    % Print current time and total number of iterations fo current
frame
    % Only display for integer times
    if ~mod(video.CurrentTime, 1)
        display = [num2str(video.CurrentTime), 's, frame ',
num2str(frameNum)];
        disp(display);
        display = ['    iterations = ', num2str(iteration)];
        disp(display);
    end

    % add bounding box to frame
    frameRGB =
insertObjectAnnotation(frameRGB,'rectangle',candBbox,'lip');

    % add frame number to top left
    frameRGB = insertText(frameRGB,[1 1],frameNum);

    % write frame with bounding box to boudingBox.avi
    writeVideo(boundingBoxVideo,frameRGB);

end

% stop timer
toc;

%% Plot Bhattacharyya coefficient of each frame
figure(2);
plot(bhatt);
ylim([0 1]);
ylabel('Bhattacharyya Coefficient'); xlabel('Frame');
grid on;

%% Plot number of iterations
numShifts = sum(shiftVector > 0);
figure(3);
stem(numShifts);
ylabel('Number of Shifts'); xlabel('Frame');
grid on;


%% close video objects
close(boundingBoxVideo);
```

```
% meanShiftTrackerLipsAdapt.m

%% Global Varibales

% trained lip detector XML file
lipDetectorXML = 'lipDetector_gray_HOG.xml';

thresh = 2;         % threshold for mean-shift
maxIteration = 10;  % maximum number of iterations (non-convergent
cases)
bhattThresh = 0.85; % threshold for bhatt coefficient

%% Create VideoWriter and VideoReader objects

% create video with bounding box around object
boundingBoxVideo = VideoWriter('boundingBox4 adapt.avi');
open(boundingBoxVideo);

% read video
video = VideoReader('mov4.avi');
videoH = video.Height;
videoW = video.Width;
videoNumFrames = int64(video.Duration .* video.FrameRate);

%% Find first candidate

% create face detector objects
faceDetector = vision.CascadeObjectDetector();
faceDetector.MergeThreshold = 4;

% create lip detector object
lipDetector = vision.CascadeObjectDetector(lipDetectorXML);
lipDetector.UseROI = true;  % use ROI feature
lipDetector.MergeThreshold = 8; % default 4

% create bounding box variables
bboxFrame = [];
bboxFaceBottom = [];
bboxLip = [];
bboxSize = 0;

% initialize current frame number count
frameNum = 0;

% find first frame with lips
while (frameNum < 500 && isempty(bboxLip))

    % read frame from videp
    frameRGB = readFrame(video);

    % increment frame number
    frameNum = frameNum + 1;

    % detect lips in current frame
    [bboxLip,bboxFrame] =
detectLips(faceDetector,lipDetector,frameRGB);

    % write frame to video
```

```
    writeVideo(boundingBoxVideo,bboxFrame);

end

% show first frame with lips
figure(1);
imshow(bboxFrame);
title(['Detected Lip(s) in frame ' num2str(frameNum)]);

%% Define model coordinates

% model bounding box [x y width height]
modelBbox = bboxLip;
modelX = modelBbox(1);
modelY = modelBbox(2);
modelW = modelBbox(3);
modelH = modelBbox(4);
modelCenter = [modelX modelY] + round([modelW modelH] / 2);

% initialize candidate bounding box [x y width height]
candBbox = modelBbox;
candX = candBbox(1);
candY = candBbox(2);
candW = candBbox(3);
candH = candBbox(4);

% set candidate local center coordinates
candLocalCenter = round([candW candH] / 2);

% max and min coordinates before candidate is out of frame
maxCandX = videoW - candW;
maxCandY = videoH - candH;
minCandX = 1;
minCandY = 1;

% generate distance matrix used as LUT
distanceMatrix = createDistanceMatrix(modelH,modelW);

% create model PDF
model = imcrop(frameRGB, modelBbox);
modelPDF = createPDF(model);

% candidate PDF starts off the same as model PDF
candPDF = modelPDF;

% shift vector saves shift value for each frame and iteration
shiftVector = zeros(maxIteration,videoNumFrames);

% Bhattacharyya coefficient for each frame
bhatt = zeros(videoNumFrames,1);

%% loop through each video frame

% start timer
tic;

while hasFrame(video)
```

```
    % read current frame
    frameRGB = readFrame(video);     % read frame
    frameNum = frameNum + 1;         % increment frame number

    % initialize iteration count
    iteration = 0;
    shift = thresh + 1;

    % calculate Bhattacharyya coefficient and update vector
    bhatt(frameNum) = bhattCoeff(modelPDF,candPDF);

    % run lip detector if bhatt falls below threshold, otherwise run MS
    if (bhatt(frameNum) < bhattThresh)

        display = ['Running lip detector on frame ',
num2str(frameNum)];
        disp(display);

        % detect lips in current frame
        [modelBbox,bboxFrame] =
detectLips(faceDetector,lipDetector,frameRGB);

        % if lips are found, update model and candidate
        if (~isempty(modelBbox))

            % update model bounding box [x y width height]
            modelX = modelBbox(1);
            modelY = modelBbox(2);
            modelW = modelBbox(3);
            modelH = modelBbox(4);
            modelCenter = [modelX modelY] + round([modelW modelH] / 2);

            % update candidate bounding box [x y width height]
            candBbox = modelBbox;
            candX = candBbox(1);
            candY = candBbox(2);
            candW = candBbox(3);
            candH = candBbox(4);

            % update candidate local center coordinates
            candLocalCenter = round([candW candH] / 2);

            % update model PDF
            model = imcrop(frameRGB, modelBbox);
            modelPDF = createPDF(model);

            % candidate PDF starts off the same as model PDF
            candPDF = modelPDF;

            % generate distance matrix used as LUT
            distanceMatrix = createDistanceMatrix(modelH,modelW);

        else

            % if no lips found, set candBbox to empty
            candBbox = [];

        end
```

```
    else

        % perform mean-shift until threshold is met or max iterations
reached
        while (shift > thresh) && (iteration < maxIteration)

            % increment interation count
            iteration = iteration + 1;

            % crop frame to isolate candidate
            cand = imcrop(frameRGB, candBbox);

            % generated candidate PDF
            candPDF = createPDF(cand);

            % generate weight function from model and candidate PDFs
            weightFunction = createWeight(modelPDF,candPDF);

            % convert candidate c1c2c3 values to integers
            candC1C2C3 = RGB_to_c1c2c3(cand) + 1;
            candC1 = candC1C2C3(:,:,1);
            candC2 = candC1C2C3(:,:,2);

            % numerator and denominator of mean-shift vector
            numerator = [0,0];
            denominator = 0;

            % calculate mean-shift vector numerator and denominator
            for i=1:modelH
                for j=1:modelW

                    weight = weightFunction(candC1(i,j), candC2(i,j));
                    distance = [distanceMatrix(i,j,1),
distanceMatrix(i,j,2)];

                    numerator = numerator + (weight .* distance);
                    denominator = denominator + weight;

                end
            end

            % calculate mean shift vector
            if denominator == 0
                meanShiftVector = [0,0];
                disp('undefined mean-shift vector');
            else
                meanShiftVector = numerator ./ denominator;
                meanShiftVector = round(meanShiftVector .*
candLocalCenter);
            end

            % calculate shift and update shift vector
            shift = sqrt(meanShiftVector * meanShiftVector.');
            shiftVector(iteration,frameNum) = shift;

            % determine new location from mean-shift vector
            candX = candX + meanShiftVector(2);
```

```
            candX = max(minCandX, candX);
            candX = min(maxCandX, candX);
            candY = candY + meanShiftVector(1);
            candY = max(minCandY, candY);
            candY = min(maxCandY, candY);
            candBbox = [candX candY candW candH];

        end

    end

    % Print current time and total number of iterations fo current
frame
    % Only display for integer times
    if ~mod(video.CurrentTime, 1)
        display = [num2str(video.CurrentTime), 's, frame ',
num2str(frameNum)];
        disp(display);
        display = ['    iterations = ', num2str(iteration)];
        disp(display);
    end

    % add bounding box to frame if candidate is found
    if (~isempty(candBbox))
        frameRGB =
insertObjectAnnotation(frameRGB,'rectangle',candBbox,'lip');
    end

    % add frame number to top left
    frameRGB = insertText(frameRGB,[1 1],frameNum);

    % Write frame with bounding box to boudingBox.avi
    writeVideo(boundingBoxVideo,frameRGB);

end

% stop timer
toc;

%% Plot Bhattacharyya coefficient of each frame
figure(2);
plot(bhatt);
ylim([0 1]);
ylabel('Bhattacharyya Coefficient'); xlabel('Frame');
grid on;

%% Plot number of iterations
numShifts = sum(shiftVector > 0);
figure(3);
stem(numShifts);
ylabel('Number of Shifts'); xlabel('Frame');
grid on;

%% close video objects
close(boundingBoxVideo);
```