The Report of MP4

I. Basic Concepts

1.1 Color Segmentation

Color image segmentation simplifies the vision problem by assuming that objects are colored distinctively, and that only gross color differences matter. It therefore discards information about color and brightness variations that provides many valuable cues about the shapes and textures of 3D surfaces. But the resulting simplified image can be processed very rapidly, which can be important in mobile robot applications.

The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image. Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristics.

The simplest method of image segmentation is called the thresholding method. This method is based on a clip-level (or a threshold value) to turn a gray-scale image into a binary image. There is also a balanced histogram thresholding. The key of this method is to select the threshold value (or values when multiple-levels are selected). Several popular methods are used in industry including the maximum entropy method, Otsu's method (maximum variance), and k-means clustering, etc.

Histogram-based methods are very efficient compared to other image segmentation methods because they typically require only one pass through the pixels. In this technique, a histogram is computed from all of the pixels in the image, and the peaks and valleys in the histogram are used to locate the clusters in the image. A refinement of this technique is to recursively apply the histogram-seeking method to clusters in the image in order to divide them into smaller clusters. This operation is repeated with smaller and smaller clusters until no more clusters are formed.

1.2 Algorithm and Implementation

In this homework, I spent most of time on color segmentation and it turns out to be good. It is implemented via three parts: cropping, training detector, and color segmentation. As for Gauss_based color segmentation, I get the deep understanding of such method. However, I didn’t implement its algorithm, but in the result part, I also add my rough results based on gauss method.

To begin with, the crop function is used to create a series of images, which are composed of cropped pixels of skin tones from various images collected on the internet. I downloaded such images from google, collecting those with clear human faces, hands and other parts of skins. I totally cropped 116 sample images.

After the skin pixels are cropped, TrainDector runs next. It loops through each of the training images, which I used to differentiate images. Using these training images, it forms a matrix of hue and saturation color values versus the frequency of such pairs. The matrix is then normalized to one through dividing by the sum of the pixels, especially ignoring whitespace.

Lastly, the normalized matrix is passed as an argument to SegmentColor. SegmentColor picks a threshold value, in this case, I selected 0.001 from empirical data. Then, the input image is scanned and transferred from RGB color model to HSI model. Any HS pairs that are below the threshold are switched to black, leaving only the skin pixels within the training data range with a black background.

1.3 The purpose of the functions

The purpose of this assignment is to implement the skin color segmentation based on H-S histogram and Gaussian method.

II. Results and Analysis

2.1 Test Results

After running the codes under the matlab R2016b, which is used for test the three given images, the results come as followed.

The training images are shown as Figure 1.

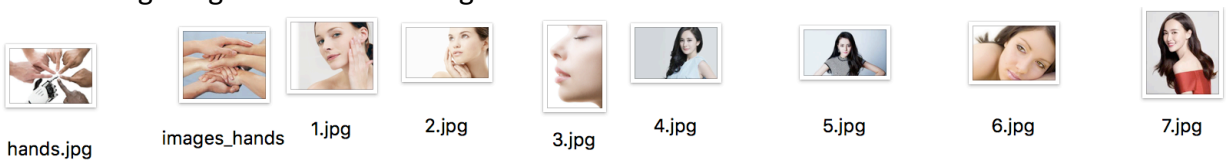

Figure 1 Training Images

Figures 2 shows the testing result of 'joy1.bmp'.

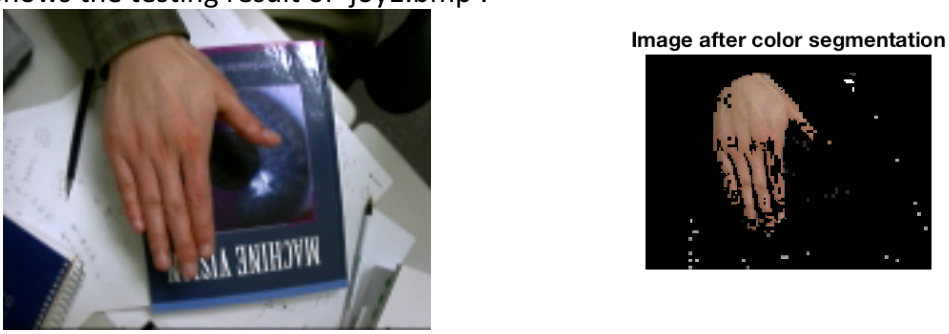

Figure 2 'joy1.bmp'

Figure 3 shows the testing result of 'gun1.bmp'.

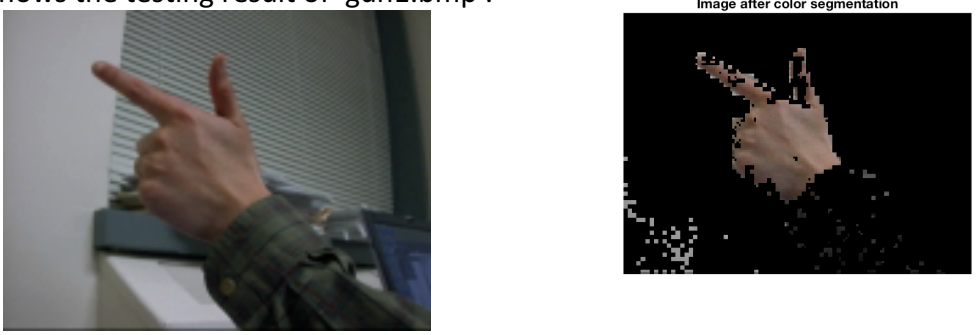

Figure 3 'gun1.bmp'

Figure 4 shows the testing result of 'pointer1.bmp'.

Figure 4 'pointer1.bmp'

Figure 5 shows the results of rough Gaussian method.

.

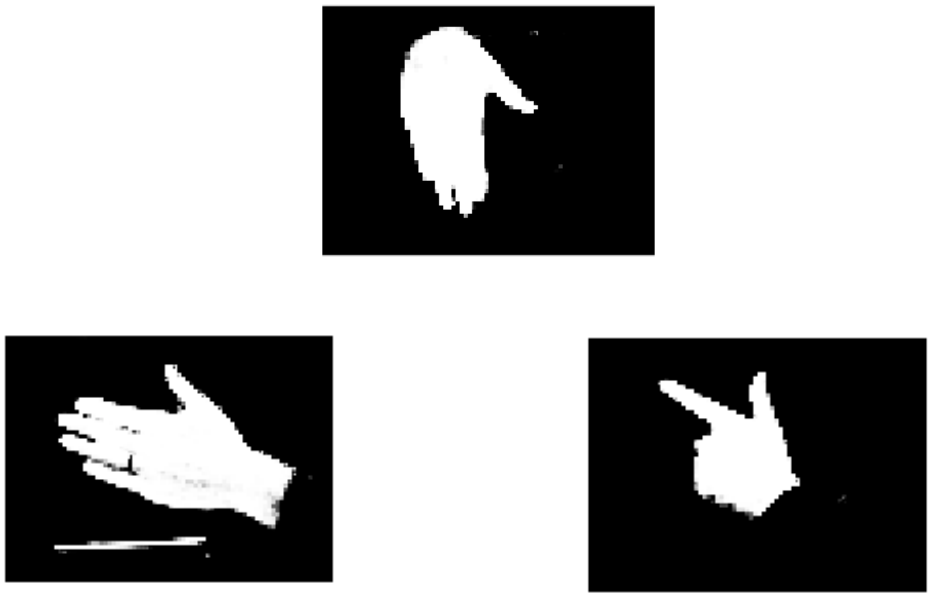

Figure 5 results of Gaussian method

2.2 Result Analysis

From the results, it is clear that they are not perfect. In the edge area, there are many small noisy spots. In my point of view, it might result from the wide variability in background and quality. And it may due to the brightness and contrast of the training images. Also, my codes have some weakness in filtering bright lighting. As for the results of Gaussian method, they prove the accuracy and effectiveness of the Gaussian method. I will finish the extra part of this homework and improve my code in the next week.

2.3 Summary

From MP4, I get a deep understanding of the core of color image segmentation. There are many algorithms of color segmentation. In this assignment, I used the histogram-based method while the results show that there exists imperfection. As for Gaussian method, I will finish the last task soon.

III. Matlab Codes

Part 1 crop function

```
function img_out = crop(img_in)
%this function is used for croping the training images
 %Read in the input image
img_in = imread(img_in);

 %Open up the crop tool for the user
CropImg = imcrop(img_in);
img_out = imshow(CropImg);
end
```

Part 2 TrainDetector function

```
function img_out1 = TrainDetector()
%input all of the training images
images = dir(fullfile('/Users/linji0801/Documents/eecs 332/trainingimages/','*.png'));

% Variables for histogram normalisation
sum = 0;
hspairs = zeros(101, 101);

% For each test file
for file = 1:length(images)
    % Process each test file
    img_in_path = images(file).name;
    img_in = imread(img_in_path);
    img_in_hsv = rgb2hsv(img_in);
    [h, s, v] = rgb2hsv(img_in);

    % For each pixel in the HSV color format image
    for i = 1:size(img_in_hsv, 1)
        for j = 1:size(img_in_hsv, 2)
            % Figure out the histogram bin value of each pixel
            adjusted_h = (round(h(i, j) * 100) + 1);
            adjusted_s = (round(s(i, j) * 100) + 1);

            % Set the bin value correspondingly in the histogram matrix
            hspairs(adjusted_h, adjusted_s) = hspairs(adjusted_h, adjusted_s) + 1;

            % Special case for white pixels
            if (~(adjusted_h == 1 && adjusted_s == 1))
                sum = sum + 1;
            end
        end
    end
end

% normalize the values in the histogram
for i = 1:size(hspairs, 1)
    for j = 1:size(hspairs, 2)
        hspairs(i, j) = hspairs(i, j) / sum;
    end
end

% Output the histogram
img_out1 = hspairs;
end
```

Part 3 SegmentColor function

```
function img_out = SegmentColor(img_in, threshold_values)

% Read in the image to be segmented
Img = imread(img_in);
Img_hsv = rgb2hsv(Img);
[h, s, v] = rgb2hsv(Img);

% A threshold value of 0.001 was found empirically to work the best
threshold = 0.001;

% Loop over the image in HSV color format
```

```
for i = 1:size(Img_hsv, 1)
    for j = 1:size(Img_hsv, 2)
        % Find the adjusted values for pixels in the input image
        adjusted_h = (round(h(i, j) * 100) + 1);
        adjusted_s = (round(s(i, j) * 100) + 1);
        % Blacken out pixels that aren't in the threshold
        if (threshold_values(adjusted_h, adjusted_s) < threshold)
            Img(i, j, 1) = 0;
            Img(i, j, 2) = 0;
            Img(i, j, 3) = 0;
        end
    end
end

img_out = Img;

% Show the final output
figure, imshow(img_out);
title('Image after color segmentation');
```