EEE 6512 Image Processing & Computer Vision

Assignment 03

Name: Chen Yang Username: cyang3 UF id: 52109967

3.14

By counting the numbers in the image, we can get the number of occurrence of each pixel value. Besides, use the occurrence number divided by 16 (because the given image is 4-bit), we can get the frequency of each pixel value. Summarizing the known information we can get the following table:

Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
#Occurrences	1	2	1	3	1	1	1	2	2	2	0	0	0	0	0	0
Frequency	0.0625	0.125	0.0625	0.1875	0.0625	0.0625	0.0625	0.125	0.125	0.125	0	0	0	0	0	0

Therefore, the normalized histogram is

$h_{normalized}$ =[0.0625, 0.125, 0.0625, 0.1875, 0.0625, 0.0625, 0.0625, 0.125, 0.125, 0.125, 0, 0, 0, 0, 0, 0]

By adding the frequency of pixel values each time we can get the cumulative histogram as follows:

$h_{cumulative}$=[0.0625, 0.1875, 0.25, 0.4375, 0.5, 0.5625, 0.625, 0.75, 0.875, 1, 1, 1, 1, 1, 1, 1]

Now, from the information we have obtained, we can calculate the following table:

Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Computation	Multiply 15 by the $H_{cumulative}$, and then round those values to get the pixel values of new image															
Value(equalized)	1	3	4	7	8	8	9	11	13	15	15	15	15	15	15	15

Replace the value in the original image with the calculated value, we can get a new histogram equalized image:

$$I' = \begin{bmatrix} 8 & 3 & 1 & 15 \\ 8 & 13 & 7 & 11 \\ 3 & 7 & 7 & 15 \\ 9 & 13 & 4 & 11 \end{bmatrix}$$

3.15

h_{ref} = [1, 1, 1, 0, 2, 2, 2, 0, 4, 3, 0, 0, 0, 0, 0, 0]

h_{nor}=[0.0625, 0.0625, 0.0625, 0, 0.125, 0.125, 0.125, 0, 0.25, 0.1875, 0, 0, 0, 0, 0, 0]

c_{ref} = [0.0625, 0.125, 0.1875, 0.1875, 0.3125, 0.4375, 0.5625, 0.5625, 0.8125, 1, 1, 1, 1, 1, 1, 1]

After computation we can get : f = [0, 3, 3, 5, 5, 7, 7, 7, 8, 9 ,15, 15, 15, 15, 15, 15]

Therefore, we can get that the new image is :

$$I' = \begin{bmatrix} 5 & 3 & 0 & 9 \\ 7 & 8 & 5 & 7 \\ 3 & 5 & 5 & 9 \\ 7 & 8 & 3 & 7 \end{bmatrix}$$

3.16

No, the second equalization will not change the image. Because in the first time histogram equalization, the image has already be flattened as much as possible. The distribution of pixel values has spread evenly throughout the intensity range of 0-255. Therefore, after the second application, the range of possible intensity is still the same, so the result of second equalization is also the same as before.

3.17

a) For (128, 128, 128), using equation 3.36, the grayscale value is 1/3*(128+128+128)=128

using equation 3.37, the grayscale value is 1/4*(128+128*2+128)=128

because the two results are the same so, the absolute value difference is 128-128 = 0 for all the three colors

b) For (64, 245, 128), using equation 3.36, the grayscale value is 1/3*(64+245+128) = 146

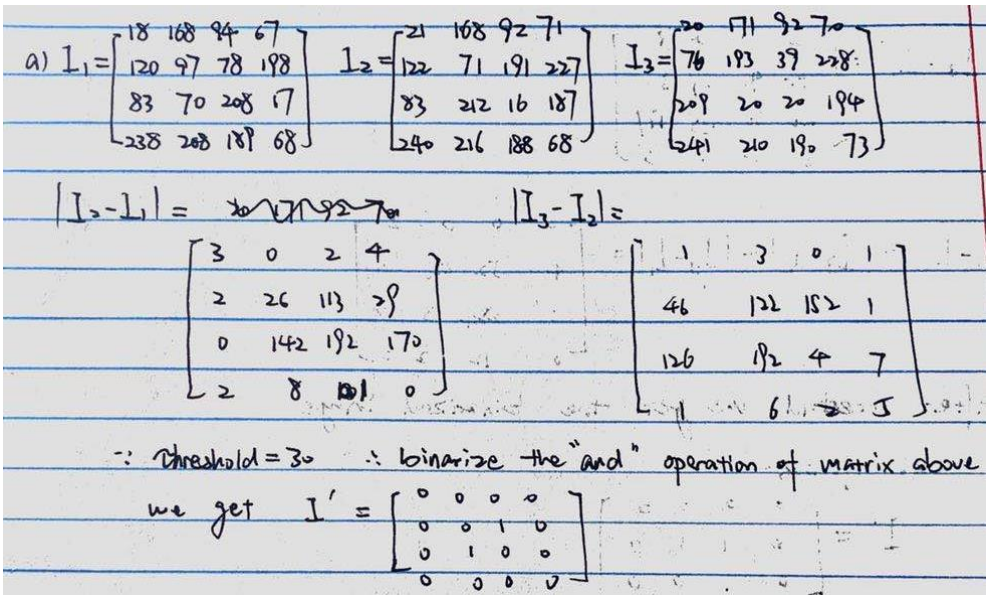
using equation 3.37, the grayscale value is 1/4*(64+2*245+128) =171

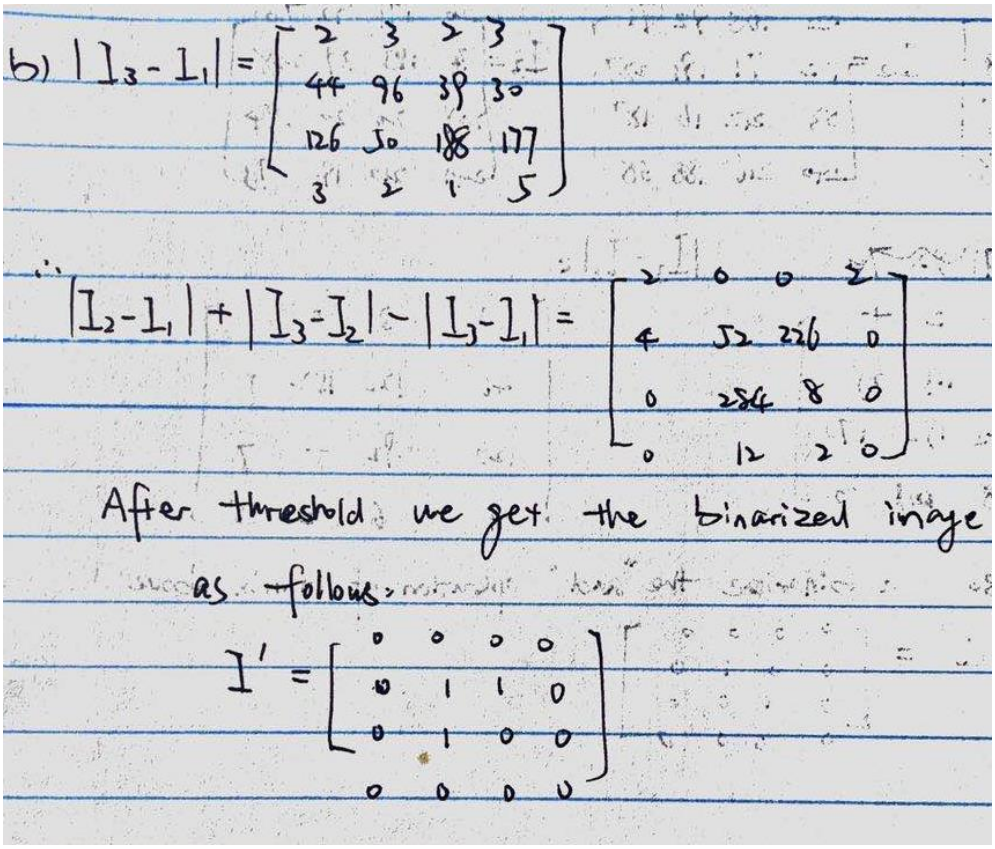
The overall grayscale absolute difference of these two equation is : abs_diff(I) = abs(146-171) = 25

So, for algorithm 3.36 abs_diff(r)= abs(64-146)=82, abs_diff(g)=abs(245-146)=99, abs_diff(b)=abs(128-146)= 18

So, for algorithm 3.37 abs_diff(r)= abs(64-171)=107, abs_diff(g)=abs(245-171)=74, abs_diff(b)=abs(128-171)= 43

3.22





3.31

If the operands were reversed, the output result will be as follows:

For OVER, the reversed output is that the house will cover the tree when they are overlapped.

For IN, the reversed output is that there will be the house part in the overlapping region of tree and house, (originally the overlapped part is tree part, now is house part)

For OUT, the reversed result is the house that without the overlapped part.

For ATOP, the reversed result is that the shape is of tree, but the content is overlapped section of house.

3.32

a) $\alpha_x = x - x_0 = 0.1$ $\bar{\alpha}_x = 0.9$ $\alpha_y = 0.5$ $\bar{\alpha}_y = 0.5$

$\therefore I(0.1, 0.5) = \bar{\alpha}_x \bar{\alpha}_y I_{00} + \alpha_x \bar{\alpha}_y I_{10} + \bar{\alpha}_x \alpha_y I_{01} + \alpha_x \alpha_y I_{11}$

$= 0.45 \times 232 + 0.05 \times 17 + 0.45 \times 201 + 0.05 \times 18$

$= 222.6 \approx 223$

b) $\alpha_x = x - x_0 = 0.2$ $\bar{\alpha}_x = 0.8$ $\alpha_y = y - y_0 = 0.7$ $\bar{\alpha}_y = 0.3$

$I(0.2, 0.7) = \bar{\alpha}_x \bar{\alpha}_y I_{00} + \alpha_x \bar{\alpha}_y I_{10} + \bar{\alpha}_x \alpha_y I_{01} + \alpha_x \alpha_y I_{11}$

$= 0.24 \times 177 + 0.06 \times 82 + 0.8 \times 0.7 \times 8 + 0.14 \times 152$

$= 78.76 \approx 79$

3.37

① $\hat{f}(x) \approx [\alpha^3, \alpha^2, \alpha, 1] \cdot \begin{bmatrix} -0.389 & 1.167 & -1.167 & 0.389 \\ 0.833 & -2 & 1.5 & -0.333 \\ -0.5 & 0 & 0.5 & 0 \\ 0.056 & 0.889 & 0.056 & 0 \end{bmatrix} \cdot \begin{bmatrix} f_{-1} \\ f_0 \\ f_1 \\ f_2 \end{bmatrix}$

② $\hat{f}(x) = \frac{1}{6} [\alpha^3, \alpha^2, \alpha, 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} f_{-1} \\ f_0 \\ f_1 \\ f_2 \end{bmatrix}$

③ $\hat{f}(x) = \frac{1}{2} [\alpha^3, \alpha^2, \alpha, 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_{-1} \\ f_0 \\ f_1 \\ f_2 \end{bmatrix}$

~~① = ② + ③~~

$\therefore$ ① = ② · Catmull-Room + ③ · Cubic-B

$\therefore$ We can Mitchell = $\frac{2}{3}$ Catmull + $\frac{1}{3}$ Cubic-B

Code Section for Assignment 03

Pls call this function by [background_sub('PeopleWalking.mp4');]

background_sub.m

```
% Chen Yang, Username:cyang3, UFID:552109967,
% ECE Department,University of Florida
% Please call this function by [ background_sub('PeopleWalking.mp4'); ]
function background_sub(videoname)

% Read the video and load the parameter
video = VideoReader(videoname);
[height,width,channel] = size(read(video,1));

% Initialize parameter and set the threshould value
whole = zeros(height,width);
th = 80; % Set the threshold value to 80

% Load the first 100 frames of video
for i = 1:100

    frame = double(read( video, i )); % Read the video frames as images
    r = frame( :, :, 1 ); % Split the red channel from the rgb images
    g = frame( :, :, 2 ); % Split the green channel from the rgb images
    b = frame( :, :, 3 ); % Split the blue channel from the rgb images

    frame = 1/4 * (r + 2*g + b ); % Convert rgb into grayscale using 1/4(r+2g+b)
    whole = whole + frame; % Add the cumputed grayscale image of each iteration to sum

end

% Compute the average image and plot it
average = 0.01 * whole ;
imshow(uint8(average));
title('The Average Image of the First 100 Frames')
```

Figure1. Computation of Average Image of First 100 Frames

Explanation:

First, by using VideoReader, we can get the input video as input, and then we use read to get the parameters of the video. Next, we initialize a variable called whole as preparation by the parameters we get last step. In this algorithm, we pick 80 as the threshold value. After testing, the threshold value in range [80,120] performs good, we pick the relatively better value 80 as the final choice. Then, we use a for loop to read the first 100 frames of video as image, and split the three channels of those images. We using the algorithm discussed in class (1/4*(r+2g+b)) to convert those rgb frames into grayscale images. I decided to use this algorithm because Prof Woodard once said people's eyes are more sensitive to green, and this will leads to better image to differentiate the details. At the end of the for loop, we add those image together in each iteration. After ending the loop, we divide the obtained 'whole' by 100 to get the average image. Finally we plot the average image out by imshow int version of average.

background_sub.m*

```
for j = [1, 25, 50, 75, 100]
    % Load the 1st, 25th, 50th, 75th, 100th frame of video as rgb image
    frame = double(read(video, j));
    figure

    % Split the three channels from the rgb image
    r = double(frame(:, :, 1));
    g = double(frame(:, :, 2));
    b = double(frame(:, :, 3));

    % Convert rgb into grayscale using algorithm 1/4(r+2g+b)
    frame = 1/4 * (r + 2*g +b);

    % Plot the original image from grayscale video
    subplot(2, 1, 1)
    imshow(uint8(frame))
    title([int2str(j),'th Frame of Grayscale Video'])

    % Compute image after substraction using absolute difference
    comp = abs(frame - average);
    comp(comp>=th) = 255; % Adjust the intensity greater than or equal to threshold to 255
    comp(comp<th) = 0; % Adjust the intensity less than threshold to 0

    % Plot the image after substraction and thresholding
    subplot(2, 1, 2)
    imshow(uint8(comp));
    title(['Processed Result of Frame ', int2str(j),' with Threshold Value: ', int2str(th)])



end

end
```

Figure2. Computation of Thresholded Image after Substracting the Average Image

Explanation:

In this section , we will get five specific frames and then compute the after background sub image of those 5 images. First, we use a for loop to get those frames. Then use the same method as the previous part to separate the color channels of these frames. Next, use the same method to converted those frames into grayscale images. (1/4(r+2g+b)), (the reason why pick this equation has already been explained in the last paragraph) and plot the processed grayscale images. After that, we use the absolute difference algorithm discussed in class to compute the background_sub images. Use the initially selected threshold to binarize the obtained picture by setting those values greater than threshold to 255 and setting those values less than threshold to 0. Finally, plot the processed image out. That is end of the second for loop and the whole algorithm.

Test Result on 'PeopleWalking.mp4'

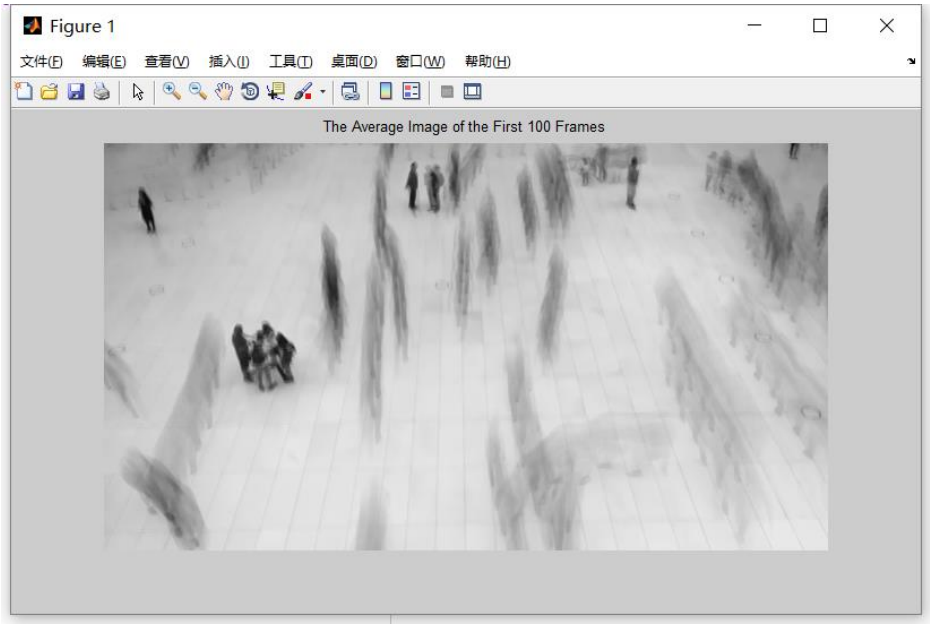


Figure3. Average Image of First 100 Frames

Figure4. Original-Background_Sub Comparison of 1st,25th frame

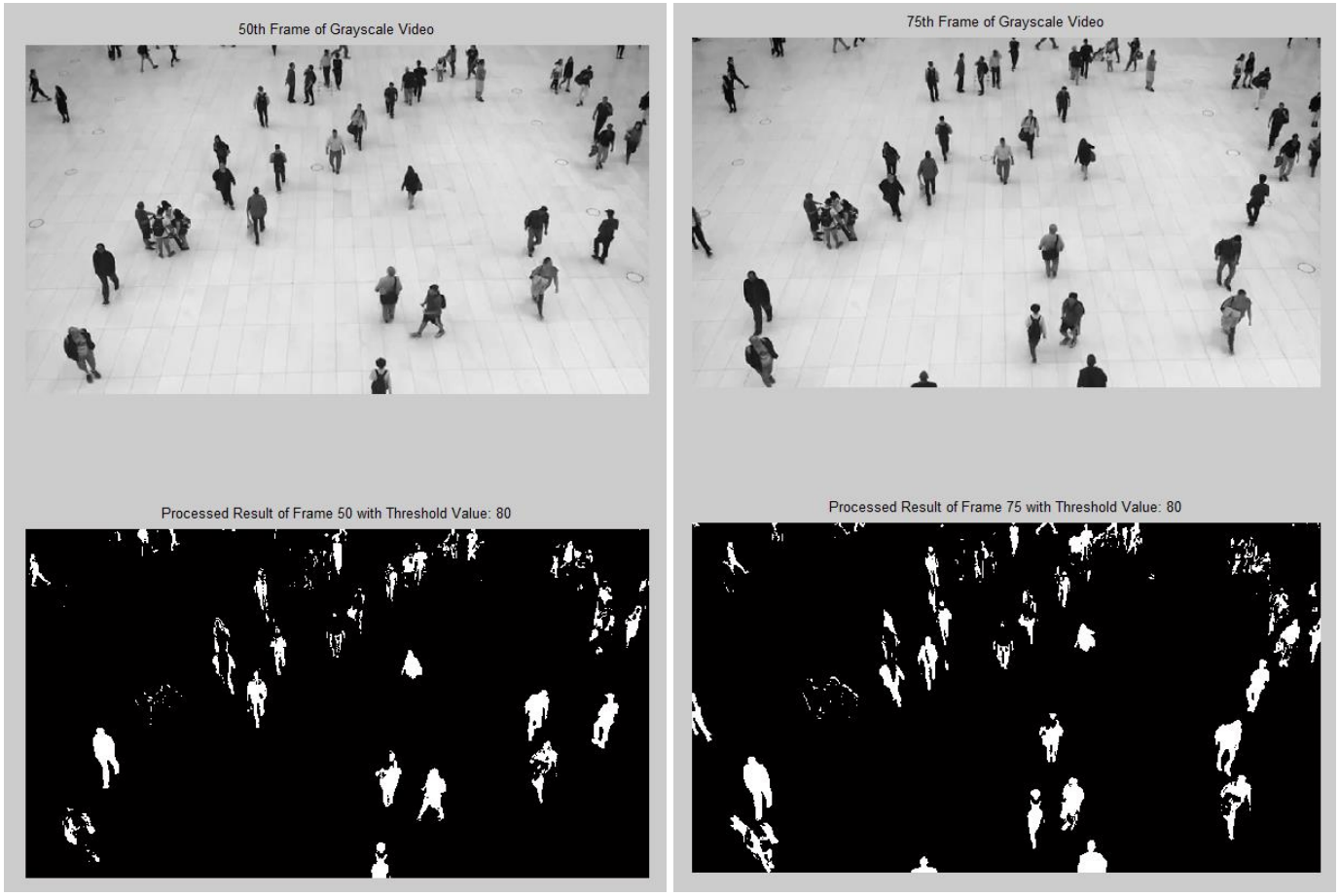


Figure5. Original-Background_Sub Comparison of 50th,75th frame

Figure6. Original-Background_Sub Comparison of 100th frame

Conclusion of the Algorithm

After testing, I find that this algorithm has many limitations. I have tried many different thresholds. I found that a relatively good performance threshold range is 80-120. Within this range, almost everyone will be marked in the background_sub image. However, no matter how much the threshold is set, it cannot perfectly mark the complete outline of all people. In the end I think the main problem is on the average image. Next, I will elaborate on main reasons I think the algorithm performs poorly and how to improve this algorithm's performance.

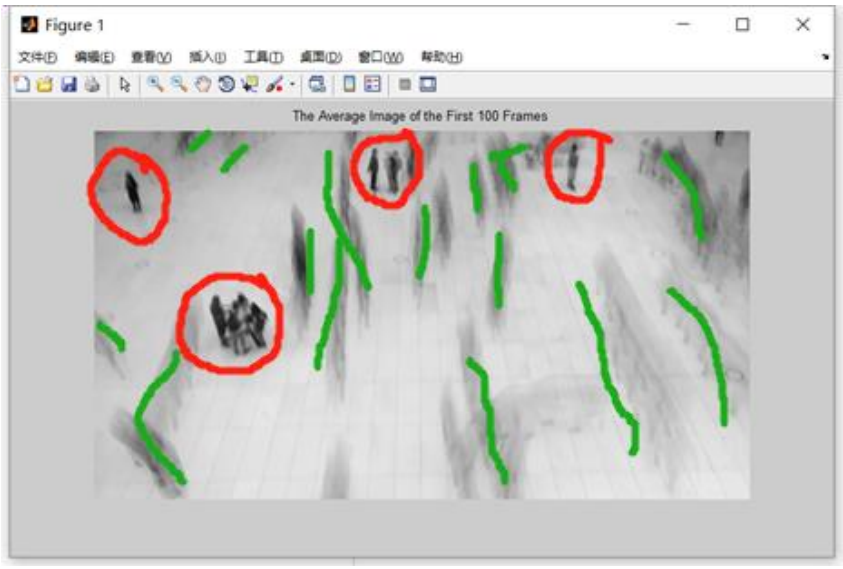


Figure7. Problem of Average Image

In the comparison picture in the previous part, we can find that not all pedestrians are perfectly marked in the background_sub image, and some pedestrians only have some white bright spots around them. This is due to the insufficient number of samples in the process of averaging. Because we are told to use the first 100 frames to compute the average grayscale image, which is not a complete selection, we cannot perfectly extract the full background out. Some people did not make any movements or postures in the first 100 frames, just like the picture above is circled in red, these people are identified as background by the algorithm in the process of averaging, this results in these people being hardly highlighted in the background_sub image. Besides, the goal of averaging the image is to weaken the influence of people as much as possible so that we can get a clean background image, due to the insufficient selection of the number of frames, the influence of some people's movements in the first 100 frames is not weakened enough by the average, which leads to the appearance of many elongated gray shadows in the average graph obtained, as marked by the green curve in the picture above. These shadows directly lead to the appearance of some gray or light gray pixels with relatively large values in the background_sub graph. These pixels make the threshold selection very limited.

In addition to the above, there are also some factors of the video itself that need to be considered. In this video called [PeopleWalking.mp4], the colors of some pedestrians' clothes are very close to the background color. This also leads to the fact that in the process of averaging, these parts of pedestrian clothes may be identified as background by the algorithm. This leads to the obvious movement of these pedestrians, but because the colors are too close, the outline of the entire pedestrian cannot be fully displayed in the background_sub image. However, this algorithm also has some special advantages. For example, even if some people's movements are extremely small, this algorithm can also mark these relatively small position changes. (Although complete object detection cannot be done).

This algorithm can be improved by the following methods：

1. Use the whole scene of the video to find the average image.
2. Before using the algorithm, use the histogram_equalization method to pre-process the video to increase the contrast of the image in the video.

In this way, with appropriate threshold selection, the algorithm will perform better.

Reference

[1] B. H. Brown, R. H. Smallwood, D. C. Barber, P. V Lawford, and D. R. Hose, *Image processing and analysis*. 2004.