ECE 510 EmbV I
Fall 2016
Lab03 notes

Histograms and Contours

Table 1. Project command line arguments.

-c		Enable face capture on mouse left button click.
-g	image_file	Image showing a group of people.
-t	image_file	Face template.

Task 01 – Histogram based face detection

Capture faces

For this task I decided to automate the process of capturing an ROI (a face) and saving the selection to the disk. I created method `Lab03::CaptureFaces` to read an image file, set a mouse event callback function and display the image. The actual capture is handled by `Lab03::ExtractROI` which is the mouse event callback function. On mouse left button click the callback function selects an ROI of 20 rows and 16 columns centered on the mouse cursor and save the image fragment to the disk to *<solution_folder>\images\task01*.

The reason for the small size ROI was to minimize the histogram "noise" if we decide to use any of these as a template. In this case the "noise" is represented by the pixels that don't belong to a "face".

I saved all the faces to the disk in *<solution_folder>\images\task01\all_faces*. This is what they look like:

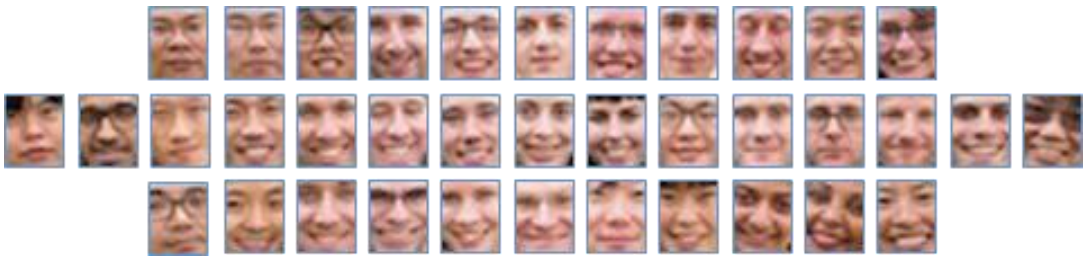

Figure 1. All the faces in the test image.

Generate color histograms

Images are saved to "C:\OpenCV\Solution\images\lab03\task01\". Please note that the application will not create the directory. If the location is not found no output is saved to disk. In order to run this application it is best to unzip the solution to "C:\OpenCV\", the same location where it expects the OpenCV to be installed.

The color reduction is handled by the lab03::ColorReduce method. Figure 2 & Figure 3 show the group image and the face template after color reduction. I generate the histogram of the face template using the Histogram class and then generate the back projection using the Histogram::BackProjection method.

Figure 2. Face template. On the left the original. On the right after color reduction.

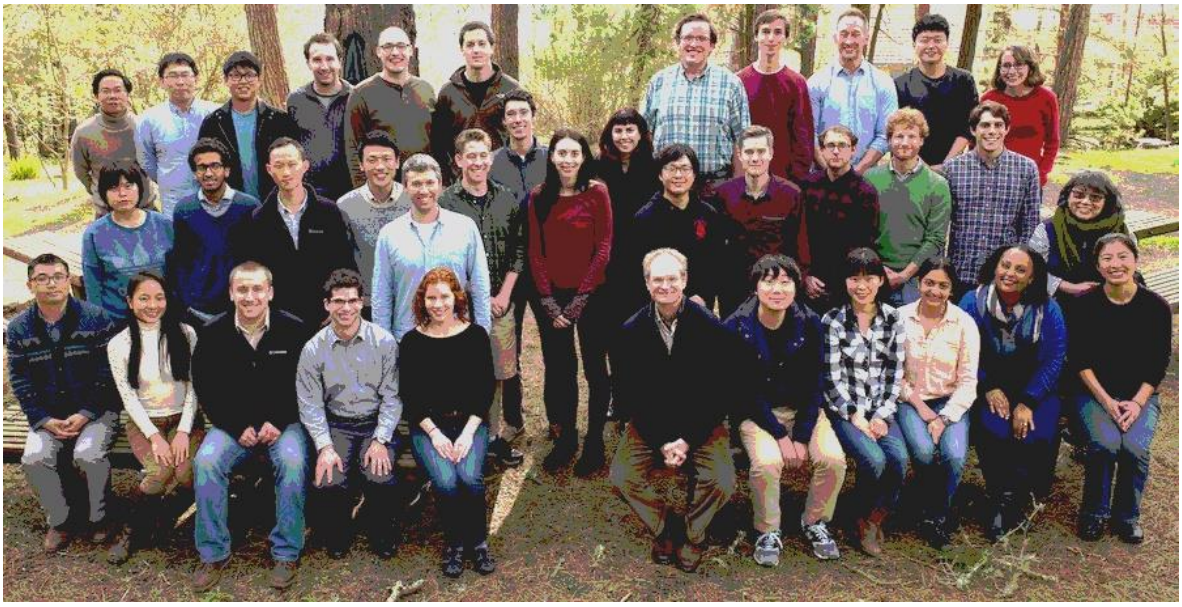

Figure 3. Group image after color reduction.

The back projection with the selected template is showing all faces but in addition to that the algorithm captured hands as well as a lot of noise in the area surrounding the group.s

Figure 4. Back projection of the group image with the histogram of the template image.

Figure 5. The back projection with a threshold of 0.78.

Using the above threshold image is not at all easy to determine the true/false positive/negative rates. Even though we can clearly see all faces there's too much noise.

An idea is to apply a blurring filter on the group and template images. Figure 6 and Figure 7 show the back projection after applying median and a Gaussian filter respectively on the group and template images. There's still a lot of noise but to a somewhat less extent than the case without blurring.

Figure 6. Back projection after applying a 3x3 median filter on the group and template images.

Figure 7. Back projection after applying a 3x3 Gaussian filter on the group and template images.

Attempt to improve the algorithm

Another idea is to apply the median filter only on the template image followed by median filter on the back projection threshold image. To that we can apply opening to eliminate trace noise and two times dilate to generate large blobs around the detection areas. We can use the blobs image (Figure 8) as a mask on the original group image to assist in the evaluation of the true/false positive/negative rates (Figure 9).

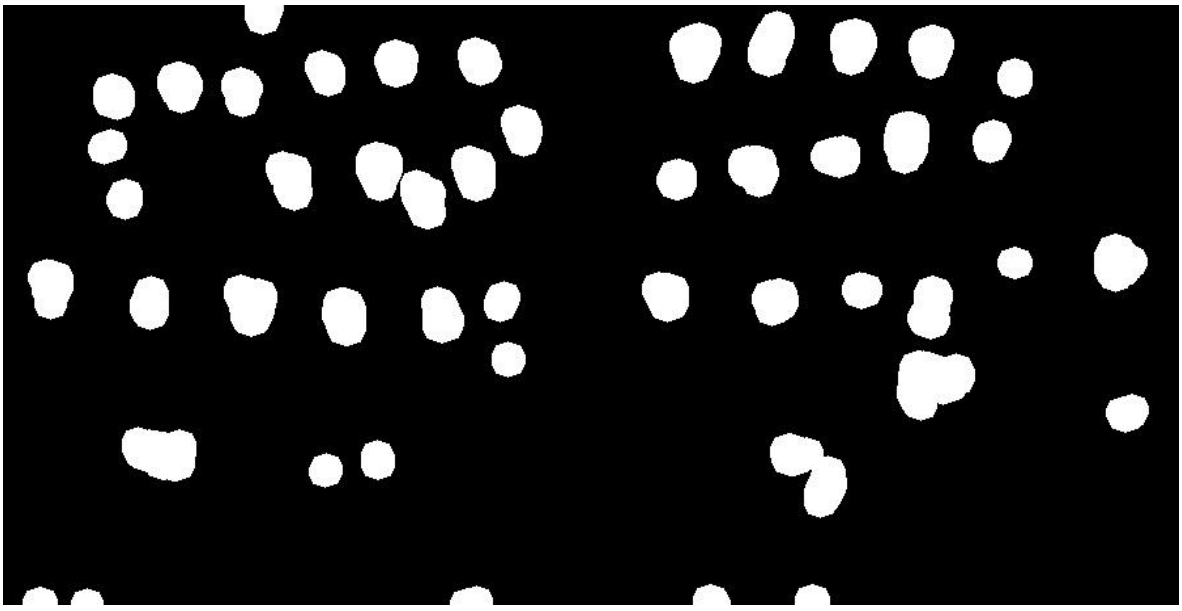

Figure 8. Blobs indicate possible detections.

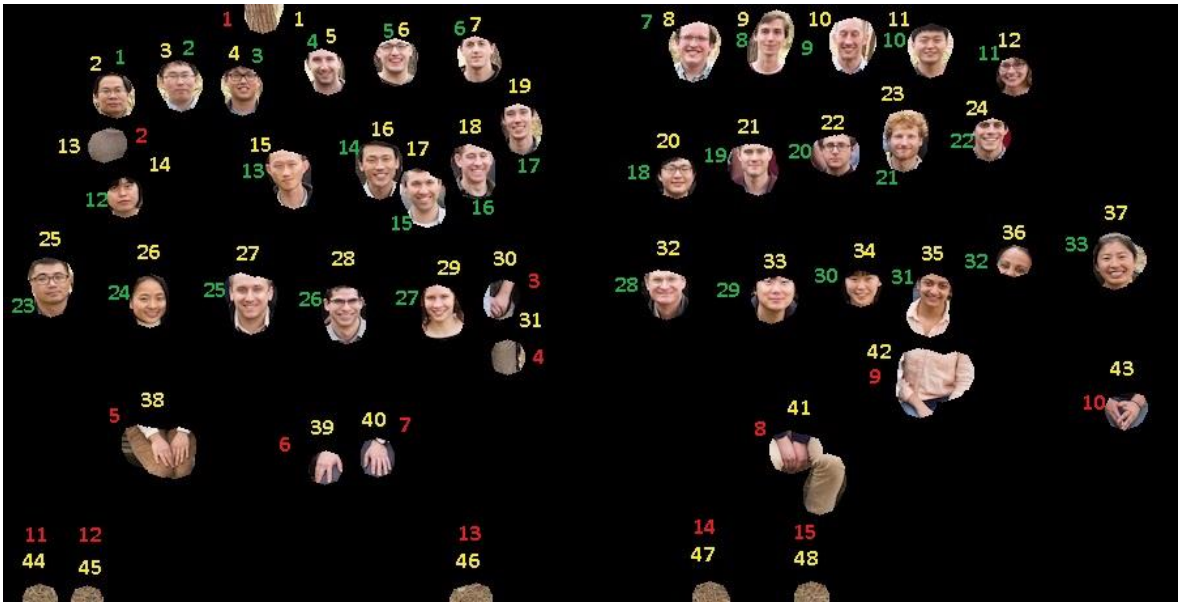

Figure 9. The original group image masked with the blobs image.

Based on this information we calculate:

Gold standard = 37 faces

Total Detections = 48 faces

True Positive = 33 faces

False Positive = 15 faces

True Positive Rate = TPR = 33 / 37 = 89%

Positive Predictive Value = PPV = 33 / 48 = 69%

False Discovery Rate = FDR = 15 / 48 = 31%

Overall these results look good but I'd like to note that the algorithm may not have applicability to other circumstances (lighting conditions, more diverse faces, different backgrounds, etc).

An interesting observation is that by blurring the template image (3x3 median filter) the PPV (positive predictive value) increase substantially relative to the version without blurring.

Task 02 – Template matching

For this task I created the method lab03::Task02. Images are saved to "C:\OpenCV\Solution\images\lab03\task02\". Please note that the application will not create the directory. If the location is not found no output is saved to disk.

The algorithm is straightforward:

1. Load input images.
2. Convert to gray images.
3. Perform matchTemplate with the normalized similarity method (CV_TM_CCORR_NORMED).
4. Normalize the result.
5. Parse the result and collect the locations of the matches above a certain threshold.
6. Draw the green boxes around the detection points and display/save the image.

The results were surprisingly low in my opinion with numerous false positives and very few true detections overall (Figure 10). I tested blurring the template image before matchTemplate (both with medianBlur and GaussianBlur) but there was no notable improvement (Figure 11 & Figure 12). In fact, the blurring led to more matches but the most of them are false positives.

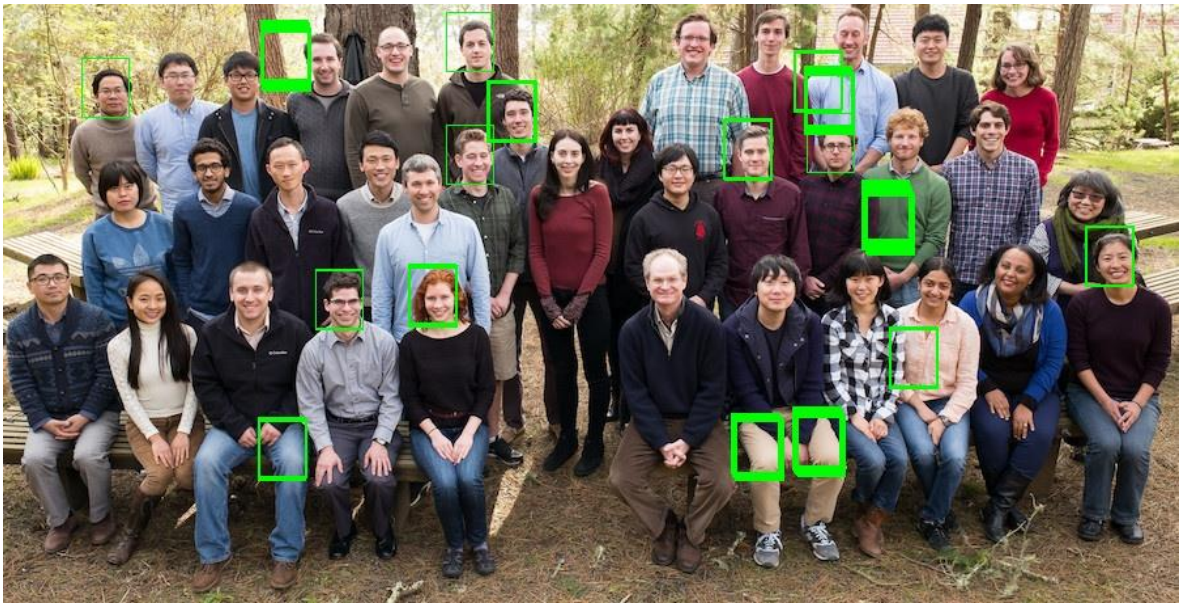

Figure 10. Face detection using the OpenCV function matchTemplate.
Here I used the image template provided with lab03 materials.
Threshold value: 0.93.

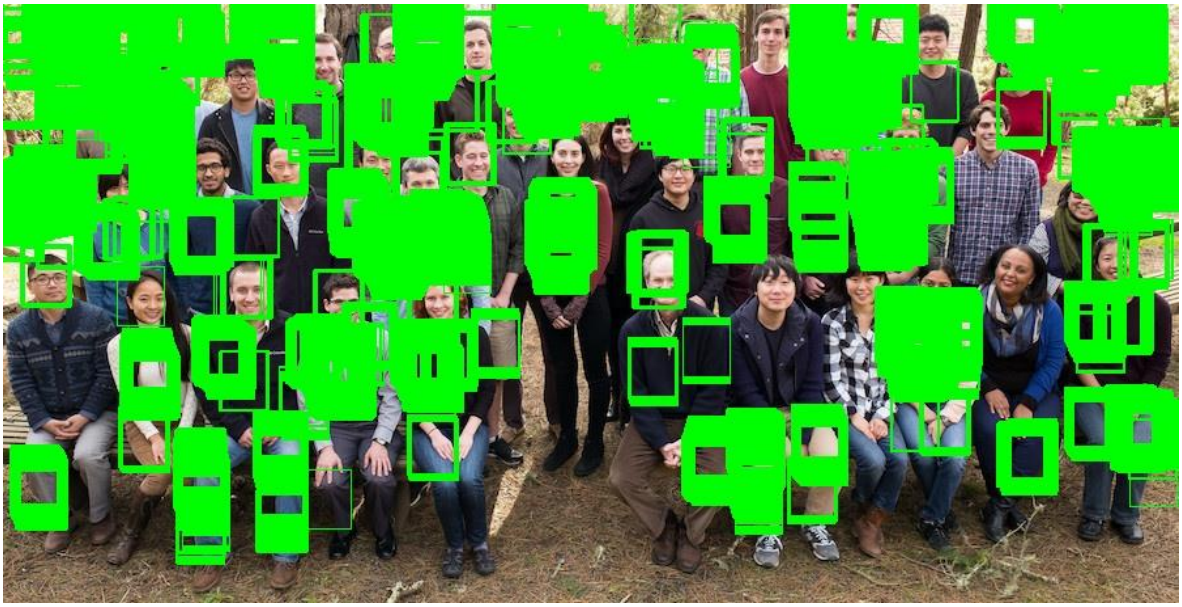

Figure 11. The same operation with a medianBlur applied on the image template.
Threshold value: 0.93.

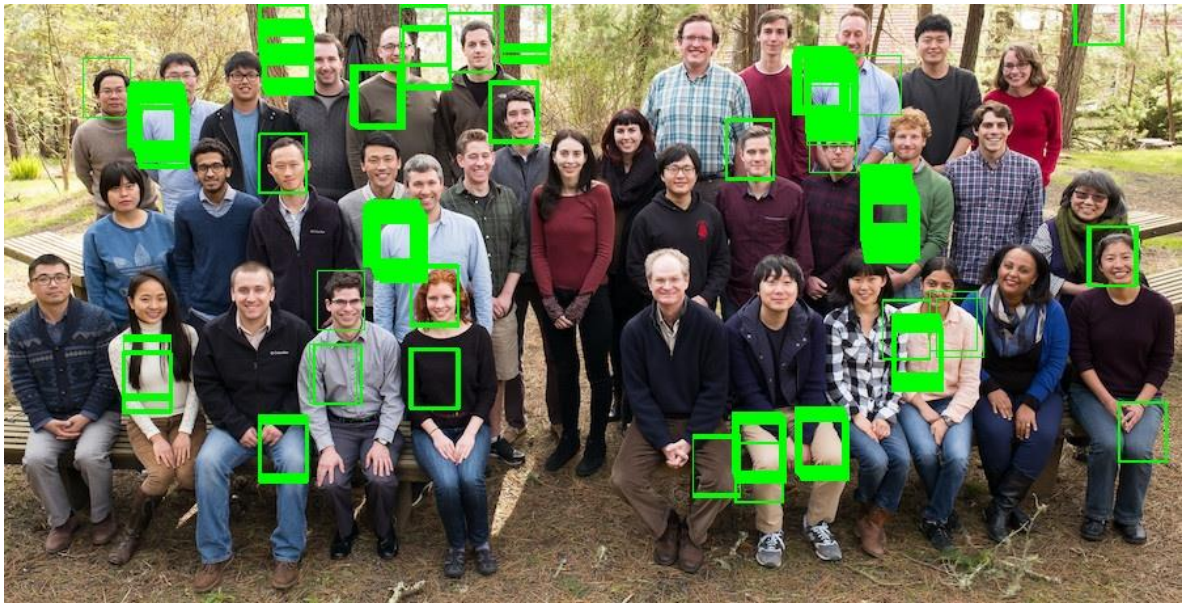

Figure 12. The same operation with a medianBlur applied on the image template.
Threshold value increased to 0.95.

I decided to test my hypothesis that the algorithm might be able to do better using one of the template images (Figure 13) I captured at the beginning. My ideas was to use a smaller box when capturing in order to minimize the amount of "noise" – that is the number of non-facial pixels.

Figure 13. Custom template image.
C:\OpenCV\Solution\images\lab03\task01\all_faces\face15.jpg

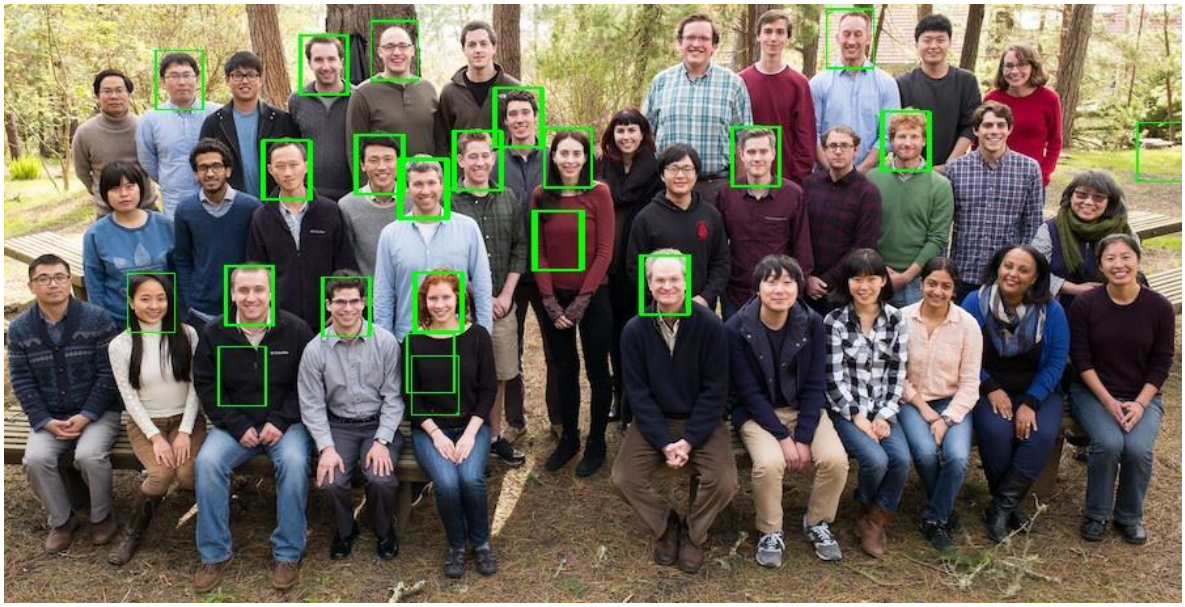

Figure 14. Template matching with a custom image template.
Please note a much better detection rate than previously.
Threshold value: 0.95.

Figure 14 shows a substantial improvement in detection rate. Not only the algorithm is detecting more actual faces but the false positive rate is pretty low. The algorithm detected 17 faces (some were matched several times making the accurate calculations difficult) and only 4 false positives. Still the number of true positives is nowhere near the golden truth. Further improvements of the template image (maybe averaging all the capture template images into one, converting to HSV, etc.) might be able to improve the detection rate.

Task 03 –

The use of findContours causes my application to crash when exiting. I spent a while researching what the cause for that might be and the most plausible explanation seems to be that OpenCV sometimes conflicts with VS2015. Fortunately the crash doesn't seem to interfere with the normal operation of the code.

The algorithm I followed:

1. Load the input image.
2. GaussianBlur with a 3x3 kernel
3. Canny with
   threshold1 = 100
   threshold2 = 200
4. findContours with
   mode = CV_RETR_TREE
   method = CV_CHAIN_APPROX_SIMPLE
5. Display the result.

As you can see from Figure 15 and Figure 16 there's no notable difference. The faces did not appear as separate contours.

Figure 15. Canny edges. Threshold: 100-200.

Figure 16. Edges after findContours.

Somewhat better results are obtained by applying a 9x9 medianBlur before Canny edge detection.

Figure 17. Same operation but blurring is done with a 9x9 medianBlur.