CENG466 - Fundamentals of Image Processing Take Home Exam 1 Report

1st Fatih Develi
Computer Engineering
Middle East Technical University
Ankara, Turkey
fatih.develi@ceng.metu.edu.tr

2nd Berk Arslan
Computer Engineering
Middle East Technical University
Ankara, Turkey
arslan.berk@metu.edu.tr

***Abstract*—Geometric transformations, histogram processing and edge detection are some basic applications in image processing. This document presents the practice of such applications.**

***Index Terms*—image processing, transformation, histogram normalization, histogram equalization, histogram specification, edge detection, blurring**

I. Introduction

This document is the presentation for the THE1 (Take Home Exam 1) for the course Fundamentals of Image Processing. As geometric transformations; rotating, scaling, shearing and reflection were practiced. In Histogram Processing section, exampes of histogram equalization and histogram specification were applied. Lastly, some examples of edge detection were presented.

Some important code snippets that show the procedures which perform the core image processing operations are included.

II. Question 1 - Geometric Transformations

In this section, necessary transformations were applied to previosly manipulated images to obtain the original images.

A. A1

Fig. 1. A1

In this part, the given image is shown in Fig. 1. It is obvious that the image has been rotated 90^o counter-clockwise. To obtain the original image, affine transform with forward mapping can be used.

$$[x\ y\ 1] = [v\ w\ 1] * T \tag{1}$$

The procedure to find the original image is to apply the equation (1), where T is the affine matrix.

The rotation operation itself is not enough to obtain the original image. A translation is needed to get rid of the empty areas in the new image.

$$T = \begin{bmatrix} cos90 & sin90 & 0 \\ -sin90 & cos90 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 334 & 0 & 1 \end{bmatrix} \tag{2}$$

The affine matrix is the product of the rotation matrix and the translation matrix as in (2).

Below code snippet shows the core of the procedure explained above. The mapping is done for each color channel.

```
T = T_rotation * T_translation;

for w = 1:height % y axis
    for v = 1:width % x axis
        new_pixel = [v w 1]*T;
        new_A1(new_pixel(2),new_pixel(1),1) ...
            = A1(w,v,1); %r
        new_A1(new_pixel(2),new_pixel(1),2) ...
            = A1(w,v,2); %g
        new_A1(new_pixel(2),new_pixel(1),3) ...
            = A1(w,v,3); %b
    end
end
```

B. A3

The given image is Fig. 2. The required operation is translation of the image. To find the translation amount, upper left corner of the inner (real) image was found first (referred to as "edge"). The coordinates of the edge is the amount of translation needed for each axis. Following code snippet shows the procedure of translation with inverse mapping.

```
T = [1 0 0; 0 1 0; -edge(2) -edge(1) 1];
% Using inverse mapping: [v w 1] = [x y 1] * T_inv

new_A3 = zeros(newheight, newwidth, 3, 'uint8');

for y = 1:newheight
```

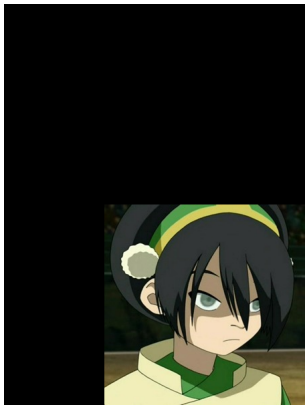

Fig. 2. A3

```
    for x = 1:newwidth
        source_pixel = [x y 1] / T; % [x y 1] * inv(T)
        new_A3(y,x,1) = A3(source_pixel(2), ...
            source_pixel(1), 1);
        new_A3(y,x,2) = A3(source_pixel(2), ...
            source_pixel(1), 2);
        new_A3(y,x,3) = A3(source_pixel(2), ...
            source_pixel(1), 3);
    end
end
```

C. A4

Fig. 3. A4

The given image is Fig. 3. In the process of obtaining the original, a different approach was followed for this image. Each pixel row in the image is shifted to right by some amount. This shift amount is inversely proportional to the height of the pixel rows. By following this logic, each row must be shifted to left by $emptypixels * (y/h)$ pixels, where emptypixels is the number of empty (black) pixels at the bottom left corner of the image, y is the height of the current row, and h is the height of the image.

Following code snippet executes this idea, applying it to each color channel.

```
for x = 1:realwidth
    for y = 1:height
        shift = round(emptypixels * (y/height));
        new(y, x, 1) = A4(y, x+shift, 1);
        new(y, x, 2) = A4(y, x+shift, 2);
        new(y, x, 3) = A4(y, x+shift, 3);
    end
end
```

D. A5

Fig. 4. A5

The given image is Fig. 4. This image is the horizontal mirror image of the original. Necessary operation is to map the pixels with coordinates (x, y) to $(height - x, y)$.

Following code snippet shows this operation to each color channel.

```
for x=1:height
    for y=1:width
        new(x,y,1) = A5(height+1-x,y,1);
        new(x,y,2) = A5(height+1-x,y,2);
        new(x,y,3) = A5(height+1-x,y,3);
    end
end
```

III. Question 2 - Histogram Processing

A. B1

The procedure is explained in detail in the next section. This part is done by following the same procedure as $B2$, the only difference is the operations are applied to 3 color channels. Refer to following section section for the details.

Following are the results.

Fig. 5. B1 original image

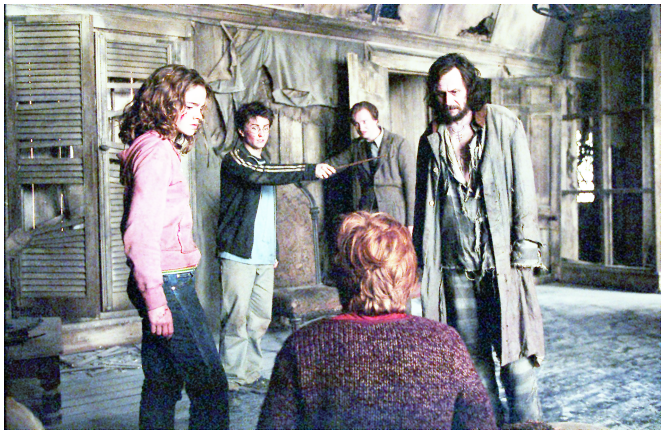

Fig. 6. B1 equalized histogram

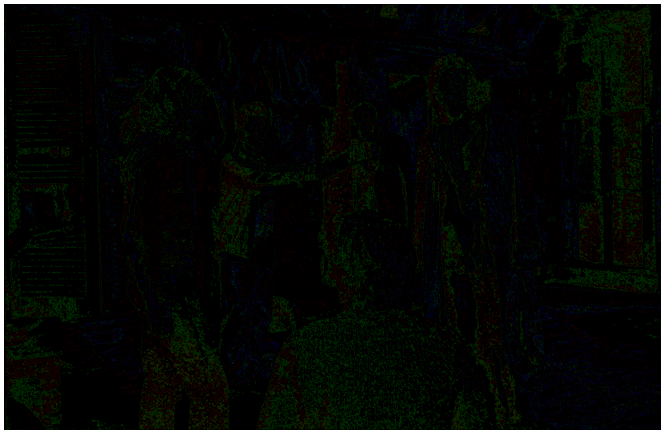

Fig. 7. B1 matched histogram

Fig. 8. B2 original image

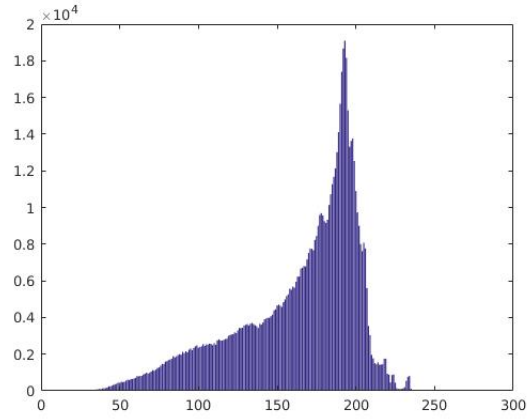

Fig. 9. B2 histogram

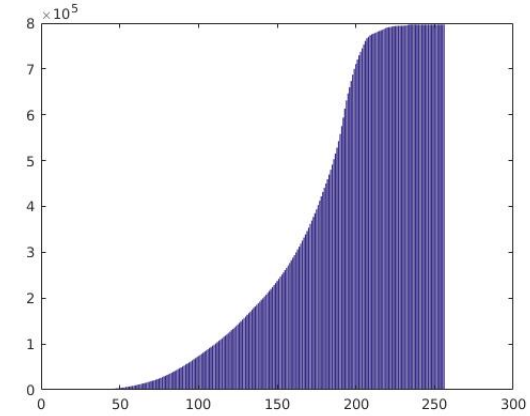

Fig. 10. B2 cumulative histogram

B. B2

1) Equalization:

In this part, histogram equalization and histogram specification techniques were applied to the given image.

First, the histogram of the image was calculated. The histogram is an array representing how many times a brightness level is encountered in the image. The program goes through each pixel in the image and increments the corresponding value by one.

Fig. 9 shows the histogram of the image. Next, this histogram is converted to a cumulative histogram.

Fig. 10 shows the cumulative histogram of the image.

Next, the mapping function should be defined. The mapping function is defined by multiplying the values in the cumulative histogram by a constant.

$$c = (L-1)/(height * width); \tag{3}$$

The constant is calculated according to (3), where $L-1$ is the number of brightness levels. Following code snippet shows the creation of the mapping function.

```
c = 255/(height*width);
B2_mapping = zeros(1,256);
for i = 1:256
   B2_mapping(i) = round(c * B2_histogram(i));
end
```

The new image is created using this mapping.

```
for y = 1:height
   for x = 1:width
      B2_histeq_output(y,x) = B2_mapping(B2(y,x)+1);
   end
end
```

This new image is the equalized image. The histogram of it is calculated again.

Fig. 11 shows the equalized histogram.

Fig. 12 shows the equalized image.

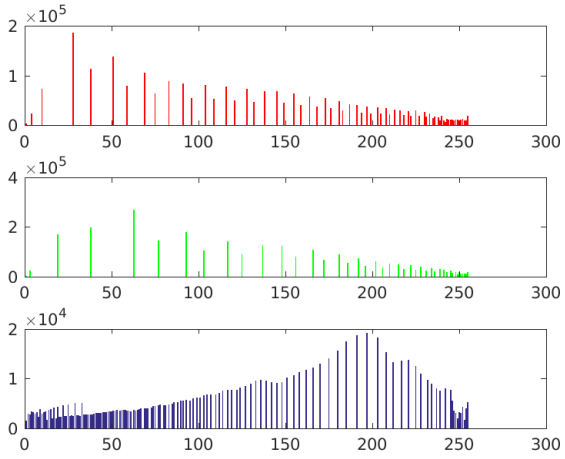

Fig. 11. B2 equalized histogram

Fig. 12. B2 equalized image

2) Specification: The next step is the histogram specification (matching).

Fig. 13. B2 reference image

Fig. 13 shows the reference image for histogram specification. Calculating histogram and cumulative histogram processes are repeated for this image. Then, mapping function of the equalization of the reference image is created.

```
mapping = zeros(1,256);
for i = 1:256
    [~, ind] = min(abs(B2_ref_mapping - B2_mapping(i)));
    mapping(i) = ind - 1;
end
```

Next, new image is created using this mapping.

```
for y = 1:height
    for x = 1:width
        B2_histmatch_output(y,x) = mapping(B2(y,x)+1);
    end
end
```

Fig. 14. B2 histogram matched image

Fig. 14 shows the resulting image.

3) Conclusion: Clearly it can be seen that histogram equalization increseas the constrast of the image. While this can work for some images such as Fig. 12, it does not always produce results, as seen in Fig. 6. To solve this problem, a reference histogram can be used, which is done via histogram specification. By using a reference image in part B1, histogram specification produced better results as seen in Fig. 7.

IV. Question 3 - Edge Detection

A. Convolution

In this part, convolution of an image with 2x2 and 3x3 filters was implemented as a function. The function checks the size of the filter and runs different processes for each filter. The problem with this process is, some pixels are missing when processing the borders of the image. For 2x2 filters the function uses a roll-technique for these missing pixels, meaning that at the border, it processes the pixels from the other side of the image for the missing ones. For 3x3 filters the function uses the values of closest pixels to handle out-of-border pixels.

B. Edge Filters

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, S_y \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$R_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, R_y \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

These are the given filters that will be used in convolution. In this part, the images are convolved with each of the filters to obtain partial edge maps.

C. Edges

$$BX\_S_{edges_{i,j}} = \sqrt{conv(BX, Sx)^2{}_{i,j} + conv(BX, Sy)^2{}_{i,j}} \quad (4)$$

$$BX\_R_{edges_{i,j}} = \sqrt{conv(BX, Rx)^2{}_{i,j} + conv(BX, Ry)^2{}_{i,j}} \quad (5)$$

In this part, the partial edge maps obtained in the previous step are combined together to produce the full edge maps using (4) and (5).

Following code snippet is an example to show the processing.

```
for y = 1:C1_height
    for x = 1:C1_width
        C1_S_edges(y, x) = ...
            round(sqrt(C1_Sx(y,x)^2 + C1_Sy(y,x)^2));
        C1_R_edges(y, x) = ...
            round(sqrt(C1_Rx(y,x)^2 + C1_Ry(y,x)^2));
    end
end
```

At the end of this process, 6 images are created as the edge maps of 3 input images with 2 different edge filters. Below is an example of these images.

Fig. 15. Edge map of C3

D. Blurring

$$GaussianMatrix = \begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

In this part, above filter is defined and applied to the images using the convolution function.

E. Blurring Edges

In this part, the procedures applied in $EdgeFilters$ and $Edges$ sections were re-applied to the blurred images created in the previous part. 2 filters were applied to 3 blurred images, producing 6 new images. Below is an example edge map created using the blurred input.

Fig. 16. Edge map of blurred C3

F. Conclusion

In this section, practice of edge detection was shown. The two types of edge filters used were Roberts and Sobel filters. While Roberts filter was good at detecting diagonal edges, Sobel filter was good at detecting horizontal and vertical edges.

The main issue with this practice was the noisy parts in the images which caused inaccurate edge detection. One way to solve this problem is to blur the images before applying edge detection filters. Once blurring was applied, the results were slightly better for the images which had noise. Fig. 16 shows a better result compared to Fig. 15, since blurring eliminated some of the noise and helped the edge detection perform better.

On the other hand, blurring did not do much in image $C2$ (not shown here), since that image does not have noise at all. Edge detection of blurred image only caused a production of a softer edge map.

Based on these results, it can be concluded that with the usage of right gaussian blur, edge detection can be improved.