ENGN4528/6528 Computer Vision – 2015
Computer-Lab 1 (C-Lab1)

Sai Ma - u5224340

March 2015

1 Task-1: Basic Image I/0

In the task, we need take a frontal face photo of myself, and apply sample I/O performance. The following is my frontal face photo 1 by *imread()*:

Figure 1: Frontal Face Photo

Then, I perform the Matlab function *size()* to get size of this image. In the first task, it asked us to resize figure to 1024×1024 without distorting. I decided to cut this figure by *imcrop()*, and performed *imresize()*. The code is following:

```
% 3. Read image
img = imread('photo_U5224340.JPG');
[rows, columns, ¬] = size(img);

% get which one is smaller
if rows > columns
    length = columns;
else
    length = rows;
end

% crop the image to make sure row and column are same
img = imcrop(img,[1 1 length length]);

% rescale the image to size of 1024 x 1024
imgScaled = imresize(img, [1024, 1024]);
```

After these performance, /textitdisplay the resized image 2.

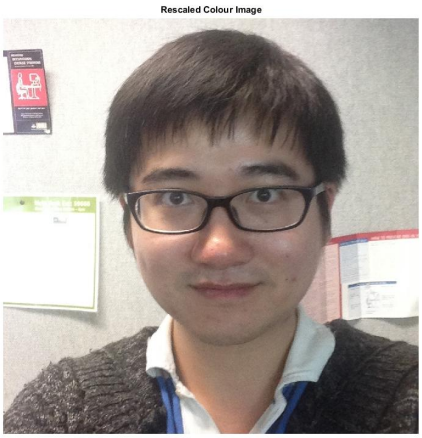

Figure 2: Resized Picture

After that, I get this image's RGB value by matrix row number, then display them.

```
% 5. convert the colour image into three grayscale channel
R = imgScaled(:,:,1);
G = imgScaled(:,:,2);
```

```
B = imgScaled(:,:,3);

% display the three channel greyscale images separately%
figure(2);
subplot(1,3,1), imshow(R), title('Grayscale image on R');
subplot(1,3,2), imshow(G), title('Grayscale image on G');
subplot(1,3,3), imshow(B), title('Grayscale image on B');
```

The following is the three RGB gray figures 3:

Figure 3: RGB Gray Figures

The following step is compute the 3 histograms for each of the grayscale images, and display the 3 histograms, I used *imhist(image, Nbins)* to finish it. In the method, *Nbins* is the *scalar* of this histograms. I set Nbins as 50 to suit data size.

```
% 6. Compute the 3 histograms of these grayscale images
bins = 50;

[Rcounts, RbinPos] = imhist(R, bins);
[Gcounts, GbinPos] = imhist(G, bins);
[Bcounts, BbinPos] = imhist(B, bins);

%display them
figure(3);
subplot(1,3,1), stem(RbinPos,Rcounts), title('Histograms ...
    on R');
subplot(1,3,2), stem(GbinPos,Gcounts), title('Histograms ...
    on G');
```

```
12 subplot(1,3,3), stem(BbinPos,Bcounts), title('Histograms ...
       on B');
```

This is the histogram result 4.

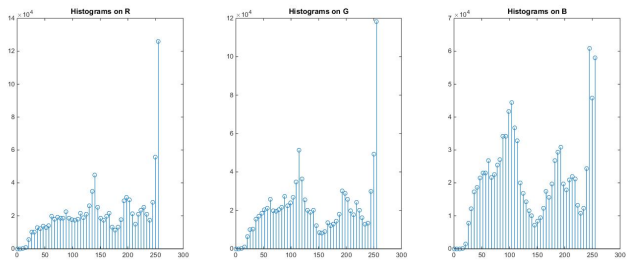

Figure 4: RGB Histogram Result

Finally, I apply "histogram equalization" to the original image, display histograms result.

```
1  REq = histeq(R, bins);
2  GEq = histeq(G, bins);
3  BEq = histeq(B, bins);
4
5  % then repeat the above step 6
6  [REqcounts, REqbinPos] = imhist(REq, bins);
7  [GEqcounts, GEqbinPos] = imhist(GEq, bins);
8  [BEqcounts, BEqbinPos] = imhist(BEq, bins);
9
10 figure(4);
11 subplot(1,3,1), stem(REqbinPos,REqcounts), ...
       title('Histograms on Equalized R');
12 subplot(1,3,2), stem(GEqbinPos,GEqcounts), ...
       title('Histograms on Equalized G');
13 subplot(1,3,3), stem(BEqbinPos,BEqcounts), ...
       title('Histograms on Equalized B');
```

The following is the equalized result 5:

2 Task-2: Morphology

The first task is load "text.png" and resize it to 1024×1024, and convert it to greyscale image I by *rgb2gray()* function.

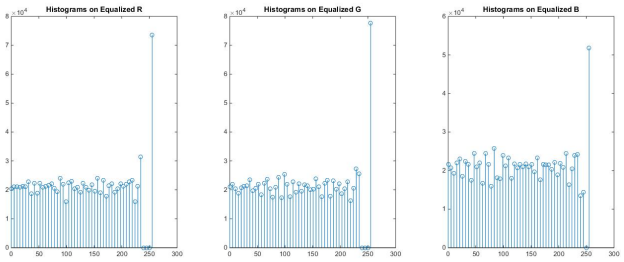

Figure 5: Equalized RGB Histogram Result

```
% Load in "text.png"
img = imread('text.png');

% resize it to size of 1024x1024
imgScaled = imresize(img, [1024, 1024]);

% 2. display the histogram of img
I = rgb2gray(imgScaled);
```

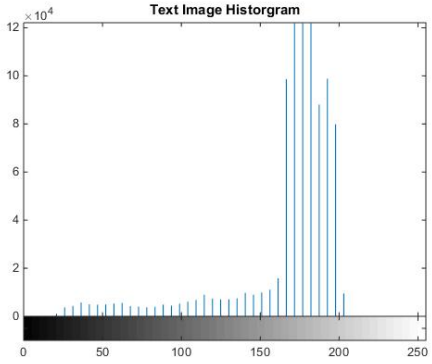

Figure 6: Text Histogram Result

After this, I also set the *Nbins* to 50 to display this grayscale image's histogram result. The following is the histogram result 6:

Then, I set *threshold* to 0.6, and process image by *im2bw()*. This action convert image to binary. In order to get the white pixel and block pixel number, I perform two loop on row index and column index on the binary

image, once we face a pixel value is 0, I add one to white pixel number, else, add one to black pixel number.

```
% 4. count the number of white and black pixels
whitePixelNum = 0;
blackPixelNum = 0;

for row = 1 : size(imgThr,1)
    for col = 1 : size(imgThr,2)
        pix = imgThr(row, col);

            whitePixelNum = whitePixelNum + 1;
        else
            blackPixelNum = blackPixelNum + 1;
        end

    end
end

fprintf('There are %d white pixels in this image.\n', ...
    whitePixelNum);
fprintf('There are %d black pixels in this image.\n', ...
    blackPixelNum);
```

After that, I constructed a structuring element by function *strel()*. The **SE** (structuring element) shape is 'disk' with radius is 20. Then, I used Matlab function *imerode*, *imdilate*, *imopen* and *imclose* to apply morphological operators of "erosion", "dilation", "opening" and "closing" to the binary image. The following figure is the result 7:

The last sub-task is design and program a morphology-based algorithm that can automatically segment each of the text lines from the image. In my mind, in order to get the textline, we should get this text area by connected components analysis. Before this, we should assign the one textline with same colour. According to morphology result, we can notice the opening and closing result's difference can get the outline of a textline 8. Then, I get difference by *imabsdiff()* function.

We can notice that the second line, third line and fifth line still exist gap between right part and left part. In order to link them, I apply dilation operation with a 'line' shape **SE**. I also using *strel()* with 'line', length is 70 and degree is 0. Then, get the result 9

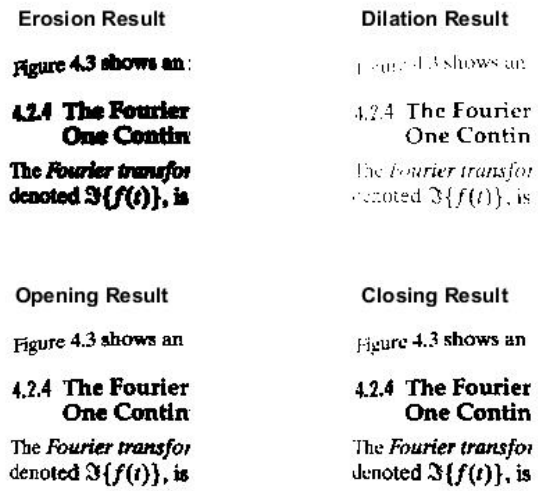

Figure 7: Four Operation On Binary Image

Then, I used connected components analysis to get these five block areas. These five struct have attribute on **boundingBoxs**. I used these points in **boundingBoxs** to cut sub-figure which represent to one textline.

```
CC = bwconncomp(resultFigure);
textLineNum = CC.NumObjects;
s = regionprops(CC,'BoundingBox');
boundingBoxs = cat(1, s.BoundingBox);

for textLineIndex = 1 : textLineNum

    testLine = boundingBoxs(textLineIndex, :);
    a = ceil(testLine(1));
    b = ceil(testLine(2));
    c = ceil(testLine(3));
    d = ceil(testLine(4));
    imgCrop = imcrop(imgScaled,[a b c d]);

    figure(3 + textLineIndex);
    imshow(imgCrop);

end
```

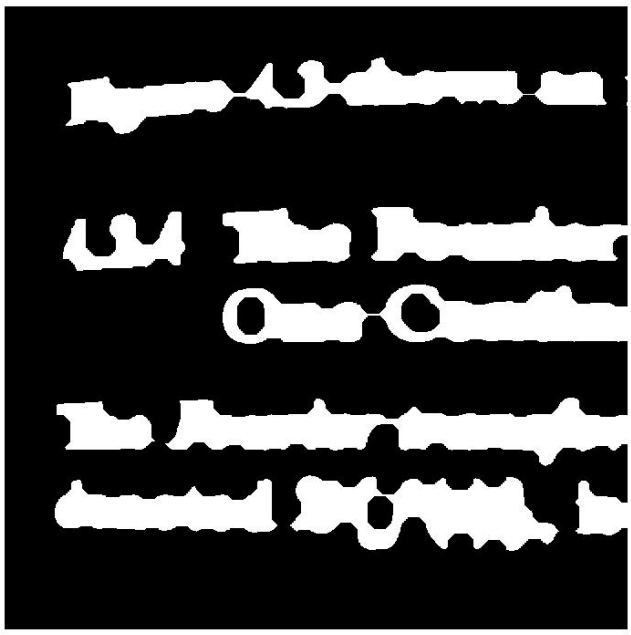

Figure 8: Difference between Closing and Opening

In order to save space, I did not put these five text line images in report. At the same time, the above analysis is based on image $'text.png'$. For the other two figures $'text2.png'$ and $'text3.png'$. I only saved threshold and **SE** information here, which can be tested by reader. For $text2.png$, assign *threshold* to 0.55, first **SE** to 'disk', radius is 4, and second **SE** to 'line', length is 400, degree is 0. For the $text3.png$, I assign *threshold* to 0.3, first **SE** to 'disk', radius is 28, second **SE** is 'line', length is 300, degree is 0.

3 Task-3: Colour Name Recognition

In this task, we asked to read in one colour image, and convert it to three images: H,S,V. After that, display the H ,S,V images as grayscale images in a 2×2 panel. In this task, I would like to use *rgb2gray(img)* to get a matrix which save *img* H,S and V values information.

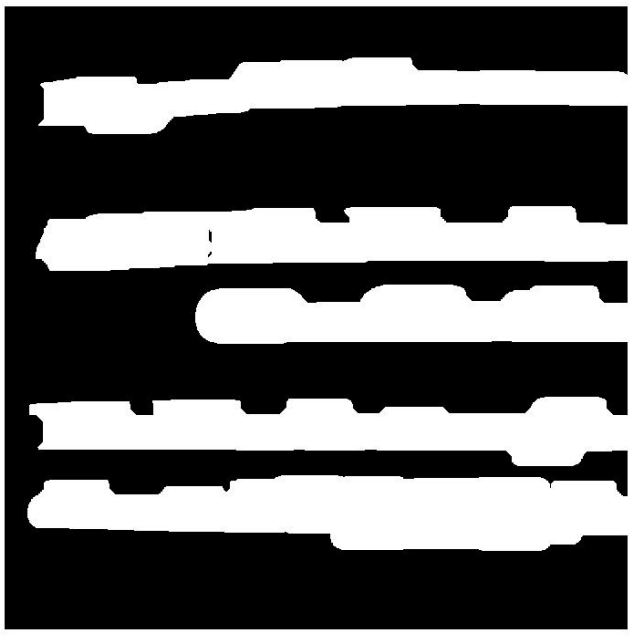

Figure 9: After Dilation by 'Line'

```
% read in the following colour image
img = imread('colourImage.png');
imgGray = rgb2gray(img);

% convert it to three images: H,S,V
imgHSV = rgb2hsv(img);

H = imgHSV(:,:,1);
S = imgHSV(:,:,2);
V = imgHSV(:,:,3);
```

The next figure is the HSV value figures displaying with original figure 10.

In order to the computed 12 H-values, and print next to these regions, I planned to use the connect *connectedcomponentsanalysis* approach to get these 12 regions in this image. However, after looked up the HSV grayscale images dispaly, we can easily notice that the *red* has the same H value with *gray* background. Furthermore, once we apply Matlab function *bw-*

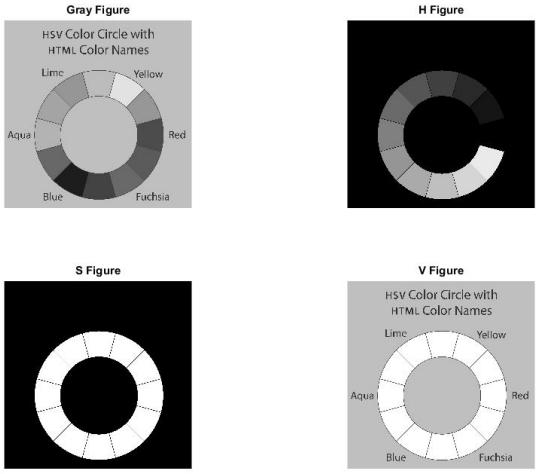

Figure 10: HSV Grayscale Images

conncomp() to get regions by inputting H image, we can only located only 11 connected components. Then, I used S image as input data for *bwconncomp()*, and get a list which save these 12 regions' information. Moreover, by using the Matlab function *regionprops* to process connect components, we can get these regions' attributes, such as 'centroid'. The following is the Matlab code:

```
CC = bwconncomp(S);
regionsNum = CC.NumObjects;
s = regionprops(CC,'Centroid');
centroids = cat(1, s.Centroid);
```

The next job is insert Hue value on these regions' centroids. I used function *insertText(I,position,numericvalue)*, where these *positions* are save in *centroids*. At same time, this insert function can also set the **inserted text**'s location and opacity. I also create a template named *inputImage* to this function's input data.

```
inputImage = img;

for regionIndex = 1 : regionsNum
```

```
4
5       region = CC.PixelIdxList{regionIndex};
6
7       % get the central of this region
8       X = round(centroids(regionIndex, 1));
9       Y = round(centroids(regionIndex, 2));
10      position = [X, Y];
11
12      % using this centroid to get H value in this region
13      HValue = H(Y, X);
14
15      % insert the H value
16      RGB = insertText(inputImage, position, HValue, ...
            'FontSize', 40, 'BoxOpacity', 0.0, 'AnchorPoint', ...
            'Center');
17
18      % save the inserted output
19      inputImage = RGB;
20
21  end
```

The following is the result of Hue value labeled colour image 11.

3.1 Method to Recognize Different Colours

According to the *colour* space in task, each one can be classified by its *Hue*, *Saturation* and *Lightness* values (Of course, we also have many other kinds of colour space to classified colour, such as RGB space). As a result, the colour can be thought as a multidimensional array of numbers or vectors. Once two pixel have the same HSV values, they are the same colour. Based on this principle, once give us a pixel, we just compare this pixel's H, S and V values with our HSV values set, we can know this pixel's colour.

In Matlab, we can use *containers.Map* to save all key-value pairs on colour information, named **colour map**. In this map, we could use one colour's H, S, V values as the key, e.g. [H, S, V]. Moreover, the colour name could act as the map's value. Then, once we meet a pixel, we just use its hue, saturation and lightness values constructed the key, and use this key to link to colour name in the **colour map**.

By using this simple approach, we can recognized difference colour in a image, and labeled them.

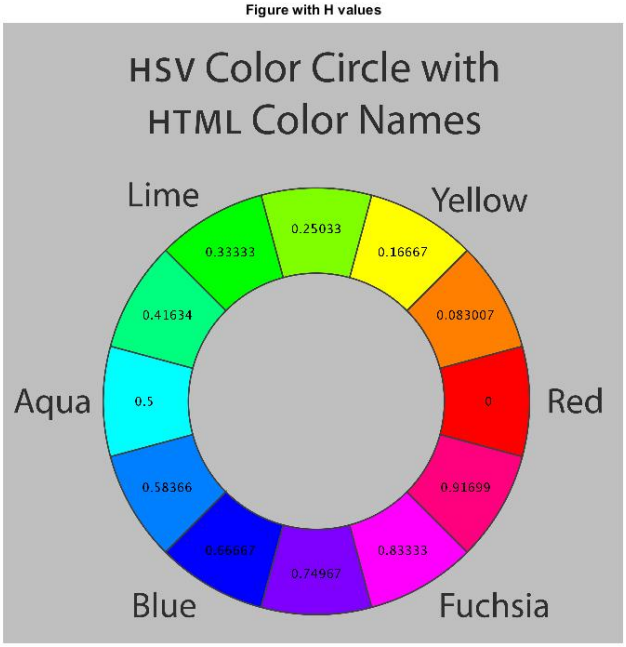

Figure 11: Labeled Colour Image

4 Task-4: Geometric Transformation

In this task, we had been asked to implement function for image rotation by an arbitrary angle ($0 \leqslant theta \leqslant 90$; use counter-clockwise rotation). I wrote code on get a arbitrary angle ($0 \leqslant angle \leqslant 90$).

```
% firstly, get the arbitrary angle
angle = randi([0, 90]);
```

In the **forward mapping** method on geometric transformation, we should use a *rotationMatrix*, which is:

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

I created the matrix by Matlab function *projective2d*.

```
% create the rotation matrix
tform = projective2d([cosd(angle) sind(angle) 0; ...
    -sind(angle) cosd(angle) 0; 0 0 1]);
```

Using this matrix to multiply source image's pixel position, to get the target image position. Then, assign source pixel's RGB value to target position's pixel.

Firstly, consider the target and source images have different size, it create a target image with different size from source one.

```
% create a new image which used to save created image, ...
    consider of the
% rotated image, the size of target image should bigger ...
    than source one
targetRows = ceil(rows*cosd(angle) + columns*sind(angle));
targetColumns = ceil(columns*cosd(angle) + rows*sind(angle));

%create target image
forwardTargetImg = uint8(zeros(targetRows , targetColumns ...
    , 3));
```

However, consider of some source pixels may have no corresponding on target image, we should *splat* this kind of pixel's value to its neighboring pixels. For instance, one pixel's target position is: $[12.2, 38.5]$, then we should assign this pixel's RGB value to these four positions on target image: $[12, 38]$, $[12+1, 38]$, $[12, 38+1]$ and $[12+1, 38+1]$.

```
% get the three R, G, B value from origianl image
        RGB = sourceImg(sourceRowIndex, sourceColumnIndex, :);

        [targetRowIndex, targetColumnIndex] = ...
            transformPointsForward(tform, sourceRowIndex, ...
            sourceColumnIndex);

        targetRowIndex = targetRowIndex + columns*sind(angle);

        % splatting the value to four points which around ...
            target position
        forwardTargetImg(targetRowIndex, ...
            targetColumnIndex, 1) = RGB(:,:,1);
```

```
10          forwardTargetImg(targetRowIndex, ...
               targetColumnIndex, 2) = RGB(:,:,2);
11          forwardTargetImg(targetRowIndex, ...
               targetColumnIndex, 3) = RGB(:,:,3);
```

In order to save space, the above code skip many details on process, such as assign G value to points, round target position. Please look at appendix for details.

In the **inverse mapping** method, we use the **inverse forward rotation matrix**. At start, we constructed the target image. Based on the target position on the target image, we use **inverse forward rotation matrix** to multiply with it to get source position. Once the source position is not located in the source figure, which means its value over the row or column range of source figure, we thought this pixel should be one **black** point. At the same time, the result on source position may located in four pixel points, then applied *linear interpolation method* to get the source position pixel value. In the two loops, I used *transformPointsInverse* to get the source position based on target position.

```
1  actualRowIndex = targetRowIndex;
2  actualRowIndex = actualRowIndex - columns*sind(angle);
3
4  [sourceRowIndex, sourceColumnIndex] = ...
      transformPointsInverse(tform,actualRowIndex, ...
      targetColumnIndex);
5
6  % before get value, we should make sure the source ...
      position in the
7  % source image (onece not in the source image, continue it)
8  if fix(sourceRowIndex) ≤ 0
9      continue
10 elseif fix(sourceRowIndex) ≥ rows
11     continue
12 elseif fix(sourceColumnIndex) ≤ 0
13     continue
14 elseif fix(sourceColumnIndex) ≥ columns
15     continue
16 end
17
18 % get their locations
19 sourcePosition = [sourceRowIndex, sourceColumnIndex];
20 targetPosition = [targetRowIndex, targetColumnIndex];
```

```
21
22 % perfrom linear interpolation appraoch based on my function
23 inverseTargetImg = linearInterpolation(sourceImg, ...
       inverseTargetImg, sourcePosition, targetPosition);
```

These codes also not contain all details, please look up on appendix for skipped parts.

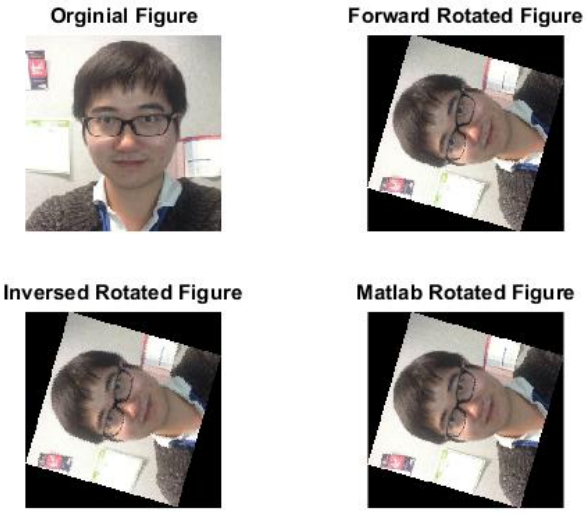

Figure 12: Rotated Figures

The following is the result on forward mapping method, inverse mapping method and matlab function 12 (the matlab apply *nearest* approach to assign pixel value, rotate angle is 72).

4.1 Compare Result With Matlab Version

In the Matlab function *imrotate()*, it provides 3 kinds of approach on assign pixel values. They are: *nearest*, *bilinear* and *bicubic*.

In my *rotate* method, the *forward mapping* one applies *splatting* approach, and the *inverse mapping* one applies **linear interpolation** approach. Consider of the *nearest* had already display below, The following are *biliear* and *bicubic* results 13 (rotate angle is 30).

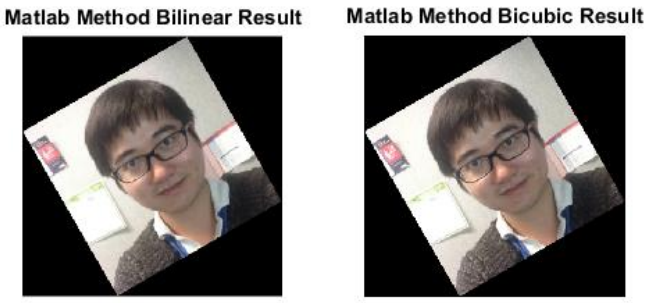

Figure 13: Matlab Biliear and Bicubic Rotate Result

When compare with the Matlab function on image rotation, the *imrotate()* results, my method output is rougher on forward mapping. Because on my forward mapping approach, I apply the **splatting** approach, which assign one pixel value to its target position's four neighbor pixel points. This situation makes each pixel 'bigger'. Then, this cause loss on pixel value by this *overwrite*. For the Matlab function, once it applies nearest point approach as default method, it did not loss as much as number of pixel points as forward one. For the other two approaches, the *biliear* will keep the 'accurate' pixel number on assign pixel values, because it will make sure each target position pixel has its own source position. For the *bicubic*, it can provide pixel value which outside the source figure. This approach also make sure each target position pixel had its own source position pixel. As a result, that is why *forward mapping* approach a bit 'rough' when compare with Matlab results.

However, for the *inverse mapping* result is as good as Matlab method. The *inverse mapping*, we use target figure pixel position to get its source position. Because I applied *linear interpolation* to assist to get target position's accurate value. This action will not loss pixel number. That is why it looked as good as Matlab results.

5 Task-5: Projective Transformation

In the job on implement a 8-parameter 2D-projective transformation, we should use Matlab function *projective2d* to construct transformation matrix, and *imwarp* to transform image. In the process on construction transformation matrix, we provide 8 parameter as the a_1, to a_8.

It has similar approach on forward mapping and inverse mapping methods. However, by different transition matrix. I used the following matrix as the 8 parameters.

$$\begin{bmatrix} \cos\theta & \sin\theta & 0.001 \\ -\sin\theta & \cos\theta & 0.01 \\ 0 & 0 & 1 \end{bmatrix}$$

I also used *projective2d()* to create matrix.

```
% get the transformation matrix struct
theta = 10;
tform = projective2d([cosd(theta) -sind(theta) 0.001; ...
    sind(theta) cosd(theta) 0.01; 0 0 1]);

% perform projective transformation by Matlab function
matlabImage = imwarp(sourceImg, tform);
```

After constructed transform matrix, I perform the similar approach with $Task-4$. Please read appendix to see details. There is only one difference on $Task-4$, I used a method which wrote by myself named *getPureFigure* to make the result match to a rectangle without useless area. This will impact on image size, and I will explain it on the next section. The final result is following 14 (Matlab method use default *nearest* approach on interpolation):

5.1 Compare Result With Matlab Version

In the Matlab method *imwarp()*, it also has same 3 approaches to interpolation as *imrotate()* one. This 15 is *biliear* and *bicubic* result.

An important difference between Matlab result and mine is the result image size. Mine target image is smaller than Matlab one. Because I used *round* to process pixel position. Once the position number rounded to 0 or its original range number, in the following process, they will be skipped, and had been assign to black pixel. Moreover, on the final process, the method

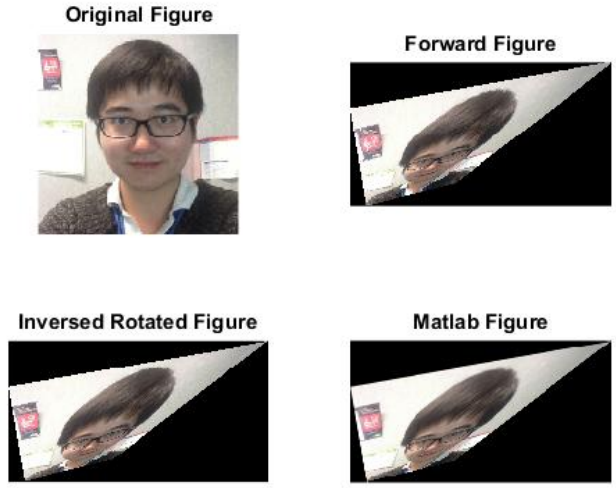

Figure 14: Projective Transformation Figures

getPureFigure also reduce these lines to get *figure* part, and that is why my results are smaller than Matlab one.

Moreover, the *forward mapping* result also had the same drawback as angle rotate result. It looks a bit rougher than *inverse mapping* and Matlab results. This is because I also apply *splatting* approach, and this will *loss* pixel by *overwrite* some pixel values. For the *inverse mapping*, its *linear interpolation* approach worked well and is same to Matlab result.

Another important difference is the transformation will left many 'empty' area, and the *imwarp()* also has some method to fill these empty area. The parameter *FillValues* will fill these area with its following pixel value. At same time, the parameter *OutputView* will make output image located in world coordinate system. These two features are not included in my functions. But if I change the *inverse mapping* method make a decision on source position is located in original image range, replace the *continue* to a special pixel value to RGB value, the method can do the same feature on *FillValue*.

6 Appendix: Matlab Code

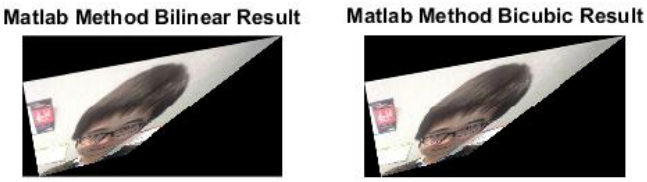

Figure 15: Matlab Biliear and Bicubic Result

```
% for task-1: basic I/O image

% clean un-relatived items
clc
clear

% 1. Take a frontal face photo of yourself
% 2. Save this photo into an image file  named "photo_UId.jpg"

% 3. Read image
img = imread('photo_U5224340.JPG');
[rows, columns, ¬] = size(img);

% get which one is smaller
if rows > columns
    length = columns;
else
    length = rows;
end

% crop the image to make sure row and column are same
img = imcrop(img,[1 1 length length]);
```

```
23
24 % rescale the image to size of 1024 x 1024
25 imgScaled = imresize(img, [1024, 1024]);
26
27 % 4. display this rescaled colour image on screen
28 figure(1), imshow(imgScaled),title('Rescaled Colour Image');
29
30 % 5. convert the colour image into three grayscale channel
31 R = imgScaled(:,:,1);
32 G = imgScaled(:,:,2);
33 B = imgScaled(:,:,3);
34
35 % display the three channel greyscale images separately%
36 figure(2);
37 subplot(1,3,1), imshow(R), title('Grayscale image on R');
38 subplot(1,3,2), imshow(G), title('Grayscale image on G');
39 subplot(1,3,3), imshow(B), title('Grayscale image on B');
40
41
42 % 6. Compute the 3 histograms of these grayscale images
43 bins = 50;
44
45 [Rcounts, RbinPos] = imhist(R, bins);
46 [Gcounts, GbinPos] = imhist(G, bins);
47 [Bcounts, BbinPos] = imhist(B, bins);
48
49 %display them
50 figure(3);
51 subplot(1,3,1), stem(RbinPos,Rcounts), title('Histograms ...
      on R');
52 subplot(1,3,2), stem(GbinPos,Gcounts), title('Histograms ...
      on G');
53 subplot(1,3,3), stem(BbinPos,Bcounts), title('Histograms ...
      on B');
54
55 % 7. Apply histogram equalization
56 REq = histeq(R, bins);
57 GEq = histeq(G, bins);
58 BEq = histeq(B, bins);
59
60 % then repeat the above step 6
61 [REqcounts, REqbinPos] = imhist(REq, bins);
62 [GEqcounts, GEqbinPos] = imhist(GEq, bins);
63 [BEqcounts, BEqbinPos] = imhist(BEq, bins);
64
```

```
figure(4);
subplot(1,3,1), stem(REqbinPos,REqcounts), ...
    title('Histograms on Equalized R');
subplot(1,3,2), stem(GEqbinPos,GEqcounts), ...
    title('Histograms on Equalized G');
subplot(1,3,3), stem(BEqbinPos,BEqcounts), ...
    title('Histograms on Equalized B');
```

```
% for task 2: Morphology

% clean un-relatived items
clc
clear

% Load in "text.png"
img = imread('text3.png');

% resize it to size of 1024x1024
img = imresize(img, [1024, 1024]);

% 2. display the histogram of img
I = rgb2gray(img);
bins = 50;

figure(1), imhist(I, bins), title('Text Image Historgram');

% 3. threshold the histogram to obtain a binary image

% firstly, get the threshold from build-in method: graythresh
% http://au.mathworks.com/help/images/ref/graythresh.html
% threshold = graythresh(imgScaled);

threshold =0.3;
imgThr = im2bw(img, threshold);

% 4. count the number of white and black pixels
whitePixelNum = 0;
blackPixelNum = 0;

for row = 1 : size(imgThr,1)
    for col = 1 : size(imgThr,2)
        pix = imgThr(row, col);

        % consider of in the imgThr, there are only two ...
```

```
            kinds of pix value
        if pix == 0
            whitePixelNum = whitePixelNum + 1;
        else
            blackPixelNum = blackPixelNum + 1;
        end

    end
end

fprintf('There are %d white pixels in this image.\n', ...
    whitePixelNum);
fprintf('There are %d black pixels in this image.\n', ...
    blackPixelNum);

% 5. testing the effects of applying morphological ...
    operators of "erosion",
% "dilation", "opening" and "closing" to the binary image,
% and visually inspect the effects

% first of all, get the Structuring Element, get it by
% function 'strel': ...
    http://au.mathworks.com/help/images/ref/strel.html
SE = strel('disk',28);

erosion = imerode(imgThr, SE);
dilation = imdilate(imgThr, SE);
opening = imopen(imgThr, SE);
closing = imclose(imgThr, SE);

figure(2);
subplot(2,2,1), imshow(erosion), title('Erosion Result');
subplot(2,2,2), imshow(dilation), title('Dilation Result');
subplot(2,2,3), imshow(opening), title('Opening Result');
subplot(2,2,4), imshow(closing), title('Closing Result');

% 6. Design and program a morphology-based algorithm that ...
    can automatically
% segment each of the text lines from the image. In other ...
    words, input the
% particular given image "text.png",  your algorithm will ...
    automatically
% extract five sub-images containing the five text lines.

% based on the paper from Pratheeba (2010 ,et al), we ...
```

```
    perform closing,
% openning, then difference two figures, finnally, perform ...
    cca, to get the
% subfigures.
opening = imopen(imgThr, SE);
closing = imclose(imgThr, SE);

difference = imabsdiff(closing, opening);

SE = strel('line', 300, 0);
resultFigure = imdilate(difference, SE);

CC = bwconncomp(resultFigure);
textLineNum = CC.NumObjects;
s = regionprops(CC,'BoundingBox');
boundingBoxs = cat(1, s.BoundingBox);

figure(3),
imshow(resultFigure);

for textLineIndex = 1 : textLineNum

    testLine = boundingBoxs(textLineIndex, :);
    a = ceil(testLine(1));
    b = ceil(testLine(2));
    c = ceil(testLine(3));
    d = ceil(testLine(4));
    imgCrop = imcrop(img,[a b c d]);

    figure(3 + textLineIndex);
    imshow(imgCrop);

end
```

```
% for task-3: colour name recognition

% clean un-relatived items
clc
clear

% read in the following colour image
img = imread('colourImage.png');
imgGray = rgb2gray(img);

```

```
% convert it to three images: H,S,V
imgHSV = rgb2hsv(img);

H = imgHSV(:,:,1);
S = imgHSV(:,:,2);
V = imgHSV(:,:,3);

% display the H,S,V images as greyscale images in a 2 x 2 ...
    panel
figure(1);
subplot(2,2,1), imshow(imgGray), title('Gray Figure');
subplot(2,2,2), imshow(H), title('H Figure');
subplot(2,2,3), imshow(S), title('S Figure');
subplot(2,2,4), imshow(V), title('V Figure');

% print the computed 12 H-values next to the 12 regions in ...
    the image
% consider of the background colour hue value and 12 hue ...
    value, apply
% saturation to get connect components (the same ...
    coordinate position in H S)
CC = bwconncomp(S);
regionsNum = CC.NumObjects;
s = regionprops(CC,'Centroid');
centroids = cat(1, s.Centroid);

inputImage = img;

for regionIndex = 1 : regionsNum

    region = CC.PixelIdxList{regionIndex};

    % get the central of this region
    X = round(centroids(regionIndex, 1));
    Y = round(centroids(regionIndex, 2));
    position = [X, Y];

    % using this centroid to get H value in this region
    HValue = H(Y, X);

    % insert the H value
    RGB = insertText(inputImage, position, HValue, ...
        'FontSize', 40, 'BoxOpacity', 0.0, 'AnchorPoint', ...
        'Center');

```

```
    % save the inserted output
    inputImage = RGB;

end

figure(2), imshow(RGB), title('Figure with H values');
```

```
% for task-4: geometric transformation

% clean un-relatived items
clc
clear

% 1. implement your own Matlab function for image rotation ...
    by an arbitrary
% angle 0=< theta = <90; (use counter-clockwise rotation)

% this part is for the forward mapping approach

% firstly, get the arbitrary angle
angle = randi([0, 90]);
% angle = 30;

% then, for each pixel in origianl image, act as source, ...
    get its target
sourceImg = imread('photo_U5224340.JPG');

% then, save the information of source image to rows, ...
    columns and channels
% and reduce the image size to save process time
[rows, columns, ¬] = size(sourceImg);
sourceImg = imresize(sourceImg, [rows/4,columns/4]);

[rows, columns, channels] = size(sourceImg);

% create a new image which used to save created image, ...
    consider of the
% rotated image, the size of target image should bigger ...
    than source one
targetRows = ceil(rows*cosd(angle) + columns*sind(angle));
targetColumns = ceil(columns*cosd(angle) + rows*sind(angle));

%create target image
forwardTargetImg = uint8(zeros(targetRows , targetColumns ...
```

```
    , 3));

% create the rotation matrix
tform = projective2d([cosd(angle) sind(angle) 0; ...
    -sind(angle) cosd(angle) 0; 0 0 1]);

% rotationMatrix = [cosd(angle) -sind(angle); sind(angle) ...
    cosd(angle)];
% rotationMatrix = tform.T;

% create rotated image by Matlab
matlabImage = imrotate(sourceImg, angle);

for sourceRowIndex = 1: rows
    for sourceColumnIndex = 1 : columns

        % get the three R, G, B value from origianl image
        RGB = sourceImg(sourceRowIndex, sourceColumnIndex, :);

        [targetRowIndex, targetColumnIndex] = ...
            transformPointsForward(tform, sourceRowIndex, ...
            sourceColumnIndex);

        % consider of the difference on sizes of source ...
            image and target image,
        % the target image's position value is not equal ...
            to source one to avoid
        % negative situation. Because its anti-clock ...
            rotation, then modify
        % column to match to the target image position
        targetRowIndex = targetRowIndex + columns*sind(angle);

        forwardTargetImg = setSplatting(forwardTargetImg, ...
            targetRowIndex, targetColumnIndex, RGB);

    end
end

% This part is for inverse appraoch

% create inverse image
inverseTargetImg = uint8(zeros(targetRows , targetColumns ...
    , 3));

% then, perform inverse mapping approach
```

```
for targetRowIndex = 1 : targetRows
    for targetColumnIndex = 1 : targetColumns

        actualRowIndex = targetRowIndex;
        actualRowIndex = actualRowIndex - columns*sind(angle);

        [sourceRowIndex, sourceColumnIndex] = ...
            transformPointsInverse(tform,actualRowIndex, ...
            targetColumnIndex);

        % before get value, we should make sure the source ...
            position in the
        % source image (onece not in the source image, ...
            continue it)
        if fix(sourceRowIndex) ≤ 0
            continue
        elseif fix(sourceRowIndex) ≥ rows
            continue
        elseif fix(sourceColumnIndex) ≤ 0
            continue
        elseif fix(sourceColumnIndex) ≥ columns
            continue
        end

        % get their locations
        sourcePosition = [sourceRowIndex, sourceColumnIndex];
        targetPosition = [targetRowIndex, targetColumnIndex];

        % perfrom linear interpolation appraoch based on ...
            my function
        inverseTargetImg = linearInterpolation(sourceImg, ...
            inverseTargetImg, sourcePosition, targetPosition);

    end
end

figure(1);
subplot(2,2,1), imshow(sourceImg), title('Orginial Figure');
subplot(2,2,2), imshow(forwardTargetImg), title('Forward ...
    Rotated Figure');
subplot(2,2,3), imshow(inverseTargetImg), title('Inversed ...
    Rotated Figure');
subplot(2,2,4), imshow(matlabImage), title('Matlab Rotated ...
    Figure');

```

```
104 fprintf('There rotated angle is %d.\n', angle);
```

```
1  function [ img ] = setSplatting( img, columnIndex, ...
       rowIndex, RGB )
2  % This method used to process an image hole by splatting ...
       appraoch
3  %   The splatting appraoch assign one point's around ...
       points has same value
4
5      columnIndex = fix(columnIndex);
6      rowIndex = fix(rowIndex);
7
8      if columnIndex == 0
9         columnIndex = 1;
10     end
11
12     if rowIndex == 0
13        rowIndex = 1;
14     end
15
16     img(columnIndex, rowIndex, 1) = RGB(:,:,1);
17     img(columnIndex, rowIndex, 2) = RGB(:,:,2);
18     img(columnIndex, rowIndex, 3) = RGB(:,:,3);
19
20     img(columnIndex + 1, rowIndex, 1) = RGB(:,:,1);
21     img(columnIndex + 1, rowIndex, 2) = RGB(:,:,2);
22     img(columnIndex + 1, rowIndex, 3) = RGB(:,:,3);
23
24     img(columnIndex, rowIndex + 1, 1) = RGB(:,:,1);
25     img(columnIndex, rowIndex + 1, 2) = RGB(:,:,2);
26     img(columnIndex, rowIndex + 1, 3) = RGB(:,:,3);
27
28     img(columnIndex + 1, rowIndex + 1, 1) = RGB(:,:,1);
29     img(columnIndex + 1, rowIndex + 1, 2) = RGB(:,:,2);
30     img(columnIndex + 1, rowIndex + 1, 3) = RGB(:,:,3);
31
32
33 end
```

```
1  function [ inverseTargetImg ] = linearInterpolation( ...
       sourceImg, inverseTargetImg, sourcePosition, ...
       targetPosition )
```

```
2  %   This method used to perform linear interpolation appraoch
3
4      % consider of the source position in the source image may
5      % not exists (not int position), we should apply linear
6      % interpolation method
7
8      sourceRowIndex = sourcePosition(1);
9      sourceColumnIndex = sourcePosition(2);
10
11     targetRowIndex = targetPosition(1);
12     targetColumnIndex = targetPosition(2);
13
14     % firstly, we get the four point around source position.
15     ltRGB = sourceImg(fix(sourceRowIndex), ...
           fix(sourceColumnIndex), :); % left top point
16     ldRGB = sourceImg(fix(sourceRowIndex) + 1, ...
           fix(sourceColumnIndex), :); % left down point
17     rtRGB = sourceImg(fix(sourceRowIndex), ...
           fix(sourceColumnIndex) + 1, :); % right top point
18     rdRGB = sourceImg(fix(sourceRowIndex) + 1, ...
           fix(sourceColumnIndex) + 1, :); % right down point
19
20     % get the weightings on rows and column line
21     m = sourceRowIndex - fix(sourceRowIndex);
22     n = sourceColumnIndex - fix(sourceColumnIndex);
23
24     % get R value
25     temp1 = m*ltRGB(:,:,1) + (1 - m)*ldRGB(:,:,1);
26     temp2 = m*rtRGB(:,:,1) + (1 - m)*rdRGB(:,:,1);
27     temp3 = n*temp1 + (1 - n)*temp2;
28     inverseTargetImg(targetRowIndex, targetColumnIndex, 1) ...
           = temp3;
29
30     % get G value
31     temp1 = m*ltRGB(:,:,2) + (1 - m)*ldRGB(:,:,2);
32     temp2 = m*rtRGB(:,:,2) + (1 - m)*rdRGB(:,:,2);
33     temp3 = n*temp1 + (1 - n)*temp2;
34     inverseTargetImg(targetRowIndex, targetColumnIndex, 2) ...
           = temp3;
35
36      % get B value
37      temp1 = m*ltRGB(:,:,3) + (1 - m)*ldRGB(:,:,3);
38      temp2 = m*rtRGB(:,:,3) + (1 - m)*rdRGB(:,:,3);
39      temp3 = n*temp1 + (1 - n)*temp2;
40     inverseTargetImg(targetRowIndex, targetColumnIndex, 3) ...
```

```
        = temp3;

end
```

```
% for task 5: Projective transformation

% clean un-relatived items
clc
clear

% read image
sourceImg = imread('photo_U5224340.JPG');

% reduce the size of source image to save processing time
sourceImg = imresize(sourceImg, 0.25);
[rows, columns, ¬] = size(sourceImg);

% get the transformation matrix struct
theta = 10;
tform = projective2d([cosd(theta) -sind(theta) 0.001; ...
    sind(theta) cosd(theta) 0.01; 0 0 1]);

% perform projective transformation by Matlab function
matlabImage = imwarp(sourceImg, tform);

% create an empty image for output
forwardTargetImg = uint8(zeros(size(sourceImg)));

% forward mapping approach
for sourceRowIndex = 1 : rows
    for sourceColumnIndex = 1 : columns

        % get the source pixel RGB values
        RGB = sourceImg(sourceColumnIndex, sourceRowIndex, :);

        % get the target position
        [targetRowIndex, targetColumnIndex] = ...
            transformPointsForward(tform, sourceRowIndex, ...
            sourceColumnIndex);
        targetColumnIndex = targetColumnIndex + ...
            rows*sind(theta);

        forwardTargetImg = setSplatting(forwardTargetImg, ...
            targetColumnIndex, targetRowIndex, RGB);
```

```
36
37      end
38  end
39
40  [targetRows, targetColumns, channels] = ...
        size(forwardTargetImg);
41
42  % inverse mapping approach
43  inverseTargetImg = uint8(zeros(size(forwardTargetImg)));
44
45  for targetRowIndex = 1 : targetRows
46      for targetColumnIndex = 1 : targetColumns
47
48          actualColumnIndex = targetColumnIndex;
49          actualColumnIndex = actualColumnIndex - ...
                rows*sind(theta);
50
51          [sourceRowIndex, sourceColumnIndex] = ...
                transformPointsInverse(tform, targetRowIndex, ...
                actualColumnIndex);
52
53          % before get value, we should make sure the source ...
                position in the
54          % source image (onece not in the source image, ...
                continue it)
55          sourceRowIndex = round(sourceRowIndex);
56          sourceColumnIndex = round(sourceColumnIndex);
57
58          if sourceRowIndex ≤ 0
59              continue
60          elseif sourceRowIndex ≥ rows
61              continue
62          elseif sourceColumnIndex ≤ 0
63              continue
64          elseif sourceColumnIndex ≥ columns
65              continue
66          end
67
68          % get their locations
69          sourcePosition = [sourceColumnIndex, sourceRowIndex];
70          targetPosition = [targetColumnIndex, targetRowIndex];
71
72          % perfrom linear interpolation appraoch based on ...
                my function
73          inverseTargetImg = linearInterpolation(sourceImg, ...
```

```
            inverseTargetImg, sourcePosition, targetPosition);

    end
end

% Then, deal with these two image, to reduce their size to ...
    regular as
% Matlab

% set the threhold as small as possible, only keep the not ...
    black part
threshold =0.1;

% reduce the black areas
forwardTargetImg = getPureFigure(forwardTargetImg, threshold);
inverseTargetImg = getPureFigure(inverseTargetImg, threshold);

figure(1);
subplot(2,2,1), imshow(sourceImg), title('Original Figure');
subplot(2,2,2), imshow(forwardTargetImg), title('Forward ...
    Figure');
subplot(2,2,3), imshow(inverseTargetImg), title('Inversed ...
    Rotated Figure');
subplot(2,2,4), imshow(matlabImage), title('Matlab Figure');
```

```
function [ reducedFigure ] = getPureFigure( img, threshold )
% This method used to get the 'pure figure' from a figure ...
    with many black
%area
% e.g. img (3/4 area are black) => reducedImg (left ...
    non-black part)

imgThr = im2bw(img, threshold);

CC = bwconncomp(imgThr);

s = regionprops(CC,'BoundingBox');
boundingBoxs = cat(1, s.BoundingBox);

BB = boundingBoxs(1, :);
a1 = ceil(BB(1));
b1 = ceil(BB(2));
c1 = ceil(BB(3));
d1 = ceil(BB(4));
```

```
18
19 reducedFigure = imcrop(img,[a1 b1 c1 d1]);
20
21 end
```