Lab 3: Colorizing the Prokudin-Gorskii photo collection Report

CSE 568: Robotic Algorithms
Lunshu Sun
Email: lunshusu@buffalo.edu

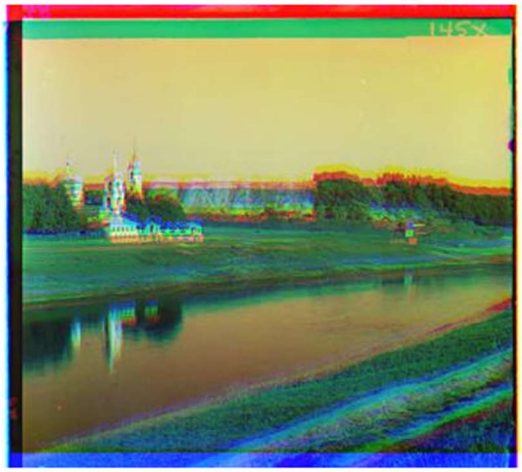
Goal: We will implement basic algorithms like SSD NCC and RANSAC (harris conner detection) to detect features to an alignment RGB image.

First, we will split the image into three pieces, using the method in piazza lab3 guidance
https://piazza.com/class/ksza7r6rp5t7a2?cid=380
The floor() function: get int

```
    fullimg = imread(strcat('image',int2str(imageIndex),'.jpg'));
    [row,column] = size(fullimg);

% Split the image and set each row and column for each one as [row/3,column
% size], different picture has different size

    blue = fullimg(1:floor(row/3),:);
%   size(blue)
    green = fullimg(floor(row/3) + 1:(2*floor(row/3)),:);
%   size(green)
    red = fullimg((2*floor(row/3))+2:row,:);
%   size(red)
```

I use the size() function to read the row and column, and split the image than I merge the image with this code

```
% Create an RGB image for each one

  rgbimage(:,:,1) = blue;
  rgbimage(:,:,2) = green;
  rgbimage(:,:,3) = red;
  imwrite(rgbimage, strcat('image',int2str(imageIndex),'-color.jpg'));
```



The first image is like this after alignment. There are shifts between these RGB images.
Add clear at the bottom line is also very important otherwise we can not output 6 images.

Assignment1-SSD

I will use SSD to calculate the shift, just find the smallest distance between two images like blue and green, before that, I cut the edges from every three images to make the picture looks cleaner.

and take the displacement with the best score. There is a number of possible metrics that one could use to score how well the images match. The simplest one is just the L2 norm also known as the Sum of Squared Differences (SSD) distance which is simply sum(sum((image1-image2)^2)). Another is normalized cross-correlation (NCC), which is simply a dot product between two normalized vectors: (image1./|image1| and image2./|image2|). See the Matlab/Octave function `normxcorr2`.

Just as the instructions on lab3
Calculate the distance between two images and sum twice to get a scale value, so we will find the smallest value and its coordinate, in this place. I search with a window and the size is set to [-15, 15], I tried [-20,20] here, it cost more time and the range of also in -15 and 15.

For SSD, the smaller the better

```
for i = -15:15
  for j = -15:15
    g_ssd = g(edge_cut+i:px-edge_cut+i , edge_cut+j:py-edge_cut+j);
    r_ssd = r(edge_cut+i:px-edge_cut+i , edge_cut+j:py-edge_cut+j);

    offg = (b_withoutedge - g_ssd).^2;
    offr = (b_withoutedge - r_ssd).^2;

    temp_ssdg = sum(sum(offg));
    temp_ssdr = sum(sum(offr));

    if temp_ssdg <= ssdg
      ssdg = temp_ssdg;
      green_x = i;
      green_y = j;
    end
```

Results: The performance is pretty good except for imge4

Assignment2-NCC

For NCC, just the same method as SSD, but for NCC, we know that the bigger the better so our judging criteria should be changed to find the biggest value as well as its shifts.

We use two for loop to search windows, and two for vector computing really cost a lot of time, especially using two for a loop.

> and take the displacement with the best score. There is a number of possible metrics that one could use to score how well the images match. The simplest one is just the L2 norm also known as the Sum of Squared Differences (SSD) distance which is simply sum(sum((image1-image2)^2)). Another is normalized cross-correlation (NCC), which is simply a dot product between two normalized vectors: (image1./|image1| and image2./|image2|). See the Matlab/Octave function `normxcorr2`.

Read the lab instruction, the formula is already given, but do not forget to add the sum() function before this formula. I forget to add sum and cost several times on this bug.

If you print this value, it will be a matrix, we should add them together, and then we can compare.

```
for i = -15:15
    for j = -15:15
        new_g = g(offset+i:px-offset+i , offset+j:py-offset+j);
        new_r = r(offset+i:px-offset+i , offset+j:py-offset+j);
        new_g = double(new_g);
        new_r = double(new_r);

        first = new_g./norm(new_g);
        second = b_withoutedge./norm(b_withoutedge);
        temp_cc_g = dot(first,second);
        temp_cc_g = sum(temp_cc_g);

        first_2 = new_r./norm(new_r);
        second_2 = b_withoutedge./norm(b_withoutedge);
        temp_cc_r = dot(first_2,second_2);
        temp_cc_r = sum(temp_cc_r);
```

The double function here is to convert values so that we can do the formula
I also write another NCC formula using python!

```
import statistics
def NCC(A,B):
    mean_A = statistics.mean(A)
    mean_B = statistics.mean(B)
    # print(mean_A, mean_B)
    upper = 0
    former = 0
    lower = 0
    for i in range(len(A)):
            up = (A[i]-mean_A)*(B[i]-mean_B)
            upper = up + upper
            dw = (A[i]-mean_A)**2
            former = former + dw
            lw = (B[i]-mean_B)**2
            lower = lw + lower

    downer = (former * lower)**(0.5)

    NCC = upper/downer
    # print("The NCC is :", NCC)
    return NCC

A1=[0, 0, 0, 72, 0, 84, 170, 26, 54]
A2=[75, 127, 52, 87, 86, 0, 12, 188, 176]
A3=[3, 9, 208, 1, 2, 6, 22, 40, 9]
B=[3, 10, 20, 18, 1, 5, 2, 30, 3]

print('The NCC of Point1:', NCC(A1,B))
print('The NCC of Point2:', NCC(A2,B))
print('The NCC of Point3:', NCC(A3,B))
```

```
The NCC of Point1: -0.24583843213618925
The NCC of Point2: 0.4067696375190906
The NCC of Point3: 0.4621932304987229
```

We can see that NCC is pretty good for all six images, but the time cost is almost 100 times bigger than SSD!

Shift and Time for each image compared two methods

```
Image1
SSD Green shift: [-5,-2]
SSD Red shift: [-11,-1]
The time is: 0.22499
NCC Green shift: [-5,-2]
NCC Red shift: [-10,-1]
The time is: 23.6592
Image2
SSD Green shift: [-4,-2]
SSD Red shift: [-10,-2]
The time is: 0.26348
NCC Green shift: [-4,-2]
NCC Red shift: [-10,-2]
The time is: 23.4944
Image3
SSD Green shift: [-7,-3]
SSD Red shift: [-15,-4]
The time is: 0.29424
NCC Green shift: [-7,-3]
NCC Red shift: [-15,-5]
The time is: 23.2534
```

```
Image4
SSD Green shift: [-6,14]
SSD Red shift: [-14,-1]
The time is: 0.27143
NCC Green shift: [-4,-1]
NCC Red shift: [-14,-1]
The time is: 24.8493
Image5
SSD Green shift: [-5,-2]
SSD Red shift: [-13,-4]
The time is: 0.33242
NCC Green shift: [-5,-3]
NCC Red shift: [-12,-4]
The time is: 23.8291
Image6
SSD Green shift: [0,0]
SSD Red shift: [-6,-1]
The time is: 0.26844
NCC Green shift: [0,0]
NCC Red shift: [-6,-1]
The time is: 23.8844
```

Assignment3 – RANSAC and Harris Corner detection

Reference: slides post on piazza

For harris corner I implement the code with this formula

Harris Corner Detection Algorithm

1. Compute x and y derivatives of image

$$I_x = G_\sigma^x * I \quad I_y = G_\sigma^y * I$$

2. Compute products of derivatives at every pixel

$$I_{x2} = I_x.I_x \quad I_{y2} = I_y.I_y \quad I_{xy} = I_x.I_y$$

3. Compute the sums of the products of derivatives at each pixel

$$S_{x2} = G_{\sigma\prime} * I_{x2} \quad S_{y2} = G_{\sigma\prime} * I_{y2} \quad S_{xy} = G_{\sigma\prime} * I_{xy}$$

4. Define at each pixel (x, y) the matrix

$$H(x,y) = \begin{bmatrix} S_{x2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y2}(x,y) \end{bmatrix}$$

5. Compute the response of the detector at each pixel

$$R = Det(H) - k(Trace(H))^2$$

6. Threshold on value of R. Compute nonmax suppression.

```
function [Corner_feature] = harris(img,threshold)
    % threshold = 1000000
    k = 0.04;
    sigma = 5;
    g = fspecial('Gaussian',[3 3],sigma);
    img1 = imfilter(img,g);
    img1 = double(img1);
    Gx = [-1 0 1;-2 0 2;-1 0 1];
    Gy = Gx';
    Ix = imfilter(img1,Gx);
    Iy = imfilter(img1,Gy);
    Ix2 = Ix .* Ix;
    Iy2 = Iy .* Iy;
    Ixy = Ix .* Iy;

    Sx2 = imfilter(Ix2,g);
    Sy2 = imfilter(Iy2,g);
    Sxy = imfilter(Ixy,g);
    for x = 1 : size(Ix2,1)
        for y = 1 : size(Iy2,2)
            M = [Sx2(x,y) Sxy(x,y);Sxy(x,y) Sy2(x,y)];
            R(x,y) = det(M) - k * (trace(M) ^ 2);
        end
    end
```

Its performance is very well on corner detection

For RANSAC, I'll randomly select two points from blue features and red features and set a window for them to calculate the distance between two points and will set a threshold as 1 pixel, if the distance is below 1 pixel, I will add them to the inliers and count the number. Calculate for 200 points from both blue and red, and blue and green, iteration for 1000 times to find the best fit (the maximum counter one is the best fit) as well as its corresponding shifts.

I have never used MATLAB before, I almost cost 4 days on the last RANSAC and HARRIS question, I only finish the harris.m algorithm because I'm not familiar with the MATLAB language, which cost a lot of time on debug and basic function use.

This lab3 is really a good question, though implementing coding, I deepened the formula of SSD, NCC, HARRIS CORNER, and RASAC, and learn and compare their performance and application. Thanks for the instructor team, I learned a lot from office hours.

I also write a help function to find the top 200 features as well as their coordinate

编辑器 - D:\OneDrive\buffalo\cse568\lab3\feature_find.m

main.m | harris.m | ransactest.m | ransac.m | maintest.m | feature_find.m

```
function [C] = feature_find(A,n)
 A = [1 2 3;4 5 6;7 8 9]
 n = 3 % Top 3 features from A, return its coordinate

    C = []; % C is x and y coordinate of these coordinates
    for k = 1:n
    max_value = max(max(A))
    [i,j]=find(A==max_value);
    C = [C;[i,j]];
    A(i,j) = -inf;
    end
end
```

命令行窗口

```
max_value =

     9


max_value =

     8


max_value =

     7


ans =

     3     3
     3     2
     3     1
```