# CSE 568 – Lab III

# Colorizing the Prokudin-Gorskii photo collection*

**Asish Kakumanu**
UB Person No : 50288695
asishkak@buffalo.edu

## Abstract

The Abstract of the project is to implement some basic image processing algorithms and colorizing the photo collection by Produkin-Gorskii and to align them correctly by calculating the SSD (Sum of Squared Differences) and NCC (Normalized Cross Correlation).

## 1 Overview of Data

He used an early color technology that involved recording three exposures of every scene onto a glass plate using a red, green, and blue filter. A digitized picture plates from his collection is displayed in **Figure. 1**


*Figure 1*

# 2 Implementation

## 2.1 Slicing Images

In this part, we slice the given images into three different channels which are Red, Blue and Green. To achieve this, we read the image and divide the whole image into three parts by calculating the dimensions of the image and dividing it by 3.

```
% -- Reading the Image --
img = imread(filename);
img = im2double(img);

% -- Dimensions of Image --
s = size(img);

% -- Individual Dimensions --
v_dimensions = s(1);
h_dimensions = s(2);
v_dimen_fix = floor(v_dimensions/3);

% -- Separating Color Channels --
b_im=img(1:v_dimen_fix,:);
g_im=img(v_dimen_fix+1:2*v_dimen_fix,:);
r_im=img(2*v_dimen_fix+1:3*v_dimen_fix,:);
```

*Figure 2*

Therefore, we have three different channels in **Figure. 2**. We need to concatenate them to get a color image which we will be doing in the next section.

## 2.2 Color Image Generation

As we have three channels named as b_im (Blue Image), r_im (Red Image) and g_img (Green Image). We concatenate three images using ***cat*** function which is a predefined function in matlab. Once, we concatenate into a single image in **Figure. 3**. We write it to a **JPEG/JPG** file.

```
% -- Aligning together --
RGB_im = cat(3,r_im,g_im,b_im); % Concatenating Images
imwrite(RGB_im,output_num_color_ext,'jpg'); % Write to File
```

*Figure 3*

Though the images are merged together. They are not aligned properly and have a color shift. We fix the alignment in the next section.

# 3 Aligning the Images.

To align the images correctly. We consider one channel i.e. Blue Channel as a Fixed Channel and the other two viz., Red Channel and Green Channel as Moving Images which move around upon the Fixed Image and calculating the **SSD** (Sum of Squared Differences) simultaneously.

## 3.1 SSD (Sum of Squared Differences)

To define the similarity between two features, we calculate the **SSD** of two images. If the difference is negligible, then the two features are perfectly matched. In this assignment, we calculate **SSD** every time we shift the one image (**MOVING**) over the Blue channel (**FIXED**)

$$\boldsymbol{sum}(\boldsymbol{sum}((\boldsymbol{image1}-\boldsymbol{image2})^2))$$

### 3.1.1 Aligning using SSD

Here, in this section we align the images by looping window over displacements and calculating the **SSD** every time we displace. If **SSD** goes below the threshold, then the images are perfectly matched. So, copy the offsets where the **SSD** is minimum and translate the image in such a way that the moving plate perfectly aligns to the fixed plate. We created a function which takes in two channels where one is moving and the other is fixed (Blue Channel) and returns back the aligned plate of the moving plate.

***Main.m***

```
fprintf('shifted_red w.r.t blue_channel\n');
shifted_red = im_align1(r_im,b_im);

fprintf('shifted_green w.r.t blue_channel\n');
shifted_green = im_align1(g_im,b_im);
```

### 3.1.2 Function (*im_align1*)

```
function align = im_align1(channel1,channel2)

    img1 = channel1(v1,h1);
    img2 = channel2(v2,h2);

    % Generate Shift Values
    ranges_1 = -20:20;
    ranges_2 = -20:20;

    %pre_ssd = sum(sum(channel1-channel2).^2);
    pre_ssd = inf;
    ranges = [0 0];

    for i = ranges_1
        for j = ranges_2
            %temp_channel = imtranslate(channel1, [i j]);
            temp_channel = circshift(img1, [i j]);
            temp_ssd = sum(sum((img2-temp_channel).^2));
            %temp_ssd = immse(img2,temp_channel);
            if temp_ssd < pre_ssd
                pre_ssd = temp_ssd;
                ranges(1) = i;
                ranges(2) = j;
            end
        end
    end

    align = circshift(channel1, [ranges(1) ranges(2)]);
end
```

### 3.1.3 Circshift

Circularly shifts the values in the array, A, by shiftsize elements. shiftsize is a vector of integer scalars where the n-th element specifies the shift amount for the n-th dimension of array A. If an element in shiftsize is positive, the values of A are shifted down (or to the right). If it is negative, the values of A are shifted up (or to the left). If it is 0, the values in that dimension are not shifted.

### 3.1.4 Writing Aligned Images to Files.

Once we have the shifted red channel and green channel. We Concatenate these two with the blue image as the blue image was kept as a fixed plate.

```
% -- Aligning together, Writing --
RGB_final_ssd = cat(3,shifted_red,shifted_green,b_im);
imwrite(RGB_final_ssd,filename_ssd,'jpg');
```

Similar way, we do the same using Normalized Cross Correlation as the metric. We will be doing that in the next section.

### 3.2 Normalized Cross Correlation

We use Normalized Cross Correlation (NCC) metric to produce a set a correspondence. The correlation score is higher only when darker parts of the template overlap darker parts of the image, and brighter parts of the template overlap brighter parts of the image. Just like the contrasting the SSD, Normalized Cross Correlation generates a peak (Higher Value) when there is a perfect match between two feature windows.

image1./|image1| and image2./|image2|

#### 3.2.1 Aligning using NCC

Here, in this section we align the images by calculating the peak values viz., **xpeak** and **ypeak**. These peak values are the coordinate values where it observed a peak or had perfect match. Now, using the peak values, we substract the horizontal and vertical dimensions of the moving image with the xpeak and ypeak to get the resultant offset values. Once we have the offset values. We use **Circshift** just like how we used in the previous section. We created a function which does the following by taking in two channels where one is moving and the other is fixed (Blue Channel) and returns back the aligned plate of the moving plate.

***Main.m***

```
fprintf('shifted_red w.r.t blue_channel\n');
shifted_red_ncc = im_align2(r_im,b_im);
fprintf('shifted_green w.r.t blue_channel\n');
shifted_green_ncc = im_align2(g_im,b_im);
```

#### 3.2.2 Function (*im_align2*)

```
offset = [0 0];
img1 = channel1;
img2 = channel2;

% -- Calculation of Norm Correlation --
c = normxcorr2(img1,img2);

% -- Calculating the Peak --
[max_c, imax] = max(abs(c(:)));
[ypeak, xpeak] = ind2sub(size(c),imax(1));

% -- Calculating Offsets --
offset(1) = ypeak - size(img1,1);
offset(2) = xpeak - size(img1,2);

align = circshift(channel1, [offset(1) offset(2)]);
```

#### 3.2.3 Writing Aligned Images to Files.

Once we have the shifted red channel and green channel. We Concatenate these two with the blue image as the blue image was kept as a fixed plate.

```
RGB_final_ncc = cat(3,shifted_red_ncc,shifted_green_ncc,b_im);
%figure, imshow(RGB_final_ncc);
imwrite(RGB_final_ncc,filename_ncc,'jpg');
```

# 4 Output

## 4.1 Aligned Image using SSD

Image aligned using SSD as the metric is displayed below

*Figure 4*

## 4.1 Aligned Image using NCC

Image aligned using NCC as the metric is displayed below

*Figure 5*

# 5 Conclusion

Therefore, creating a color image for each of the alignments and writing back to file is done successfully.