Thanks:

I wants to thank Professor Min Wu and her kind teaching assistant Chau-wai for their helps during spring 2015. They really helped me in having a better understanding of image processing and implementing what we learned in the class. Their project not only taught me technical issues in image processing, also gave me insight into engineering issues such as timelines, group works, consistency, and project handling.

Introduction:

Suppose you are in a small class with your college friends and you want to take a selfie but your camera lens cannot cover everybody, what would you do in this case without changing the camera? The answer is PANORAM image stitching. It can help you to take overlapping pictures and then stitch them together to make a picture that can cover a wider area. As another example, let's say you are in Grand Canyon and you'd like to take a picture that is able to show the same view as the one your eyes are able to catch. This is the same technique that Google uses to make its map. If you go to maps.google.com and change the view option into land view you are able to see a wide view of your desired area.

This was a good motivation for us to start doing this project. By having multiple pictures from the same panoramic view but different angles how it is possible to make one picture capable of representing all of them. Also, it should be taken into account that the pictures of the same panorama has overlapping parts that enable us to extract the common space between two consecutive picture and stitch them together. In other words as it can be seen from the following pictures, the pictures have overlapping parts which is the key in our analysis in order to stitch them to each other.

(a)

(b)

(c)

Figure 1. Overlapping parts of consecutive images a, b,c.

Project Overview:

MATLAB is used for simulations and implementing this projects. This project consists of different sections in the form of different MATLAB sub functions. In this section we are going to give an overview of different sections of this project and give you an initial insight of what's going on here while in the next sections we will deal with different parts specifically and dig more into them. The overall block diagram of the project is as figure (2). The program works on pairs of pictures, it takes two pictures as input. The first part of the program deals with extracting the key points of each picture. By the aid of SIFT algorithm described by D. Lowe [1], it is able to extract the important points. After that, the key points of each picture are fed into another function which is able to find the pairs that match together, let's call the match points MP1 and MP2 in picture one and two respectively. MP1 and MP2 are the main structure of our project from now, in order to make two picture to be seemed in the same back ground we need to first transform one of them to the other picture space. The good news is that we can use MP1 and MP2 as the sample points to find the appropriate transformation, this transformation is called affine transformation. This transformation helps us to transform pictures into each other's space. The next step is providing a substrate that we can lay two picture, the transformed one and the other one and make them seem as a whole unit picture. Now we can combine two pictures in the new space and do some averaging on the common areas. At the end, there might be some imperfections and seams on the joint parts. We try to merge a specific region in the first image to a user specified region in the second image using multi-band blending. In the following sections we are going to describe each part of the block diagram more specifically and detailed.

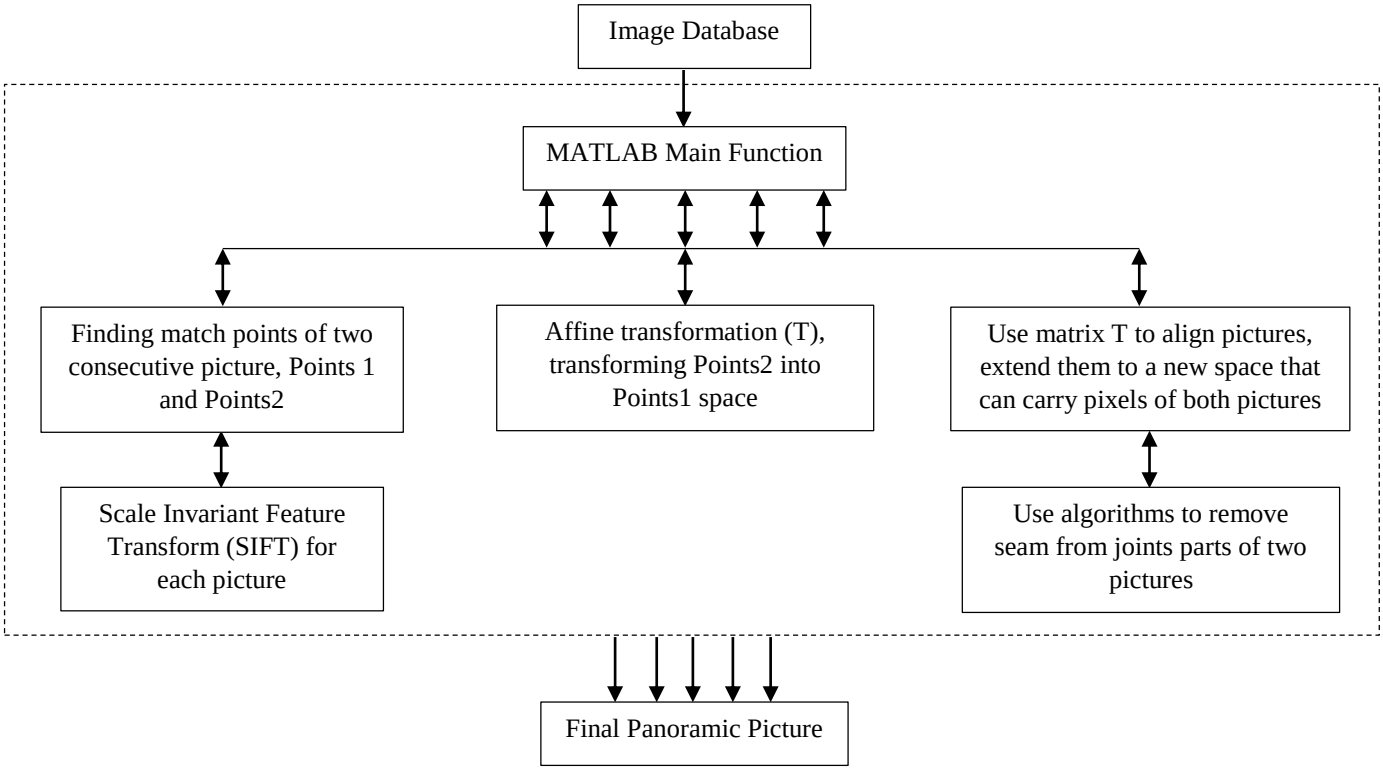

Figure 2. Overall Block diagram of Panorama image stitching

Data Description:

Two sample pictures as shown in figure (3) are used in order to describe different steps of this project. The pictures and the pictures of figure (1) are given by Professor Min Wu at University of Maryland, College park.

(a)

(b)

Figure 3. Sample pictures for explanation of panorama image stitching

Scale Invariant Feature Transform (SIFT):

This part of the project is based on David G. Lowe's paper [1] on extracting invariant features from image which are independent of the image quality, different views of object, rotation, noise, and brightness. Suppose our pictures database consists of two pictures that these two pictures have some overlapping parts as it is shown in figure (3). SIFT algorithm is able to extract the key points of the two pictures which is very helpful for our application. SIFT gives a vector of features for key points in two pictures. Then we compare different feature vectors of two images and decide which points matches to each other.

For now let's focus on sift and see how it works. According to [1] there are four steps in extracting image features;

1) Scale-space extrema detection: Using a difference of Gaussian function to identify points that are invariant to scale and orientation. If we consider I(x,y) as input image, and G(x,y,σ) as variable scale Gaussian, L(x,y,σ) is defined as scale space of an image.

$$L(x,y,\sigma)=G(x,y,\sigma)*I(x,y) \tag{1}$$

$$G(x,y,\sigma)=\frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2}$$

The following equation is used in order to extract the stable keypoint locations in scale space. It can also make the image smoother. Also the difference of Gaussian is a close approximation of Laplacian of Gaussian.

$$D(x,y,\sigma)=\big(G(x,y,k\sigma)-G(x,y,\sigma)\big)*I(x,y)=L(x,y,k\sigma)-L(x,y,\sigma) \tag{2}$$

2) Keypoint localization: This step is done in order to reject low contrast and poor localized edges. This approach uses Tylor expansion of the scale-space function D(x,y,σ).

$$D(x)=D+\frac{\partial D^T}{\partial x}x+\frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{3}$$

In order to find the location of the extremum, $\hat{x}$, we take the derivative of D(x) and set it to zero.

$$\hat{x}=-\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x} \tag{4}$$

By putting equation (4) into equation (3), we can reject the lowest contrast extrema.

$$D(\hat{x})=D+\frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x} \tag{5}$$

In order to make it even more stable, Lowe also eliminates edge responses. He uses 2×2 Hessian matrix at the keypoint location and taking the derivative around the keypoint neighborhood. By doing the following operation, just a few points will survive, in which r is some experimental value.

$$\frac{Tr(H)^2}{Det(H)}<\frac{(r+1)^2}{r} \tag{6}$$

3) Orientation assignment: This is a necessary step in order to make keypoints invariant to the rotation. By using L from the first equation, we can compute gradient magnitude, m(x,y), and orientation, θ(x,y).

$$m(x,y)=\sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2} \quad (7)$$

$$\theta(x,y)=\tan^{-1}((L(x,y+1)-L(x,y-1))/(L(x+1,y)-L(x-1,y))) \quad (8)$$

Any point that is in the orientation histogram peak point is selected as the keypoint.

4) Keypoint descriptor: This step tries to find the overall gradient magnitude and orientation in the keypoint location neighborhood.

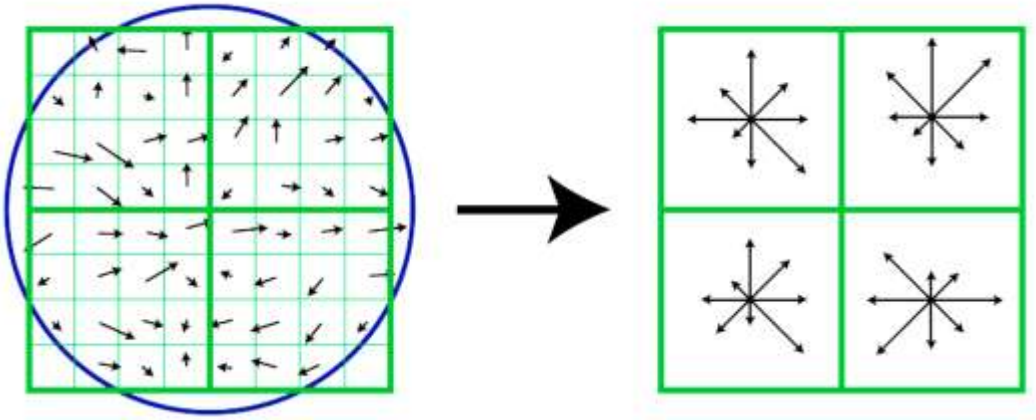

Figure 4. Keypoint descriptor by computing the gradient magnitude and orientation around each keypoint. [1]

In our simulations, the final output of SIFT section consists of K keypoints for each picture. Each point has a vector consists of two parts, the first part describes the pixel location, orientation, and scale of the point. The second part is a vector which has the size of 128, this the feature vector of that specific point. We used the location of these K keypoint to pin point some of them in figure (5). Thanks to Lowe, for implementing this part, the SIFT package for extracting keypoints was used.

After performing SIFT on both pictures, some key points on two pictures are shown in figures (5).

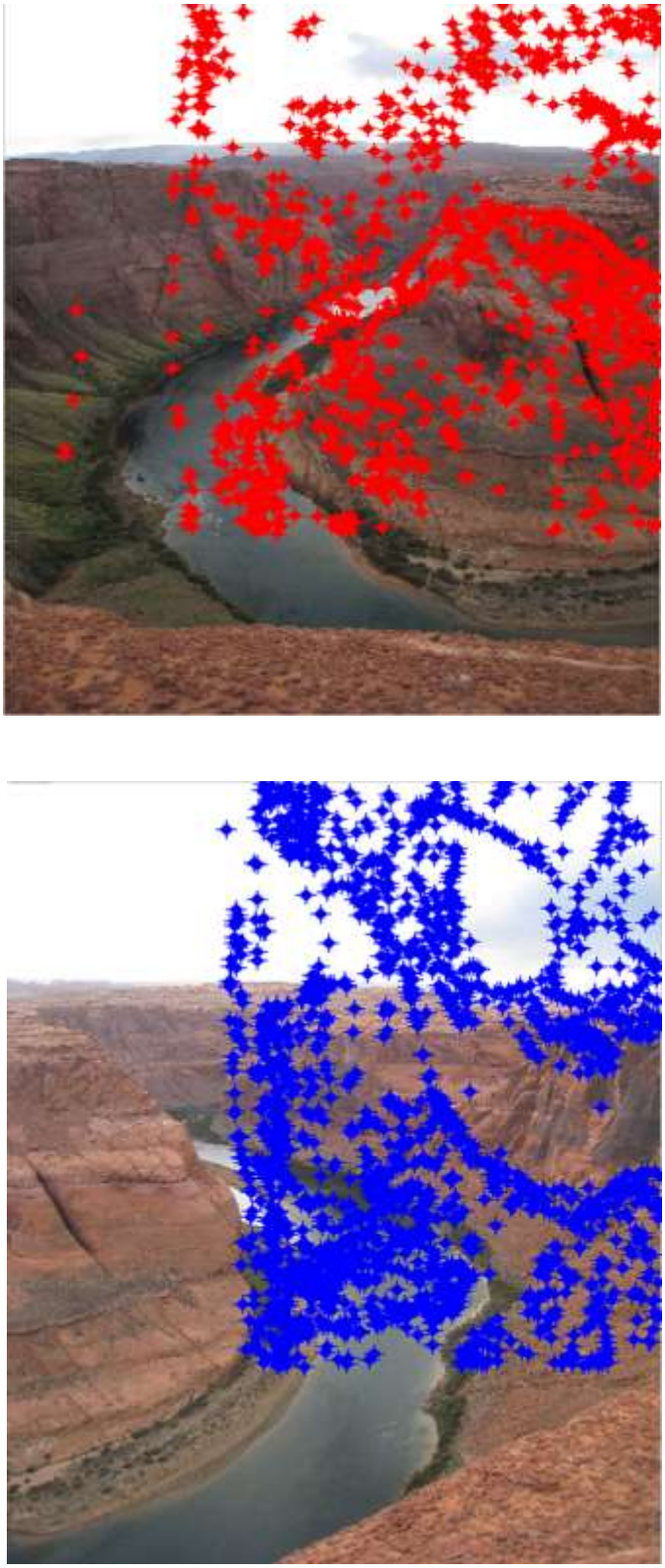

Figure 5. Extracting key points from two pictures with overlaps.

Matched Keypoints Between Two Pictures:

After finding the key point of each picture, we need to find the pairs of keypoints in two pictures that match together. In other words, the overlapping parts has some common points that we need to stitch two picture from these common points. In this section we are going to see how it is possible to find the common keypoints between two pictures. As it was noted in the previous section, each keypoint has a feature vector F that describes some of the properties of that keypoint. If we consider the whole feature vector in the first picture F_1, then F_1 will be a $K \times 128$ matrix in which K is the number of keypoints and 128 is the size of feature vector for each keypoint. The same procedure applies for picture two and we call it F_2. The matching points feature vector has the least distance from each other. The measurement criteria for our case would be dot products, in other words, we take the dot product of each key point in the first picture with all points in the second picture. We show this by using one sample point A from picture 1.

F_1=K×128 feature vector for K keypoints in picture 1.

F_2=K×128 feature vector for K keypoints in picture 2.

$$DP_A = F_1(A,:).F_2^T \tag{9}$$

DP_A is the dot product of the feature vector of keypoint A with all the feature vectors in the second picture. DP_A is vector of the size 1×K, one measurement criteria here could be Arccos of DP_A, because the points that has the nearest feature vector to each other has the largest value in DP_A while their Arccos has the least value. Arccos(DP_A) is a vector of the same size as DP_A, we first sort this vector from the lowest to the highest value. The lowest value is the first element of this vector, and the next lowest value is the second element of that. We set some experimental values that if the first value is less than 0.7 times second value, this point (A) is a good candidate. Otherwise, we go to the next point and check the same condition for the next point. We name the sorted Arccos(DP_A) as SArccos(DP_A) and express this paragraph in the following format.

If SArccos(DP_A)[1]<0.7×SArccos(DP_A)[1] then we take the corresponding indices from Arccos(DP_A) and treat that as the match point.

We can do the same for all other points in picture 1 and find their corresponding match points in the second picture. The good thing about dot product is its cheap computation, this enables the program to be much faster than other possible choices.

Implementing the matched keypoints algorithm for all points will result in figure (6) which connects the corresponding points to each other.

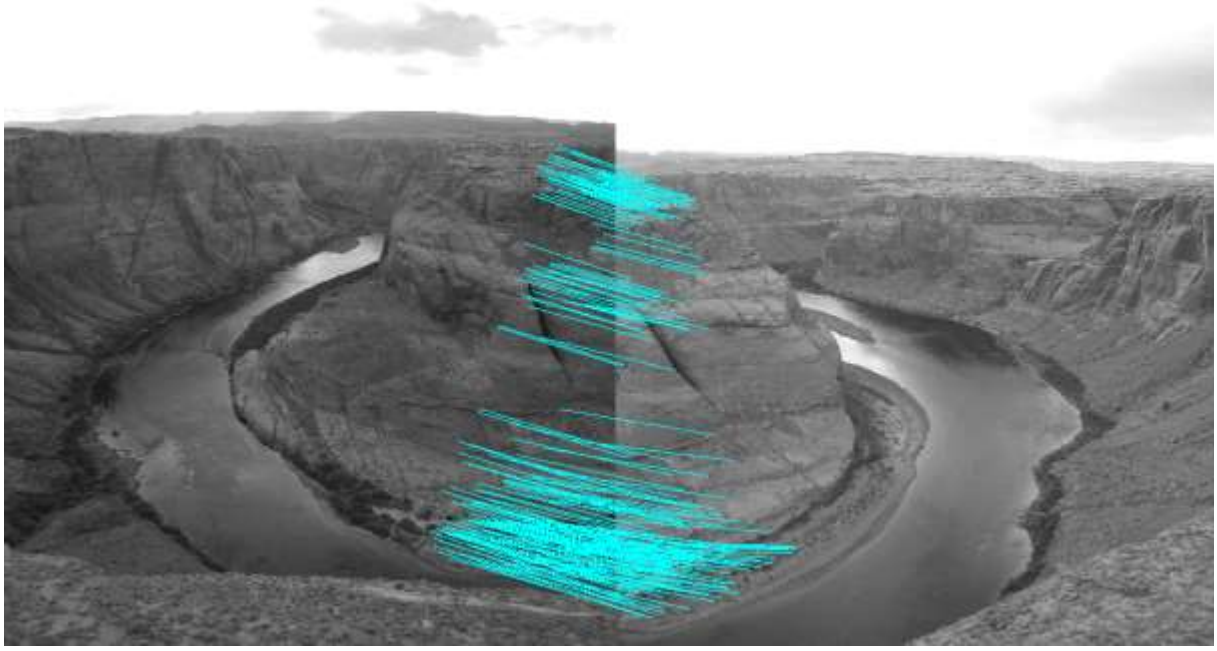

Figure 6. Corresponding match points between first and second picture.

Calculating Appropriate Transformation:

Now that we have keypoints from the first and second picture, it's the time to calculate the appropriate transformation that aligns K keypoints of picture 1, P1, to K keypoints of picture 2, P2. In order to perform the aligning section, RANdom SAmple Consensus (RANSAC) transformation was used. According to [2] RANSAC is an efficient resampling technique which uses the minimum number of datapoints to estimate the underlying model parameters. It tries to use the smallest possible datasets and proceeds to enlarge this set with consistent data points. As said in [2] RANSAC consists of 5 main steps.

1) The minimum number of parameters are chosen to randomly model the system parameters.

2) Solve the model parameters with the parameters of last step.

3) Define a tolerance ϵ and see how many points from the set of all points fit this tolerance.

4) Define a threshold TH. If the ratio of inliers over the whole number of points in the set is greater than TH, re-estimate the model parameters using the identified inliers. You are done!

5) If step 4 does not hold, repeat step one to four (N times maximum).

By having this back ground about RANSAC, we want to find the appropriate transformation for aligning P_1 to P_2. As we said, RANSAC is very efficient in terms of model parameter estimation and it can help us to reduce the simulation time without needing all the points P_1 and P_2 in order to find the appropriate

transformation. It will help to find the desired transformation that aligns P_1 and P_2 with as few points as possible.

Algorithm for finding the appropriate transformation is as follows.

1) Choose three random points.

2) Using the points of the previous step to calculate the appropriate transformation. Least square is our option for finding the transformation. We have three points in picture 1, let's call them (A_1,B_1), (A_2,B_2), (A_3,B_3), as well as three points in picture 2, (A'_1,B'_1), (A'_2,B'_2), (A'_3,B'_3). We can write the transformation equations in the following format.

$$[A_1 \quad B_1 \quad 1]\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}=[A'_1 \quad B'_1 \quad 1] \tag{10}$$

$$[A_2 \quad B_2 \quad 1]\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}=[A'_2 \quad B'_2 \quad 1] \tag{11}$$

$$[A_3 \quad B_3 \quad 1]\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}=[A'_3 \quad B'_3 \quad 1] \tag{12}$$

We can combine equation 10, 11, and 12 in the following format.

$$\begin{bmatrix} A_1 & B_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & A_1 & B_1 & 0 & 1 \\ A_2 & B_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & A_2 & B_2 & 0 & 1 \\ A_3 & B_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & A_3 & B_3 & 0 & 1 \end{bmatrix}\begin{bmatrix} t_{11} \\ t_{21} \\ t_{12} \\ t_{22} \\ t_{31} \\ t_{32} \end{bmatrix}=\begin{bmatrix} A'_1 \\ B'_1 \\ A'_2 \\ B'_2 \\ A'_3 \\ B'_3 \end{bmatrix} \tag{13}$$

In equation (13), the only unknowns are the transformation matrix elements. We can easily find the by some inverse operations in MATLAB. It will give us the transformation matrix as follows.

$$T=\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix} \tag{14}$$

3) Once we found the transformation T that transforms the keypoints P_1 to the keypoints P_2, we can transform other candidate points in P_1 and find their corresponding points in the second image and call them P_2. At the end we compare them with P_2 to see what the error is. Least square error calculation is as follows, we do this process for all K keypoints.

$$\Delta x^2(i)=\left(P_{2x}(i)-P'_{2x}(i)\right)^2 \tag{15}$$

$$\Delta y^2(i)=\left(P_{2y}(i)-P'_{2y}(i)\right)^2 \tag{16}$$

$$Error(i)=\sqrt{\Delta x^2(i)+\Delta y^2(i)} \tag{17}$$

For each i, if the error is less than some threshold, one will be added to the number of inliers, $N_{inliers}$.

4) Iterating over the last three steps will give us N different $N_{inliers}$. At the end we choose the transformation that has the maximum number of inliers.

The Overall block diagram of these 4 steps is explained in figure (7).

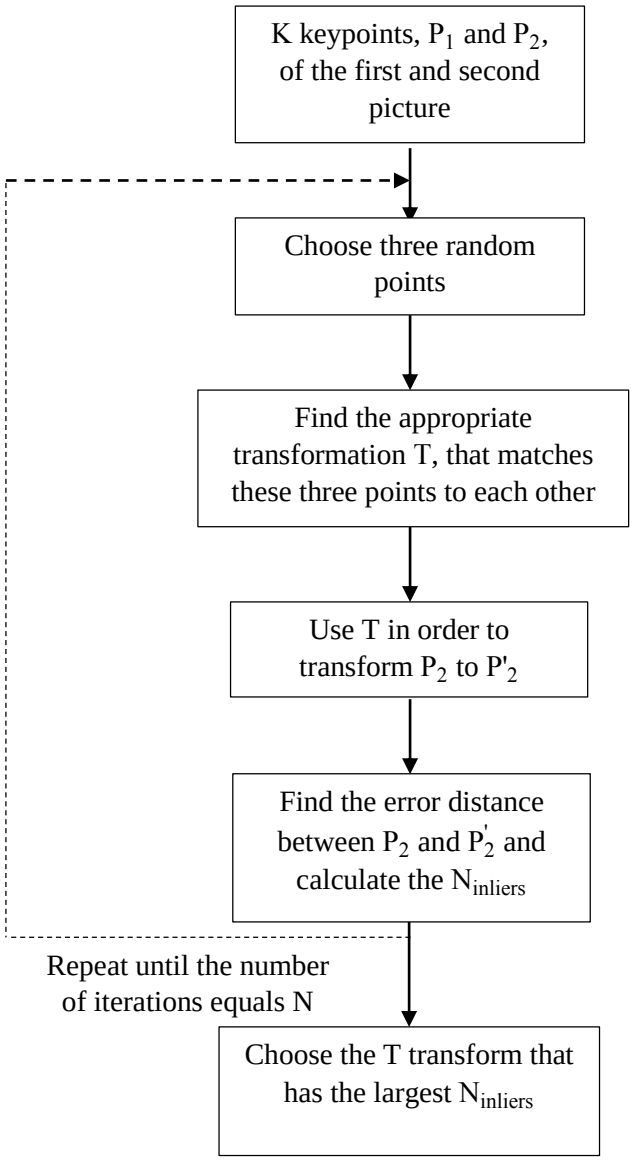

Figure 7. Block diagram of finding T transformation for transforming P1 to P2.

Stitching Pictures:

Now that we have the appropriate transformation, T, we can align two pictures together and make them seem as a whole united picture. In order to do that we first use T to transform one of the pictures to the other picture domain and form a new substrate that has both pictures. After that, we keep the uncommon part of two pictures and take average of the common parts of two pictures and replace it in its appropriate position. Different steps of stitching two pictures are described in this section. Before introducing different steps, lets define some parameters.

X_{I1}: Number of pixels in x direction for first picture.

Y_{I1}: Number of pixels in y direction for first picture.

X_{I2}: Number of pixels in x direction for second picture.

Y_{I2}: Number of pixels in y direction for second picture.

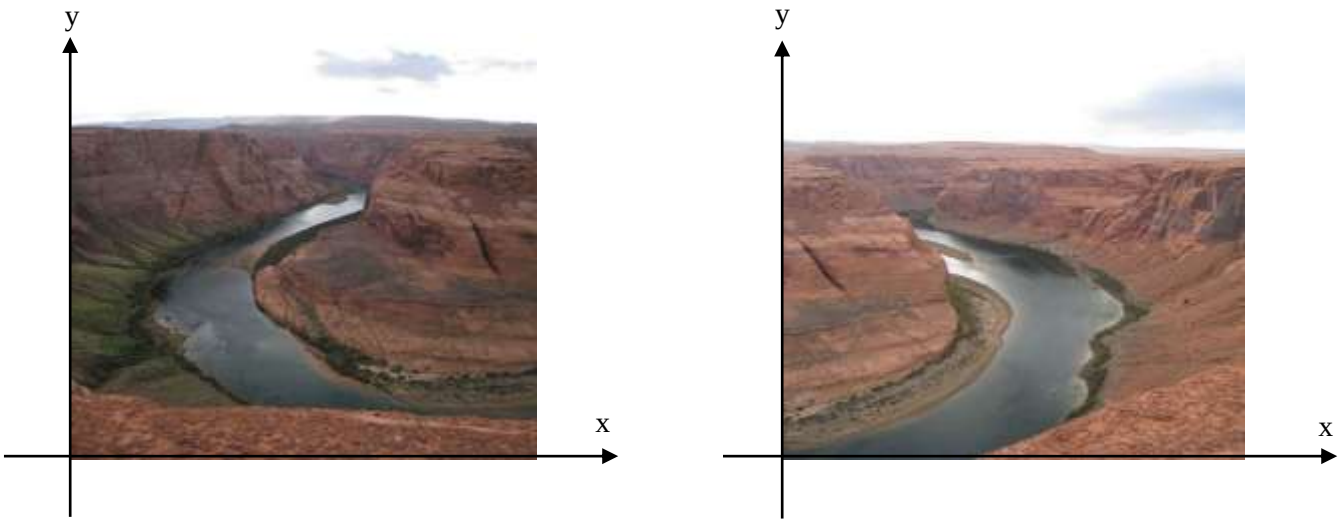

Figure 8. Two pictures in their Cartesian coordinate.

1) Create a two masks $M_1$and M_2 for picture one and two respectively, they have the same size of their corresponding image and they are matrices of ones and zeros. We use the sample pictures that are shown in figure (8) to describe.

2) Use T transformation matrix to transform one of the picture (Here we consider the second one for our simulations). Let's call them I_2 and M_2, where I_2 is the matrix of pixel values for the second image and M_2 is the constructed mask in the first step.

$$I_2 \xrightarrow{T} I_{2t} \tag{18}$$

$$M_2 \xrightarrow{T} M_{2t} \tag{19}$$

When we transform I_2 to I_{2t}, the new transformed image has some minimum and maximum values for its Cartesian boundary in x and y direction. We call the minimum boundary as $[X_{min},Y_{min}]$ and the maximum boundary as $[X_{max},Y_{max}]$. It is also shown in figure (9).

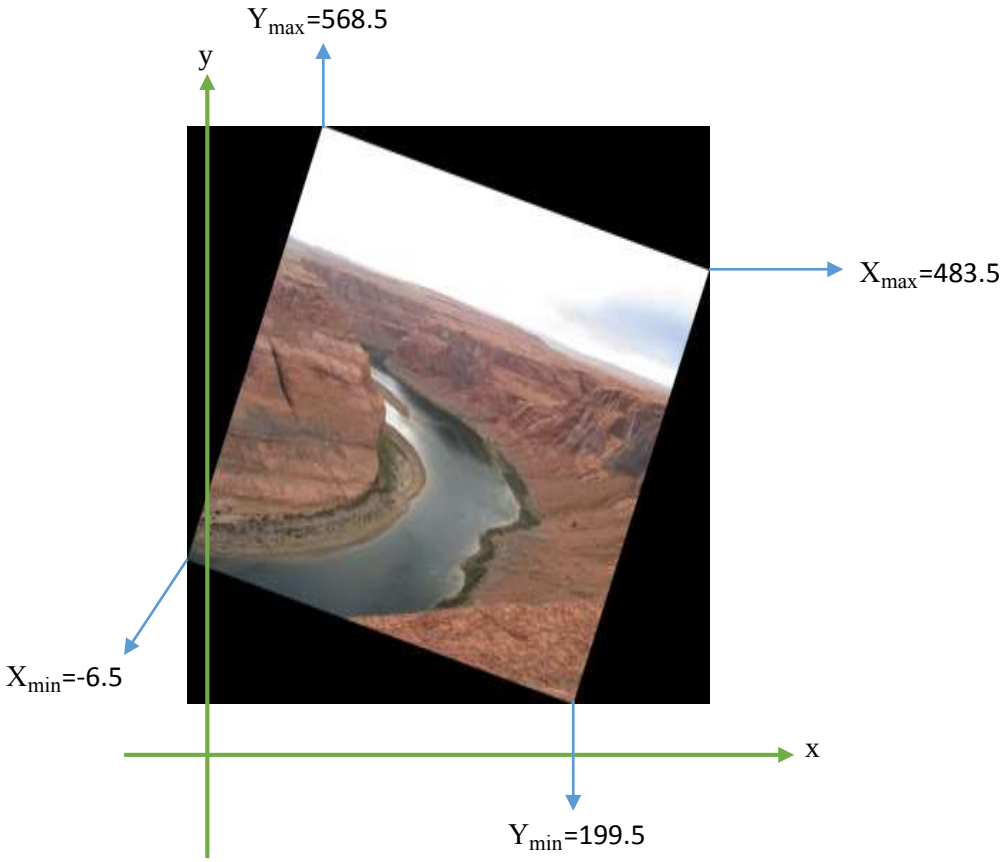

Figure 9. Second picture after transformation.

3) The stitched picture that consist of two pictures has some boundaries for its pixel value. In this step we are going to set the number of pixels for the stitched picture in the x and y directions. We name the number of pixels in x direction as *H* and in y direction as W.

$$W=\text{Max}\ (X_{I1}, X_{I1}-X_{min}, X_{I2}, X_{I2}+X_{min}) \quad (20)$$

$$H=\text{Max}\ (Y_{I1}, Y_{I1}-Y_{min}, Y_{I2}, Y_{I2}+Y_{min}) \quad (21)$$

As we said H and W are the height and width of the new stitched picture, which can be shown in the following picture. In other words H and W are height and width of the substare that we want to put the first and second picture on that, which are shown in figure (10).

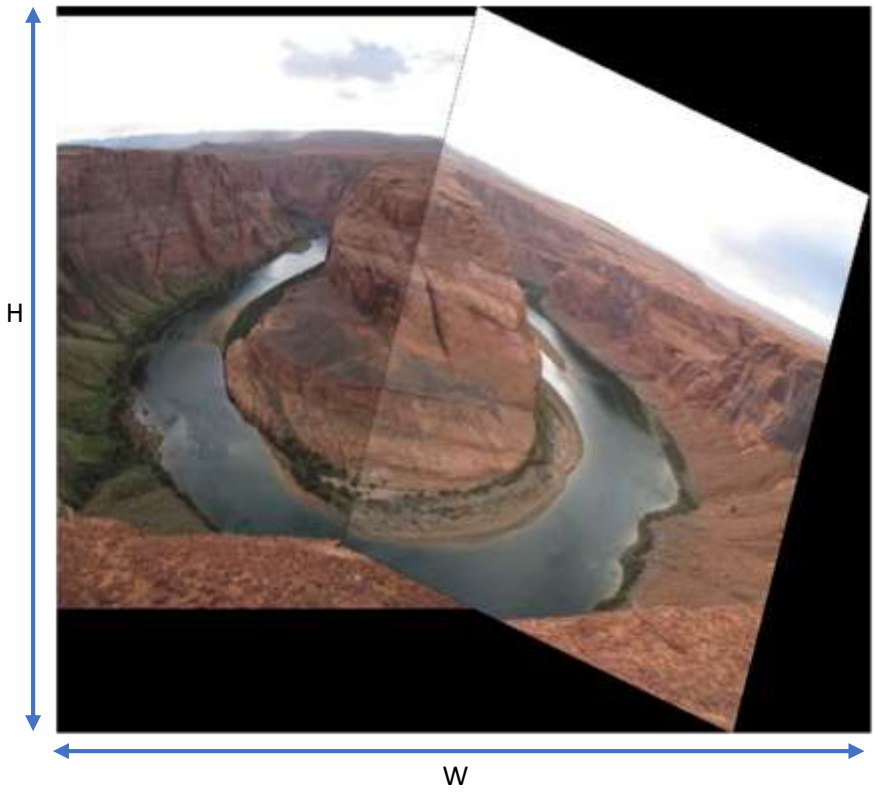

Figure 10. Height and width of the stitched pictures.

If you pay close attention to figure (10), it can be seen that picture one and two are shifted based on the the values of $[X_{min},Y_{min}]$ and $[X_{max},Y_{max}]$ after transforming I_2 which is I_{2t}. We do the shift based on the following rules, also the shift matrix for picture one and two are T_{shift1} and T_{shift2}respectively.

If $X_{min}<0$, $T_{shift1}(3,1)=-X_{min}$ otherwise it will be zero.

If $Y_{min}<0$, $T_{shift1}(3,2)=-Y_{min}$ otherwise it will be zero.

$$T_{shift1}=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{shift1}(3,1) & T_{shift1}(3,2) & 1 \end{bmatrix}$$

The same method applies for the second picture as well.

If $X_{max}>0$, $T_{shift2}(3,1)=X_{max}$ otherwise it will be zero.

If $Y_{max}<0$, $T_{shift2}(3,2)=Y_{max}$ otherwise it will be zero.

$$T_{shift2}=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{shift2}(3,1) & T_{shift2}(3,2) & 1 \end{bmatrix}$$

We use T_{shift1} and T_{shift2} to shift the new formed pictures and it will result in figure(10). Also take into account that in figure (10), the common parts consists of average of two pictures and the uncommon parts just consists of the corresponding picture.

Removing the Seam:

If one pays close attention to the figure (10), it seems that there some imperfections in the boundaries of the common parts. It is called seam and in this sections we are going to remove those seams and make the final result more perfect. We use Poisson Image editing paper[3] and simple example of [4] to describe how it is possible to remove seam from the final output picture. Our goal is to seamlessly cut the area shown in figure (12) and add it to the area shown in figure (11). Solving Poisson equations for interpolation can be a good approach to have seamless editing of the image.

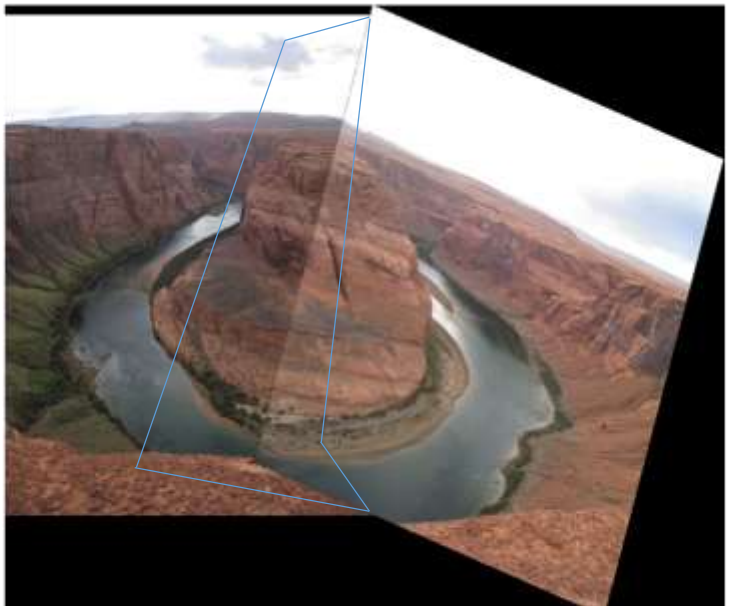

Figure 11. Target Image.

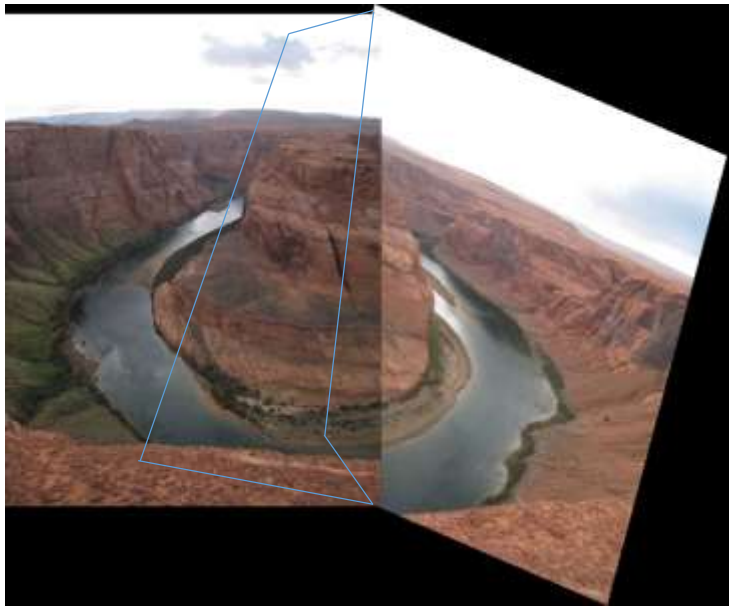

Figure 12. Sources Image.

According to [3] and figure (13), first we define some parameters.

v: Gradient vector of a region in the source image

g: Selected region of the source

f^*: Known function in the S domain which is the target image.

f: Unknown function in the Ω domain, which we are interested to find it out.

Ω: g that is placed on S.

∂Ω: Boundaries between the source and target region.

Here we interested to find f which is the blended region in the target picture. We can express it with the following equation.

$$\min_f \iint_\Omega |\Delta f - v|^2 \text{ given that } f|_{\partial\Omega} = f^*|_{\partial\Omega} \quad (22)$$

The solution of the equation (22) is the solution of the Poisson equation which can be written as the equation (23).

$$\Delta f = \text{div} v \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \quad (23)$$

In the above equation, $\text{div} v = \frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}$ and Δ is the Laplacian operator.

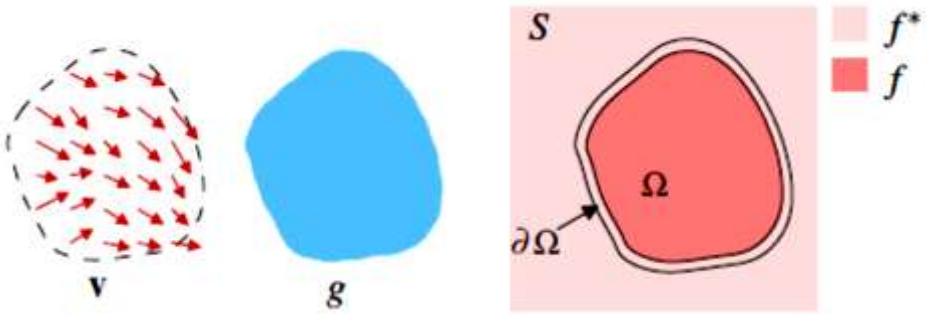

Figure 13. Poisson Image Blending.[3]

After applying what we said here, the final result will be as in figure (14).

Figure 11. Final picture after Poisson image blending.

References:

[1] Lowe, David G. "Distinctive image features from scale-invariant keypoints." International journal of computer vision 60.2 (2004): 91-110.

[2] G. Derpanis, Konstantinos. "Overview of the RANSAC Algorithm". May 13 2010.

[3] Pérez, Patrick, Michel Gangnet, and Andrew Blake. "Poisson image editing." ACM Transactions on Graphics (TOG). Vol. 22. No. 3. ACM, 2003.

[4] http://eric-yuan.me/poisson-blending/