Computer Vision 1

Image Mosaicing Report

Department of EECE, Northeastern University

Taoran Liu and Jianpeng Chen

Mar 12, 2023

Abstract

The purpose of this project is to use computer vision algorithms to stitch two images together to create a single panoramic image. Specifically, we will use a Harris corner detector to find corner features in both images, automatically find corresponding features, and estimate a homography between them to align one image with the coordinate system of the other. We write a program to implement these ideas and test its performance in experiments. The report will include a description of the algorithms, a detailed explanation of the experimental results, a discussion of the parameter values used, and the conclusions we drew. In addition, we will provide the source code and flowchart of our program, as well as sample images to demonstrate our algorithm.

Key words: Image Mosaic, Harris Corner, NCC, Homography, RANSAC

1 Flowchart

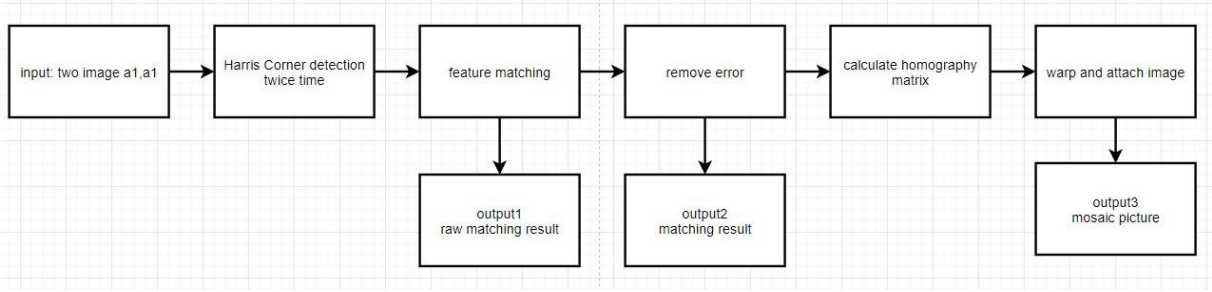


Figure 1-1 Main Flowchart

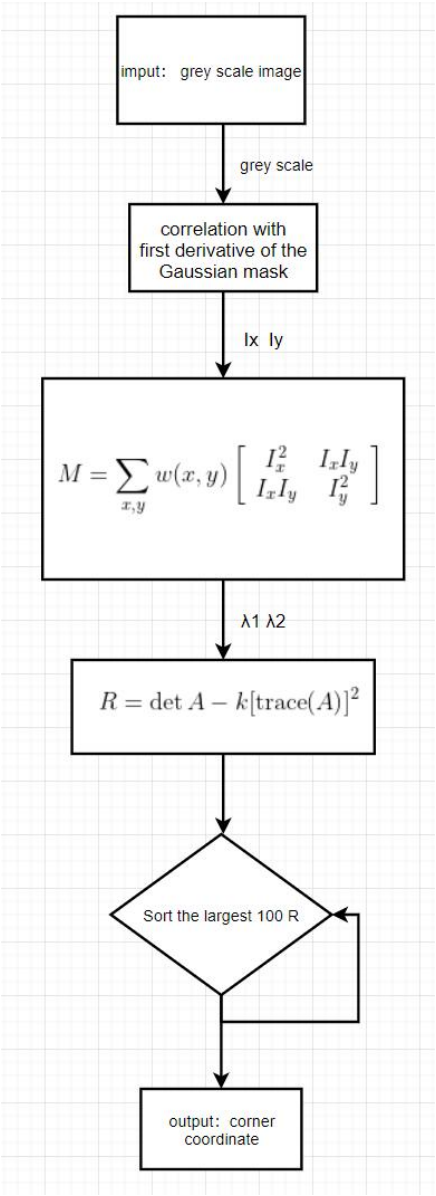


Figure 1-2 Harris Corner Detect

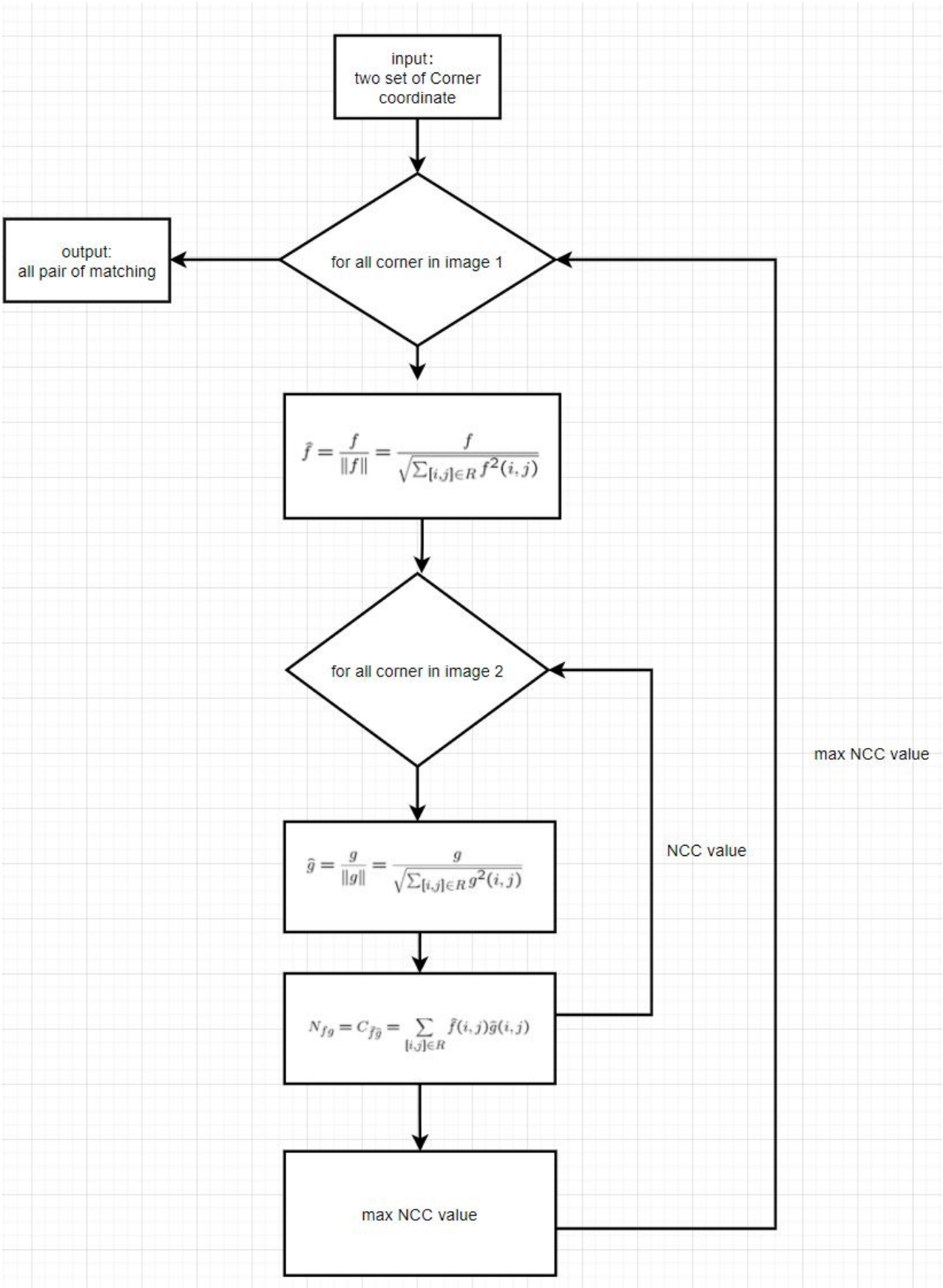


Figure 1-3 NCC process

2 Algorithm Description

2.1 Conner detection

2.1.1 Introduction

Corner is an important feature of an image, which plays an important role in the understanding and analysis of image graphics. While retaining the important features of the image, the corner point can effectively reduce the amount of information data, make the information content very high, effectively improve the calculation speed, and facilitate reliable image matching and real-time processing.

2.1.2 The function of extracting point features

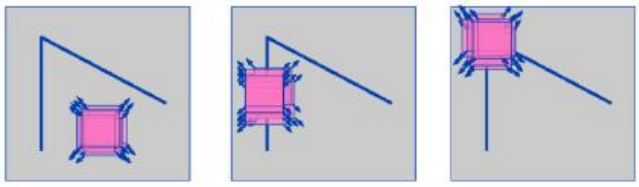
The point feature of the image is the basis of many computer vision algorithms: using feature points to represent the content of the image has many uses in the directions of moving target tracking, object recognition, image registration, panoramic image stitching and 3D reconstruction.

There is an important class of point features: corner points. Corner points: Points where the local window moves in all directions, resulting in significant changes, and points where the local curvature of the image changes suddenly Typical corner detection algorithms: Harris corner detection, CSS corner detection, etc.

2.1.3 Harris Basic idea

Basic idea: observe image features from a small window in the local image.

Corner definition: The movement of the window to any direction will cause obvious changes in the gray level of the image.



"flat" region: no change in all directions

"edge": no change along the edge direction

"corner": significant change in all directions

2.1.4 Harris corner detection: mathematical description

Translate the image window in $[u, v]$ to produce grayscale changes $E(u, v)$

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+u) - I(x,y)]^2$$

$$E(u,v) = \sum_{x,y} w(x,y)\left[I_x u + I_y v + O(u^2, v^2)\right]^2$$

$$\left[I_x u + I_y v\right]^2 = [u, v] \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Therefore, for a small local movement $[u, v]$, the following expression can be approximated:

$$E(u,v) \cong [u,\ v] \quad M \quad \begin{bmatrix} u \\ v \end{bmatrix}$$

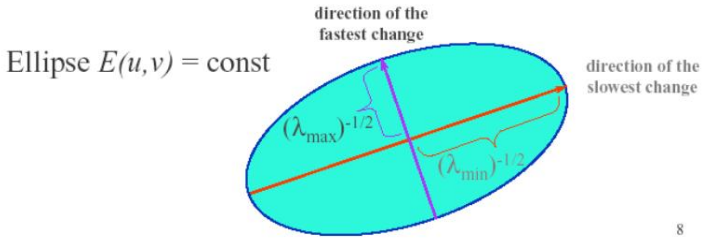
Among them, M is an 2X2 matrix, which can be obtained by the derivative of the image:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Image changes due to window movement: eigenvalue analysis of real symmetric matrices

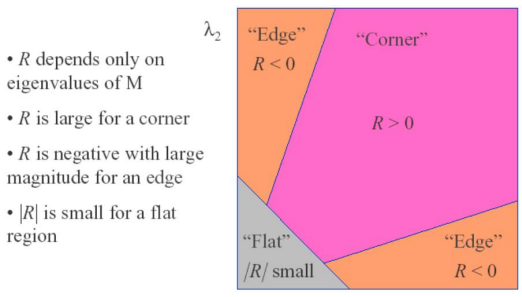
$$E(u,v) \cong [u,\ v] \quad M \quad \begin{bmatrix} u \\ v \end{bmatrix}$$

Among them, the eigenvalues of λmax λmin



8

$$R = \det M - k\ \mathrm{trace}^2(M)$$



2.1.5 Conclusion and Program Application

The Harris corner detector is divided into three steps: gradient calculation, matrix formation and eigenvalue calculation.

First, smooth gradients in the x and y directions are computed (using first derivative of the Gaussian) to detect corners in a given grayscale image I(x,y), given by:

$$g_x(x,y) = \frac{-x}{2\pi\tau_g^4}\exp\left(-\frac{x^2+y^2}{2\tau_g^2}\right)$$

$$g_y(x,y) = \frac{-y}{2\pi\tau_g^4}\exp\left(-\frac{x^2+y^2}{2\tau_g^2}\right)$$

In our code, we used fx = [5 0 -5;8 0 -8;5 0 -5] and fy = [5 8 5;0 0 0;-5 -8 -5] as first derivative of the Gaussian mask.

Calculate the smooth gradient I(x,y) of the image as:

$$I_x = g_x(x,y) \otimes I(x,y)$$

$$I_y = g_y(x,y) \otimes I(x,y)$$

Calculate the M matrix corresponding to the picture:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Use the det and trace of the M matrix to calculate the R value corresponding to each pixel, and use the regional maximum value as the corner pixel coordinates

$$R = \det A - k[\mathrm{trace}(A)]^2$$

After we got R value for every pixel, we set use a window with 3X3 size to find local maximum R value as identified corners.

In our code, we didn't use any constant value of R as threshold to judge whether one result is a good conner. We sort all the results from largest to smallest, and use largest 100 R value as the best output conner set.

2.2 Feature matching

2.2.1 Concepts of NCC

NCC (normalized cross correlation) algorithm, normalized cross-correlation matching method, is a matching method based on image grayscale information. It is a common image processing method to compare the similarity of two images.

There are three main methods of image matching: grayscale-based, feature-based, and transform domain-based.

The NCC algorithm can effectively reduce the influence of light on the image comparison results. Moreover, the final result of NCC is between 0 and 1, so it is particularly easy to quantify the comparison results, as long as a threshold is given to judge whether the result is good or bad. The traditional NCC comparison method is time-consuming. Although it can be optimized by adjusting the window size and the step rectangle of each detection, it cannot meet the real-time requirements for industrial production detection. Pre-calculation is realized by integrating images, and the template image is compared with the produced one. Subtle differences between electronic versions can help companies improve product quality, reduce the rate of defective products, and control quality.

2.2.2 Basic principle of NCC algorithm

Construct an n×n neighborhood as a matching window for any pixel (px, py) in the original image. Then, for the target pixel position (px+d,py), a matching window of size n×n is also constructed to measure the similarity between the two windows. Using following equation.

$$\hat{f} = \frac{f}{\|f\|} = \frac{f}{\sqrt{\sum_{[i,j]\in R} f^2(i,j)}}$$

$$\hat{g} = \frac{g}{\|g\|} = \frac{g}{\sqrt{\sum_{[i,j]\in R} g^2(i,j)}}$$

$$N_{fg} = C_{\hat{f}\hat{g}} = \sum_{[i,j]\in R} \hat{f}(i,j)\hat{g}(i,j)$$

$$\hat{f} = \frac{f}{\|f\|} \quad \hat{g} = \frac{g}{\|g\|}$$

2.2.3 Program Application

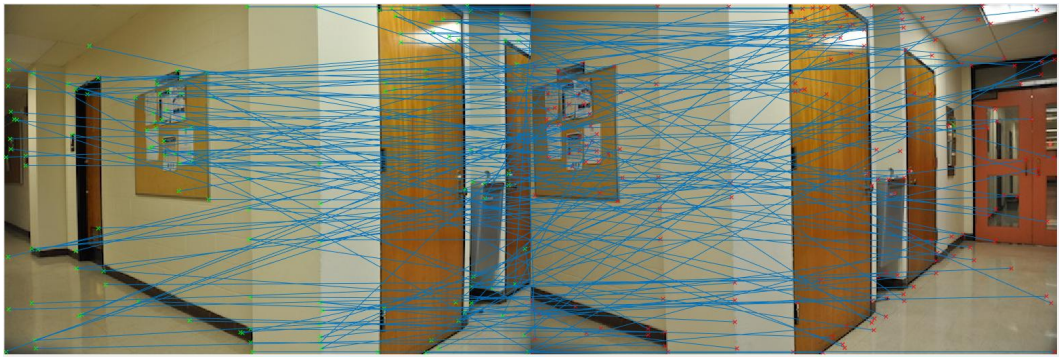
In Harris conner detection part, we find 100 corner with largest R value in each image. In this step, we used these conners as feature points to do NCC matching algorism.

The main idea of our matching function is to do one-way searching for matching points from one image to another. And in main function we run matching function twice times, used first-time output point set as input run matching function again inversely. It will make sure every pair of matches is bidirectional.

In the matching function, first, we pick one conner pixel from image1 and use it's nXn neighbor as a matching window f(i, j). Then use this window and each corner point in picture 2 and the corresponding nXn neighbor g(i, j) to calculate the NCC value, sort all the NCC result from largest to smallest, and use the g(i,j) with largest NCC value as the matching pair with f(i ,j). Last repeat this process for each conner pixels in image1, get all matching pairs from image1 to image2.

2.3 Mispair Correction

Because there are many error cases in the matching pairs obtained from normalized cross correlation, as shown in the figure.



In order to improve the efficiency of ransac to select the correct point, these wrong pairs can be preprocessed. We propose two approaches: eliminating cases where one point corresponds to multiple points, and eliminating cases of obvious mispairings.

1) Eliminate the situation where one point corresponds to multiple points: Find pairs that correspond to multiple points, compare the results of normalized cross correlation, delete matching pairs with a smaller threshold, and ensure that the obtained pairs are all in one-to-one correspondence.

2) Eliminating cases of obvious mispairings: Because the image stitching of the left and right images is performed, the Euclidean distance between the two corresponding points should not be too large, and the absolute value of the slope of the connection should not be too large. For this, we set two thresholds: slope_treshold, dis_treshold. Among them, the slope_treshold value is 6, that is, the pairs that discard the connection line greater than 9.46°. The value of dis_threshold is 0.4*height*width.

The result after processing is shown in the figure.

2.4 Homography

A Homography is a transformation (a 3×3 matrix)that maps the points in one image to the corresponding points in the other image.

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \Leftrightarrow p_2 = Hp_1$$

Multiplicative expansion, we can get

$$\begin{cases} x_2 = h_{11}x_1 + h_{12}y_1 + h_{13} \\ y_2 = h_{21}x_1 + h_{22}y_1 + h_{23} \\ 1 = h_{31}x_1 + h_{32}y_1 + h_{33} \end{cases}$$

To transform the above formula into the form of A x = b Ax=bAx=b, the following transformation is required:

$$\begin{cases} x_2(h_{31}x_1 + h_{32}y_1 + h_{33}) = h_{11}x_1 + h_{12}y_1 + h_{13} \\ y_2(h_{31}x_1 + h_{32}y_1 + h_{33}) = h_{21}x_1 + h_{22}y_1 + h_{23} \end{cases}$$

Transpose, change the right-hand expression to 0

$$\begin{cases} x_2(h_{31}x_1 + h_{32}y_1 + h_{33}) - h_{11}x_1 + h_{12}y_1 + h_{13} = 0 \\ y_2(h_{31}x_1 + h_{32}y_1 + h_{33}) - h_{21}x_1 + h_{22}y_1 + h_{23} = 0 \end{cases}$$

Change the above formula into vector product form, That is, A ∗H = 0, where

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_2 & -y_1x_2 & -x_2 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_2 & -y_1y_2 & -x_2 \end{bmatrix}$$

In order to solve H, there are 8 unknowns in H, so 8 equations are needed to solve, which requires 4 pairs of matching points (that is, 4 points in the original image)

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_2 & -y_1x_2 & -x_2 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_2 & -y_1y_2 & -x_2 \\ x_3 & y_3 & 3 & 0 & 0 & 0 & -x_3x_4 & -y_3x_4 & -x_4 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_4 & -y_3y_4 & -x_4 \\ x_5 & y_5 & 1 & 0 & 0 & 0 & -x_5x_6 & -y_5x_6 & -x_6 \\ 0 & 0 & 0 & x_5 & y_5 & 1 & -x_5y_6 & -y_5y_6 & -x_6 \\ x_7 & y_7 & 1 & 0 & 0 & 0 & -x_7x_8 & -y_7x_8 & -x_8 \\ 0 & 0 & 0 & x_7 & y_7 & 1 & -x_7y_8 & -y_7y_8 & -x_8 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ 1 \end{bmatrix} = \begin{bmatrix} x_2 \\ y_2 \\ x_4 \\ y_4 \\ x_6 \\ y_6 \\ x_8 \\ y_8 \end{bmatrix}$$

Linear equations $\overset{2Nx8}{\mathbf{A}}\ \overset{8x1}{\mathbf{h}} = \overset{2Nx1}{\mathbf{b}}$

Solve:

$$\overset{8x2N}{\mathbf{A}^T}\ \overset{2Nx8}{\mathbf{A}}\ \overset{8x1}{\mathbf{h}} = \overset{8x2N}{\mathbf{A}^T}\ \overset{2Nx1}{\mathbf{b}}$$

$$\overset{8x8}{(\mathbf{A}^T\ \mathbf{A})}\ \overset{8x1}{\mathbf{h}} = \overset{8x1}{(\mathbf{A}^T\ \mathbf{b})}$$

$$\mathbf{h} = (\mathbf{A}^T\ \mathbf{A})^{-1}\ (\mathbf{A}^T\ \mathbf{b})$$

Matlab: $\mathbf{h} = \mathbf{A} \setminus \mathbf{b}$

However, in real scenes, there is often a lot of noise. In order to calculate accurately, we usually use far more than 4 points for calculation. Here we have chosen 8 points for calculation.

2.5 RANSAC

Randomly select a part of the given data (randomly draw 4 samples from the matching data set and ensure that the four samples are not collinear) to calculate the model parameters, and then use all point pairs to evaluate the calculation results, and iterate continuously until The selected data calculates the lowest error rate.

Proceed as follows

(1) Select the least random point pairs required to solve the model

(2) According to the selected random point pair to solve/fit the model, get the parameters

(3) According to the model parameters, evaluate all point pairs, divided into outlier and inlier

(4) If the number of all inliers exceeds the predefined threshold, use all inliers to re-evaluate the model parameters and stop the iteration

(5) If the conditions are not met, continue to cycle 1~4.

In the case of our model, we select eight points from the acquired matching pairs to calculate the homography matrix each time, and set the threshold, consider points with an error smaller than the threshold as inliers, and iterate 2000 times, and select the homography with the highest proportion of inliers as the final matrix.

2.6 Synthesis of new images

Before image projection, a blank canvas must be created first. Compare the upper, lower, left, and right boundaries of the two-dimensional coordinates of the two images after projection, and select the maximum value of the boundaries in each direction as the size of the new image. Before image projection, a blank canvas must be created first. Compare the upper, lower, left, and right boundaries of the two-dimensional coordinates of the two images after projection, and select the maximum value of the boundaries in each direction as the size of the new image. The left image to be spliced is obtained according to the transformation matrix to the spliced image, and the blank area between the images is filled by an interpolation method to ensure that the image is coherent.

First, calculate the part where the two images will overlap. For the pixel value of this part, the method of cross dissolve is used to fill the pixels in this area, and finally we can get our stitching result.

3 Results

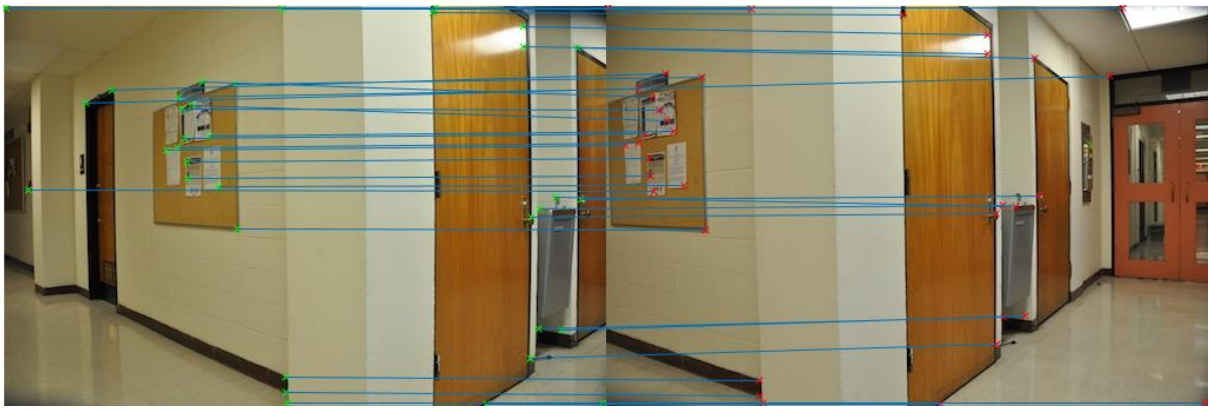
First, we use the harris corner detection method to perform feature detection on the two images. Among them, we only select the 100 corner points with the highest value as feature points, and the results are shown in the following figure:

Then use normalized cross correlation (NCC) to calculate the NCC value of each

feature point, and find the corresponding pairs in the two images. In this step, The comparison window size is 3*3, Find the corresponding point in graph a for each corner point in graph b, and then find the corresponding point in graph b for the corner point that has been determined to have a corresponding point in graph a. This ensures that each pair of matching points is the best result of the two-way matching, and to some extent exclude outlier values

Because there are a large number of false matches, we exclude most of the false matches based on the slope and Euclidean distance of the line connecting the corresponding points. Among them, the slope treshold is set to 9.46 ° , The value of distance threshold is 0.4*height*width.

Here is the result:



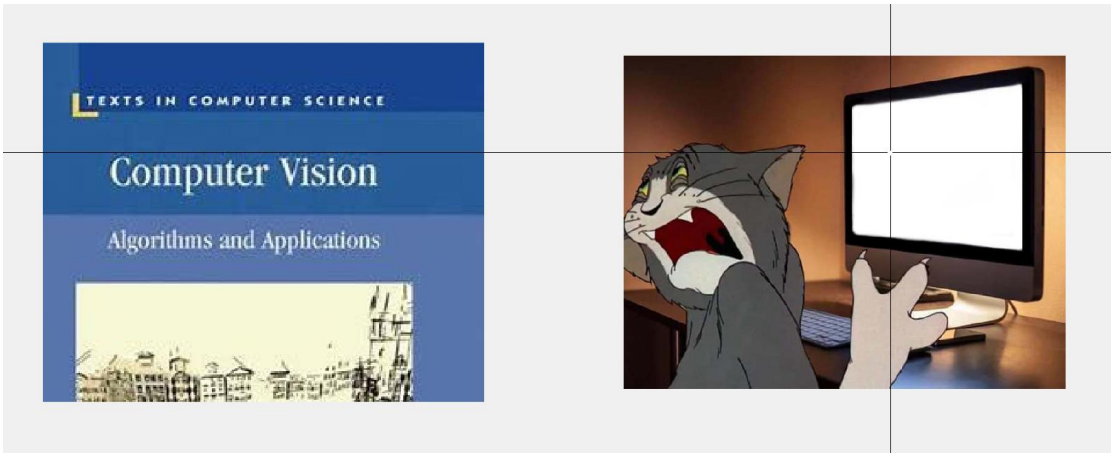
Then we select eight points from the acquired matching pairs to calculate the homography matrix each time, and set the threshold, consider points with an error smaller than the threshold as inliers, and iterate 2000 times, and select the homography with the highest proportion of inliers as the final matrix.

Create a blank canvas by calculating the size of final result by using homography and warped image. Then draw the warped image into that canvas and what we need to do is just combine these two images together:


Finally, calculate the part where the two images will overlap. For the pixel value of this part, the method of cross dissolve is used to fill the pixels in this area, and finally we can get our stitching result:

4 Extra Credits

We first obtain the four coordinates of the warped image, and then use the ginput method to obtain the four position coordinates of the plane to be pasted in clockwise order in the unwarped image (the coordinates of the upper left corner are used as the starting point).



Then calculate the homograpy matrix according to the coordinates of these four pairs of matching, then do the transformation of the warped image, and finally paste the result to the corresponding position of the unwarped image to obtain the final result picture, as shown below.

Appendix

Matlab Code

```
%%
main-----------------------------------------------------------------
---------------------------------
% Matching of feature points, mainly using the detection of harris corner points, match
one-way matching function
% is suitable for images with white edges, because there is no limited range when
windowing and filtering, try to ensure that the corners are not on the edge
clc,clear all;
a1=imread('DSC_0281.JPG');
a2=imread('DSC_0282.JPG'); % double plays a big role
[ysize, xsize] = size(a1);

i_fig=1; % figure number

[result1,cnt1,r1,c1]=harris(a1);% Corner detection, get the original focus position map result
[result2,cnt2,r2,c2]=harris(a2);
figure(i_fig);subplot(1,2,1);imshow(a1);hold on;plot(c1,r1,'g.');
subplot(1,2,2);imshow(a2);hold on;plot(c2,r2,'g.'); title('Corner plots of Figures 1 and 2');
```

```
i_fig=i_fig+1;


[res2,ans2]=match(a1,cnt1,r1,c1,a2,cnt2,r2,c2);% Starting from result1 to search for
possible reach in result2
[r22,c22]=find(res2==1);
[m22,n22]=size(r22);
cnt22=m22;


[res1,ans1]=match(a2,cnt22,r22,c22,a1,cnt1,r1,c1);% reverse search res2--result1
res1=and(res1,result1); % Guaranteed reverse matching against impossible points
[r11,c11]=find(res1==1);
[m11,n11]=size(r11);
cnt11=m11;


respic = [a1,a2];
[rw,~] = size(ans1);


figure(i_fig);
imshow(respic);hold on;plot(ans1(:,2)+512,ans1(:,1),'rx');plot(ans1(:,4),ans1(:,3),'gx');
i_fig=i_fig+1;


for i=1:rw
```

```
% if ans1(i, 5)>0.985 %Set thresholds for corresponding points to remove wrong pairs
line([ans1(i,2)+512,ans1(i,4)],[ans1(i,1),ans1(i,3)])
% end
end
title('The resulting matching corners');

ans1 = remove_same_pairs(ans1);
ans1 = remove_false(ans1, xsize*ysize);
[rw,D] = size(ans1);

figure(i_fig);
imshow(respic);hold on;plot(ans1(:,2)+512,ans1(:,1),'rx');plot(ans1(:,4),ans1(:,3),'gx');
i_fig=i_fig+1;

for i=1:rw
% if ans1(i, 5)>0.985 % Set thresholds for corresponding points to remove wrong pairs
line([ans1(i,2)+512,ans1(i,4)],[ans1(i,1),ans1(i,3)])
% end
end
title('The resulting matching corners');

match_a1=[ans1(:,2),ans1(:,1),ones(rw,1)]';
```

```
match_a2=[ans1(:,4),ans1(:,3),ones(rw,1)]';

[hh, inliers] = ransacfithomography(match_a1, match_a2, rw, 10);

[height_wrap, width_wrap,~] = size(a1);

[height_unwrap, width_unwrap,~] = size(a2);

% USE INVERSE WARP METHOD

% DETERMINE THE SIZE OF THE WHOLE IMAGE

[newH, newW, newX, newY, xB, yB] = getNewSize(hh, height_wrap, width_wrap,

height_unwrap, width_unwrap);

[X,Y] = meshgrid(1:width_wrap,1:height_wrap);

[XX,YY] = meshgrid(newX:newX+newW-1, newY:newY+newH-1);

AA = ones(3,newH*newW);

AA(1,:) = reshape(XX,1,newH*newW);

AA(2,:) = reshape(YY,1,newH*newW);

AA = hh*AA;

XX = reshape(AA(1,:)./AA(3,:), newH, newW);

YY = reshape(AA(2,:)./AA(3,:), newH, newW);

% INTERPOLATION, WARP IMAGE A INTO NEW IMAGE
```

```
newImage(:,:,1) = interp2(X, Y, double(a1(:,:,1)), XX, YY);

newImage(:,:,2) = interp2(X, Y, double(a1(:,:,2)), XX, YY);

newImage(:,:,3) = interp2(X, Y, double(a1(:,:,3)), XX, YY);


% BLEND IMAGE BY CROSS DISSOLVE

[newImage] = blend(newImage, a2, xB, yB);


% DISPLAY IMAGE MOSIAC

figure(i_fig);

imshow(uint8(newImage));

i_fig=i_fig+1;


%% harris corner detection

function [y1,y2,r,c]=harris(X)

% Corner detection, using the Harris algorithm

% The output is an image

% [result,cnt,r,c]=harris(X)


% f=rgb2gray(X);

f=X;


if(size(f,3)==3)
```

```
ori_im = rgb2gray(uint8(f)); % Convert to grayscale image

end


%ori_im=double(f)/255; % uint8 converted to 64-bit double precision double 64


fx = [5 0 -5;8 0 -8;5 0 -5]; % Gaussian function first order differential, x direction

%fx = [-2 -1 0 1 2]; % Gradient operator in x direction

Ix = filter2(fx,ori_im); % Filtering in the x direction

fy = [5 8 5;0 0 0;-5 -8 -5]; % Gaussian function first order differential, y direction

%fy = [-2;-1;0;1;2]; % Gradient operator in y direction

Iy = filter2(fy,ori_im); % Filtering in the y direction


Ix2 = Ix.^2;

Iy2 = Iy.^2;

Ixy = Ix.*Iy;

clear Ix;

clear Iy;

h= fspecial('gaussian',[10 10 ],2); % Generate a 7*7 Gaussian window function, sigma=2

Ix2 = filter2(h,Ix2);

Iy2 = filter2(h,Iy2);

Ixy = filter2(h,Ixy); % Gaussian filtering

height = size(ori_im,1);
```

```
width = size(ori_im,2);

result = zeros(height,width); % Record the corner position, the value at the corner is 1, and

the background is black

R = zeros(height,width);


for i = 1:height

for j = 1:width

M = [Ix2(i,j) Ixy(i,j);Ixy(i,j) Iy2(i,j)]; % auto correlation matrix

R(i,j) = det(M)-0.06*(trace(M))^2;

end

end


% Select the corner point according to the maximum value in the window

cnt = 0;

for i = 2:height-1

for j = 2:width-1

% Perform non-maximum suppression, window size 3*3

if R(i,j) > R(i-1,j-1) && R(i,j) > R(i-1,j) && R(i,j) > R(i-1,j+1) && R(i,j) > R(i,j-1) && R(i,j) >

R(i,j+1) && R(i,j) > R(i+1,j-1) && R(i,j) > R(i+1,j) && R(i,j) > R(i+1,j+1)

result(i,j) = 1;

cnt = cnt+1;

end
```

```
end

end


Rsort=zeros(cnt,1);

[posr, posc] = find(result == 1); % Return corner coordinates

for i=1:cnt

Rsort(i)=R(posr(i),posc(i));

end


% Sort all corner R values and select the largest 100 as output

[Rsort,ix]=sort(Rsort,1);

Rsort=flipud(Rsort);

ix=flipud(ix);

ps=300;

posr2=zeros(ps,1);

posc2=zeros(ps,1);

for i=1:ps

posr2(i)=posr(ix(i));

posc2(i)=posc(ix(i));

end

y1=result;

y2=ps;
```

```
r=posr2;c=posc2;

return;

end

%% pairs match

function [res,ans_]=match(A1,cnt1,r1,c1,A2,cnt2,r2,c2)

% res=match(a1,a2)

% The best matching point in a2 will be found from a1, and the res extracted from a2 will

be obtained, that is, one-way search


if(size(A1,3)==3)

a1 = rgb2gray(uint8(A1));

end


if(size(A2,3)==3)

a2 = rgb2gray(uint8(A2));

end

a1=double(a1);

a2=double(a2);


% [m1,n1]=size(a1);

[m2,n2]=size(a2);

% res1=zeros(m1,n1);
```

```
res2=zeros(m2,n2); % match point

ans_=zeros(100,5);

for s=1:cnt1

max=0; p=0;q=0;i=r1(s,1);j=c1(s,1); % p.q store coordinates

for v=1:cnt2

m=r2(v,1);n=c2(v,1);

mask_size=1;% 3*3 mask

if i+mask_size<340 && i-mask_size>0 && m+mask_size<340 && m-mask_size>0 &&

n+mask_size<340 && n-mask_size>0 && j+mask_size<340 && j-mask_size>0

else

mask_size=1;

end


a1_norm = (sum(sum(a1(i-mask_size:i+mask_size,j-mask_size:j+mask_size).^2))).^0.5;

a2_norm =

(sum(sum(a2(m-mask_size:m+mask_size,n-mask_size:n+mask_size).^2))).^0.5;

ncc=sum(sum((a1(i-mask_size:i+mask_size,j-mask_size:j+mask_size)/a1_norm).*(a2(m-

mask_size:m+mask_size,n-mask_size:n+mask_size)/a2_norm)));

if ncc>max

max=ncc;

p=m;

q=n;
```

```
end

end

res2(p,q)=1;

ans_(s,:)=[i,j,p,q,max];

end


res=res2;

return

end


%% remove same pairs

function newres = remove_same_pairs(res)

% res = [left_row, left_col, right_row, right_col, score]

total = 0;

for i=1:100

if res(i, 5)~=0

total = total+1;

end

end


count = 0;

for s=1:total
```

```
p = res(s,3); q = res(s,4);
for i=1:total
if i ~=s
r = res(i,3); c = res(i,4);
if p==r && q ==c
if res(i, 5)>res(s,5)
if res(s, 5) ~= 0
res(s, 5) = 0;
count =count+1;
end
else
if res(i, 5) ~= 0
res(i, 5) = 0;
count =count+1;
end
end
end
end
end
end
end

newres = zeros(total-count, 5);
```

```
k=1;
for i = 1:100
if res(i, 5) ~= 0
newres(k, :) = res(i, :);
k = k+1;
end
end
end

%% remove false pairs by slope and Euclidean distance
function newres = remove_false(res, imgsize)
% input
% res = [left_row, left_col, right_row, right_col, score]
% imgsize = img_height*img_width
[row, ~] = size(res);
dis = zeros(row,1);
slope = zeros(row,1);
for i = 1:row
slope(i) = abs((res(i, 4)-res(2))/(res(i,3)-res(i,1)));
dis(i) = sqrt((res(i,4)-res(i,2))^2+(res(i,3)-res(i,1))^2);
end
```

```
dis_threshold = imgsize*0.4;

slope_threshold = 6; %20°

count=0;

for i = 1:row

if dis(i)>dis_threshold || slope(i)<slope_threshold

res(i,5) = 0;

count =count+1;

end

end



newres = zeros(row-count, 5);

k=1;

for i =1:row

if res(i, 5) ~= 0

newres(k,:) = res(i,:);

k = k+1;

end

end



end
```

```
%%

function [hh] = getHomographyMatrix(point_ref, point_src, npoints)

% Use corresponding points in both images to recover the parameters of the

transformation

% Input:

% x_ref, x_src --- x coordinates of point correspondences

% y_ref, y_src --- y coordinates of point correspondences

% Output:

% h --- matrix of transformation


% NUMBER OF POINT CORRESPONDENCES

x_ref = point_ref(1,:)';

y_ref = point_ref(2,:)';

x_src = point_src(1,:)';

y_src = point_src(2,:)';


% COEFFICIENTS ON THE RIGHT SIDE OF LINEAR EQUATIONS

A = zeros(npoints*2,8);

A(1:2:end,1:3) = [x_ref, y_ref, ones(npoints,1)];

A(2:2:end,4:6) = [x_ref, y_ref, ones(npoints,1)];

A(1:2:end,7:8) = [-x_ref.*x_src, -y_ref.*x_src];

A(2:2:end,7:8) = [-x_ref.*y_src, -y_ref.*y_src];
```

```
% COEFFICIENT ON THE LEFT SIDE OF LINEAR EQUATIONS

B = [x_src, y_src];

B = reshape(B',npoints*2,1);


% SOLVE LINEAR EQUATIONS

h = A\B;


hh = [h(1),h(2),h(3);h(4),h(5),h(6);h(7),h(8),1];

end


%%

function [hh, inliers] = ransacfithomography(ref_P, dst_P, npoints, threshold)

% 8-point RANSAC fitting

% Input:

% matcher_A - match points from image A, a matrix of 3xN, the third row is 1

% matcher_B - match points from image B, a matrix of 3xN, the third row is 1

% thd - distance threshold

% npoints - number of samples


ninlier = 0;

fpoints = 8; %number of fitting points
```

```
for i=1:2000

rd = randi([1 npoints],1,fpoints);

pR = ref_P(:,rd);

pD = dst_P(:,rd);

h = getHomographyMatrix(pR,pD,fpoints);

rref_P = h*ref_P;

rref_P(1,:) = rref_P(1,:)./rref_P(3,:);

rref_P(2,:) = rref_P(2,:)./rref_P(3,:);

error = (rref_P(1,:) – dst_P(1,:)).^2 + (rref_P(2,:) – dst_P(2,:)).^2;

n = nnz(error<threshold);

if(n >= npoints*.95)

hh=h;

inliers = find(error<threshold);

pause();

break;

elseif(n>ninlier)

ninlier = n;

hh=h;

inliers = find(error<threshold);

end

end

end
```

```
%%

function [newH, newW, x1, y1, x2, y2] = getNewSize(transform, h2, w2, h1, w1)

% Calculate the size of new mosaic

% Input:

% transform – homography matrix

% h1 – height of the unwarped image

% w1 – width of the unwarped image

% h2 – height of the warped image

% w2 – height of the warped image

% Output:

% newH – height of the new image

% newW – width of the new image

% x1 – x coordate of lefttop corner of new image

% y1 – y coordate of lefttop corner of new image

% x2 – x coordate of lefttop corner of unwarped image

% y2 – y coordate of lefttop corner of unwarped image



% CREATE MESH-GRID FOR THE WARPED IMAGE

[X,Y] = meshgrid(1:w2,1:h2);

AA = ones(3,h2*w2);

AA(1,:) = reshape(X,1,h2*w2);
```

```
AA(2,:) = reshape(Y,1,h2*w2);


% DETERMINE THE FOUR CORNER OF NEW IMAGE

newAA = transform\AA;

new_left = fix(min([1,min(newAA(1,:)./newAA(3,:))]));

new_right = fix(max([w1,max(newAA(1,:)./newAA(3,:))]));

new_top = fix(min([1,min(newAA(2,:)./newAA(3,:))]));

new_bottom = fix(max([h1,max(newAA(2,:)./newAA(3,:))]));


newH = new_bottom - new_top + 1;

newW = new_right - new_left + 1;

x1 = new_left;

y1 = new_top;

x2 = 2 - new_left;

y2 = 2 - new_top;

end


%%

function [newImage] = blend(warped_image, unwarped_image, x, y)

% Blend two image by using cross dissolve

% Input:

% warped_image - original image
```

```
% unwarped_image – the other image

% x – x coordinate of the lefttop corner of unwarped image

% y – y coordinate of the lefttop corner of unwarped image

% Output:

% newImage


% MAKE MASKS FOR BOTH IMAGES

warped_image(isnan(warped_image))=0;

maskA = (warped_image(:,:,1)>0 |warped_image(:,:,2)>0 | warped_image(:,:,3)>0);

newImage = zeros(size(warped_image));

newImage(y:y+size(unwarped_image,1)–1, x: x+size(unwarped_image,2)–1,:) =

unwarped_image;

mask = (newImage(:,:,1)>0 | newImage(:,:,2)>0 | newImage(:,:,3)>0);

mask = and(maskA, mask);


% GET THE OVERLAID REGION

[~,col] = find(mask);

left = min(col);

right = max(col);

mask = ones(size(mask));


if( x<2)
```

```
mask(:,left:right) = repmat(linspace(0,1,right-left+1),size(mask,1),1);
else
mask(:,left:right) = repmat(linspace(1,0,right-left+1),size(mask,1),1);
end

% BLEND EACH CHANNEL
warped_image(:,:,1) = warped_image(:,:,1).*mask;
warped_image(:,:,2) = warped_image(:,:,2).*mask;
warped_image(:,:,3) = warped_image(:,:,3).*mask;

% REVERSE THE ALPHA VALUE
if( x<2)
mask(:,left:right) = repmat(linspace(1,0,right-left+1),size(mask,1),1);
else
mask(:,left:right) = repmat(linspace(0,1,right-left+1),size(mask,1),1);
end
newImage(:,:,1) = newImage(:,:,1).*mask;
newImage(:,:,2) = newImage(:,:,2).*mask;
newImage(:,:,3) = newImage(:,:,3).*mask;

newImage(:,:,1) = warped_image(:,:,1) + newImage(:,:,1);
newImage(:,:,2) = warped_image(:,:,2) + newImage(:,:,2);
```

```
newImage(:,:,3) = warped_image(:,:,3) + newImage(:,:,3);

end


%% extra credits
clear;
clc;

a1=imread('book_size.jpg');
a2=imread('tom_computer.png'); % double plays a big role
a=get(0);
figure('position',a.MonitorPositions);
subplot(1,2,1);imshow(a1);
subplot(1,2,2);imshow(a2);

num_points = 4;
[x,y] = ginput(num_points);
close all;
% reshape(x(1:8),1,8),ones(1,8);
point_ref =
[reshape(x(1:num_points),1,num_points);reshape(y(1:num_points),1,num_points);ones(1,num_points)]
;
[h, w,~] = size(a1);
point1=[1,1,1];
point2=[w,1,1];
point3=[w,h,1];
point4=[1,h,1];
point_src=[point1;point2;point3;point4]';

hh = getHomographyMatrix(point_ref,point_src,num_points);

[height_wrap, width_wrap,~] = size(a1);
[height_unwrap, width_unwrap,~] = size(a2);

[newH, newW, newX, newY, xB, yB] = getNewSize(hh, height_wrap, width_wrap, height_unwrap,
width_unwrap);

[X,Y] = meshgrid(1:width_wrap,1:height_wrap);
[XX,YY] = meshgrid(newX:newX+newW-1, newY:newY+newH-1);
AA = ones(3,newH*newW);
AA(1,:) = reshape(XX,1,newH*newW);
AA(2,:) = reshape(YY,1,newH*newW);
```

```
AA = hh*AA;
XX = reshape(AA(1,:)./AA(3,:), newH, newW);
YY = reshape(AA(2,:)./AA(3,:), newH, newW);

% INTERPOLATION, WARP IMAGE A INTO NEW IMAGE
newImage(:,:,1) = interp2(X, Y, double(a1(:,:,1)), XX, YY);
newImage(:,:,2) = interp2(X, Y, double(a1(:,:,2)), XX, YY);
newImage(:,:,3) = interp2(X, Y, double(a1(:,:,3)), XX, YY);

% BLEND IMAGE BY CROSS DISSOLVE
[newImage] = blend(newImage, a2, xB, yB);
figure('position',a.MonitorPositions);
subplot(1,3,1);imshow(a1);
subplot(1,3,2);imshow(a2);
subplot(1,3,3);imshow(uint8(newImage));
%%
function [hh] = getHomographyMatrix(point_ref, point_src, npoints)
% Use corresponding points in both images to recover the parameters of the transformation
% Input:
% x_ref, x_src --- x coordinates of point correspondences
% y_ref, y_src --- y coordinates of point correspondences
% Output:
% h --- matrix of transformation

% NUMBER OF POINT CORRESPONDENCES
x_ref = point_ref(1,:)';
y_ref = point_ref(2,:)';
x_src = point_src(1,:)';
y_src = point_src(2,:)';

% COEFFICIENTS ON THE RIGHT SIDE OF LINEAR EQUATIONS
A = zeros(npoints*2,8);
A(1:2:end,1:3) = [x_ref, y_ref, ones(npoints,1)];
A(2:2:end,4:6) = [x_ref, y_ref, ones(npoints,1)];
A(1:2:end,7:8) = [-x_ref.*x_src, -y_ref.*x_src];
A(2:2:end,7:8) = [-x_ref.*y_src, -y_ref.*y_src];

% COEFFICIENT ON THE LEFT SIDE OF LINEAR EQUATIONS
B = [x_src, y_src];
B = reshape(B',npoints*2,1);

% SOLVE LINEAR EQUATIONS
h = A\B;
```

```
hh = [h(1),h(2),h(3);h(4),h(5),h(6);h(7),h(8),1];
end

%%
function [newH, newW, x1, y1, x2, y2] = getNewSize(transform, h2, w2, h1, w1)
% Calculate the size of new mosaic
% Input:
% transform - homography matrix
% h1 - height of the unwarped image
% w1 - width of the unwarped image
% h2 - height of the warped image
% w2 - height of the warped image
% Output:
% newH - height of the new image
% newW - width of the new image
% x1 - x coordate of lefttop corner of new image
% y1 - y coordate of lefttop corner of new image
% x2 - x coordate of lefttop corner of unwarped image
% y2 - y coordate of lefttop corner of unwarped image

% CREATE MESH-GRID FOR THE WARPED IMAGE
[X,Y] = meshgrid(1:w2,1:h2);
AA = ones(3,h2*w2);
AA(1,:) = reshape(X,1,h2*w2);
AA(2,:) = reshape(Y,1,h2*w2);

% DETERMINE THE FOUR CORNER OF NEW IMAGE
newAA = transform\AA;
new_left = fix(min([1,min(newAA(1,:)./newAA(3,:))]));
new_right = fix(max([w1,max(newAA(1,:)./newAA(3,:))]));
new_top = fix(min([1,min(newAA(2,:)./newAA(3,:))]));
new_bottom = fix(max([h1,max(newAA(2,:)./newAA(3,:))]));

newH = new_bottom - new_top + 1;
newW = new_right - new_left + 1;
x1 = new_left;
y1 = new_top;
x2 = 2 - new_left;
y2 = 2 - new_top;
end


%%
function [newImage] = blend(warped_image, unwarped_image, x, y)
```

```
% Blend two image by using cross dissolve
% Input:
% warped_image - original image
% unwarped_image - the other image
% x - x coordinate of the lefttop corner of unwarped image
% y - y coordinate of the lefttop corner of unwarped image
% Output:
% newImage

% MAKE MASKS FOR BOTH IMAGES
warped_image(isnan(warped_image))=0;
maskA = (warped_image(:,:,1)>0 |warped_image(:,:,2)>0 | warped_image(:,:,3)>0);
newImage = zeros(size(warped_image));
newImage(y:y+size(unwarped_image,1)-1, x: x+size(unwarped_image,2)-1,:) = unwarped_image;

% overlap part paint black
newImage(:,:,1) = uint8(newImage(:,:,1)- 1000*warped_image(:,:,1));
newImage(:,:,2) = uint8(newImage(:,:,2)- 1000*warped_image(:,:,2));
newImage(:,:,3) = uint8(newImage(:,:,3)- 1000*warped_image(:,:,3));

newImage(:,:,1) = warped_image(:,:,1) + newImage(:,:,1);
newImage(:,:,2) = warped_image(:,:,2) + newImage(:,:,2);
newImage(:,:,3) = warped_image(:,:,3) + newImage(:,:,3);

end
```