Chapter 8
Image Stitching

Abstract In this chapter we firstly introduce the application background and basic process of image stitching, then depict several image stitching methods based on region, image stitching methods based on feature points, and panoramic video image stitching techniques.

8.1 Introduction

In practice, it often needs the wide-view and high-resolution panoramic images, but the size of the image depends on the performance of the camera due to the limitation of the imaging device. Therefore, an approach which utilizes the computer software to stitch images was then raised for making panoramas. Image stitching refers to put several images with overlapping parts together into a large, seamless and high-resolution image. Figure 8.1 shows the sketch map of image stitching.

Generally, image stitching mainly includes the following five steps:

(1) Image preprocessing. It contains basic operations of digital image processing (such as denoising, edge extraction and histogram processing), establishment of image matching templates, image transforms (FT, WT, etc.) and other operations.
(2) Image registration. It adopts some kinds of matching algorithms to find the corresponding positions of the templates or feature points in stitching images so as to determine the transformation relation between two images.
(3) Build the transform model. A mathematical transform model can be built between two images by calculating parameters of the model based on the correspondences of image templates or features.
(4) Unified coordinate transformation. In accordance with the mathematical transform model built in step 3, the image to be stitched will be transferred into the coordinate system of the reference image in order to accomplish the unified coordinate transformation.

S. Gong et al., *Advanced Image and Video Processing Using MATLAB*,
Modeling and Optimization in Science and Technologies 12,
https://doi.org/10.1007/978-3-319-77223-3_8

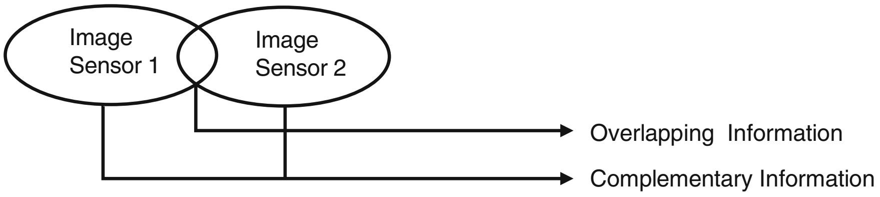


Fig. 8.1 Sketch map of image stitching

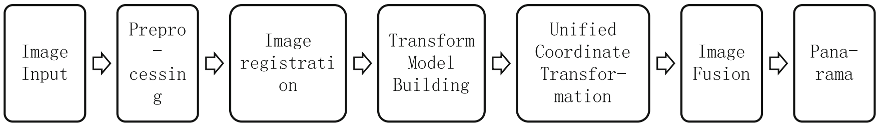


Fig. 8.2 Flowchart of image stitching

(5) Image fusion and reconstruction. Merging the overlapping portions of the images to be stitched to a smooth and seamless reconstructed panorama.

Figure 8.2 shows the basic flowchart of image stitching.

Image registration is the key to image stitching algorithms. According to different image registration methods, the image stitching algorithms can be classified into two categories: image stitching based on region and image stitching based on feature points.

8.2 Image Stitching Based on Region

Image stitching based on region starts from comparing the grayscale values of an area in image to be stitched with the area in referenced image which have the same size by the least squares methods and other mathematical methods. From the comparisons we can measure the similarity between the overlapping areas in images to be stitched, and get the range and position of the overlapping area in the image to be stitched to accomplish the image stitching task. We can also transform the images from spatial domain into frequency domain with FFT and operate the image registration later. To the images with large displacement, we can correct the rotation of the image and then establish the mapping between two images. When take the difference between the grayscale values of pixels in two regions as criterion, the simplest approach is directly adding up the differences pixel by pixel. Another way is to calculate the correlation coefficient between the pixel grayscale values of the two areas. The larger the correlation coefficient is, the higher the matching degree of the two images will be, and this way shows better performances as well as a higher success rate. Nowadays, the commonly used image stitching algorithms based on region include Ratio Matching, Block-based Matching, Line Matching and Grid Matching.

8.2.1 Image Stitching Based on Ratio Matching

Image stitching based on ratio matching first selects the ration of two columns of pixels with a certain distance between the overlapped parts of the image as a template [1]. Then search the best match for the overlapped region in second image and find the two columns corresponding to the template taken from the first image to complete the image stitching. Figure 8.3 is a sketch map of the algorithm. Picture 1 stands for an $(W_1 \times H)$ image in pixels and Picture 2 is a $(W_2 \times H)$ one. W_1 and W_2 may be equal or not. Picture 1 is on the left of Picture 2. Another situation that images are vertical overlapped will not be discussed in this chapter since we can handle it in a similar way.

Following are the steps of this algorithm:

(1) Select two columns of pixels with the interval *span* from overlapped area of Picture1, calculate the corresponding pixel ratio as template a.

$$a(i,j) = \frac{P_1(i,j)}{P_1(i,j+span)}, \quad i \in (1,H) \tag{8.1}$$

(2) In Picture 2, each two columns with the interval of span are selected in turn from the first column, the ratio of its corresponding pixels is calculated as template b.

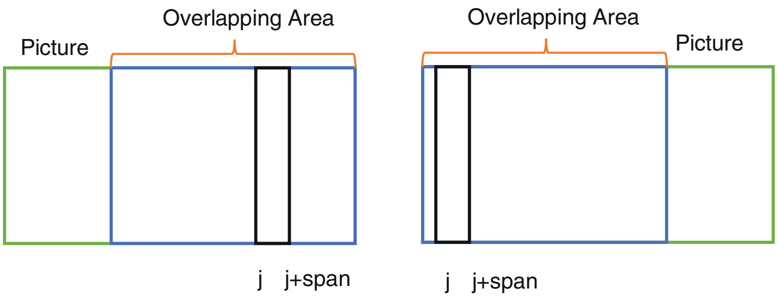


Fig. 8.3 Sketch map of template choosing

$$b(i,j) = \frac{P_{21}(i,j)}{P_{22}(i,j)} \tag{8.2}$$

$$P_{21}(i,j) = P_2(i,j), (i \in (1,H), j \in (1,\ W_2 - span))$$
$$P_{22}(i,j) = P_2(i,j), (i \in (1,H), j \in (1,\ W_2 - span))$$

(3) Calculate the differences between template a and b as template c.

$$c(i,j) = (a(i,j) - b(i,j))^2, \quad i \in (1,H), j \in (1,\ W_2 - span) \tag{8.3}$$

(4) c is a two-dimensional array. Add each column vector up into another array called sum: $sum(j) = \sum_{i=1}^{H} c(i,j)$. The value of $sum(j)$ reflects the difference of selected columns in two images. The column coordinates of $sum\,(j)$'s minimum value $sum_{\min}$ are the best match.

PROGRAMME 8.1 is the code of image stitching based on ratio matching.

PROGRAMME 8.1: Image stitching based on ratio matching

```
clear;
clc;
A=imread('lenna_left.jpg');% read Picture 1， A represents the pixel array of Picture 1
B=imread('lenna_right.jpg');% read Picture 2
[x1,y1]=size(A(:,:,1));
% transform into greyscale images， calculate the length and height of Picture 1
[x2,y2]=size(B(:,:,1));
A1=double(A);
B1=double(B);
sub_A=A1(:,end-1)./A1(:,end);%calculate the ratio of last two columns in Picture 1
sub_D = zeros(size(B,2)-1,2);%define sub_D
for y=1:y2-1
    sub_B=B1(:,y)./B1(:,y+1);% calculate the ratio of two adjacent columns in Picture 2
sub_C=(sub_A-sub_B)'*(sub_A-sub_B);
%calculate the difference between template a and b, calculate the sum of column vector
    sub_D(y,1)=y;
    sub_D(y,2) =sub_C;%sub_D is a two-dimensional array
end
```

```
[a b]= sort(sub_D(:,2));% ascending sort
row = b(1,:);% the coordinate of first element is the best match
x3=x1;
y3 = y1-1+y2-row;%length and height of image to be stitched
C=zeros(x3,y3);
for i=1:x3
    for j=1:y3
      if j<y1
          C(i,j)=A(i,j);
      else
          C(i,j)=B(i,row+j-y1);
      end
    end
end% image stitching
imwrite(C,'picture3.bmp');
imshow(mat2gray(C));
```

In order to confirm the effectiveness of image stitching based on ratio matching, simulation experiments are carried out for two images with overlapped regions. Figure 8.4 shows the result.

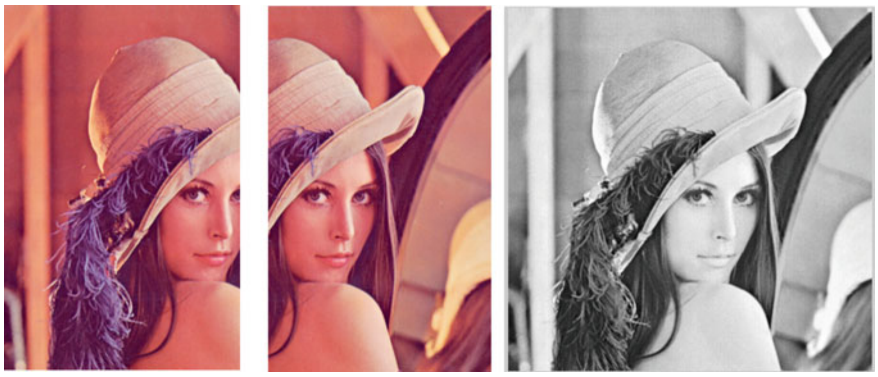


Fig. 8.4 Input and output of experiment

8.2.2 *Image Stitching Based on Line and Plane Feature*

Image stitching based on line and plane feature mainly includes: image preprocessing, feature block searching, image stitching and image fusion. Figure 8.5 shows the flowchart.

(1) Image preprocessing. Because of the different illumination, it is easy to make stitching errors if the raw images obtained directly from the camera are stitched. Histogram equalization is an effective way to alleviate the effects of illumination. After applying histogram equalization to the two images to be stitched, the grayscale histograms of two images are spread into all gray level ranges and the difference of illumination in adjacent images is reduced efficiently, which will make the image stitching easy to realize.

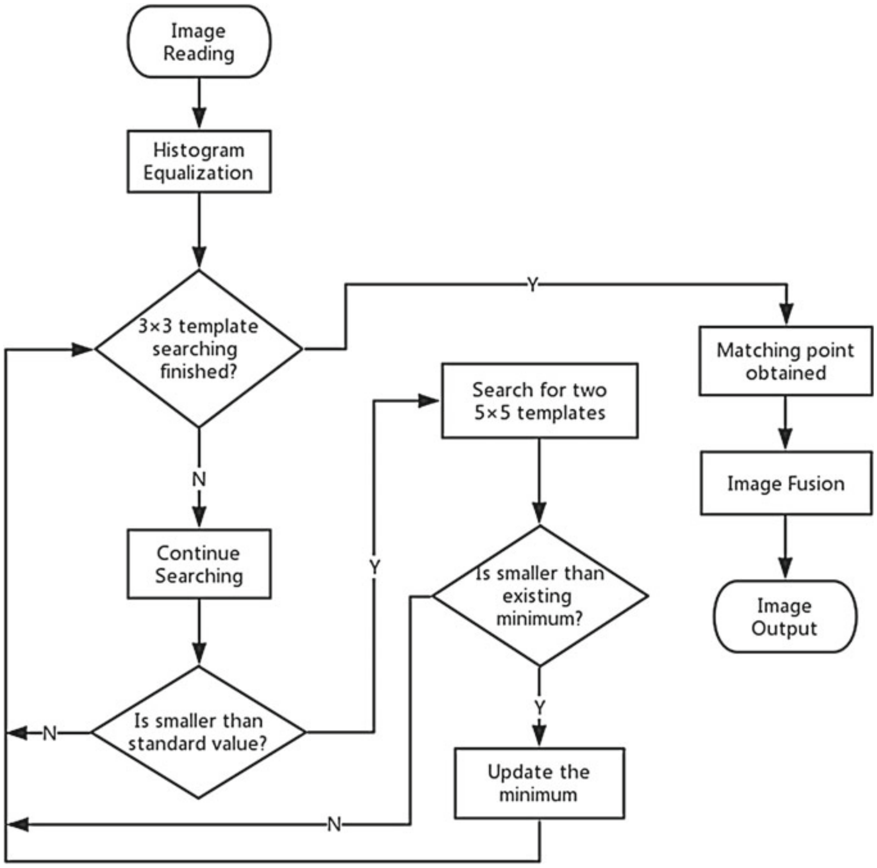


Image Reading
Histogram Equalization
3×3 template searching finished?
Y
N
Matching point obtained
Search for two 5×5 templates
Image Fusion
Continue Searching
Y
Is smaller than existing minimum?
Image Output
Y
N
Is smaller than standard value?
Update the minimum
N

Fig. 8.5 The flowchart of image stitching using line and surface feature

(2) Feature area searching. We take P_1 as the referenced image and P_2 as the image to be stitched. The size of P_1 is $(M_1 \times N_1)$ and Picture 2 is $(M_2 \times N_2)$. This algorithm will select 3 tiny feature templates in P_1 for matching. First of all, limit the area used to select the template is from line 1 to line M_1 and column $N_{1/2}$ to column N_1 in P_1. We will first select a 3×3 small template named H_1 in this area. Then, according to the image features, we select the other two 5×5 templates named H_2 and H_3 respectively in the area. As shown in Fig. 8.6, a feature template group consisting of 3 tiny templates is made up. (Note: We suppose that H_2 and H_1 are at a same level in horizonal direction and so as H_3 and H_1 are in vertical, and the distance from Hl is Ll and L2, respectively.)
We adopt the method which calculates the variance values of pixels in templates when selecting the feature template. We select the template with maximum sum of variance as the standard template because the detail features in images are determined by edge features or inflection points of the grayscale value. Where the maximum sum of variance is equals to the position of edges or inflection points that fluctuate the most in the curves of gray levels. We can measure how many detail features there are in a template with the sum of pixel variance values. The more details and texture information P_1 contains, the easier to find similar areas in P_2. Here are the equations for selecting the feature template.

$$S(x, y) = \sum_{i=1}^{M} \sum_{j=1}^{M} |I(i, j) - \omega|^2 \tag{8.4}$$

$$S_{result}(x, y) = MAX(S(x, y)) \tag{8.5}$$

In Eq. (8.4), $I(i, j)$ represents the pixel value and ω stands for the average gray value of the template. When we set M as 3, we can obtain the best 3×3 feature

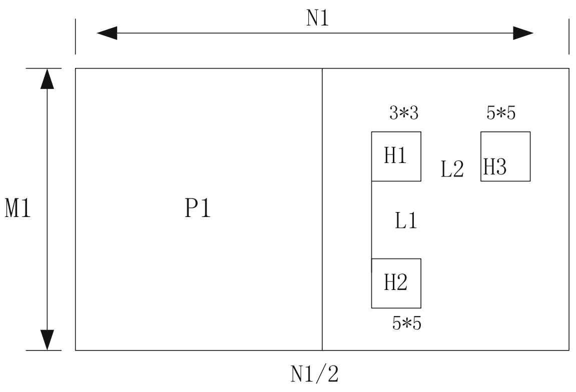


Fig. 8.6 Extracting the group of feature template

template through Eqs. 8.4 and 8.5. Repeat the calculation twice to get the other two 5 × 5 templates. In this way, 3 feature templates with best details are extracted. Record the distance information around them to compose the feature template group.

After extracting appropriate feature template groups, searching from top to bottom and left to right with 3 × 3 template Fl in P_2, and calculate the pixel difference between F_1 and H_1 one by one. MSE function is used to define the difference function. Here's the definition:

$$S_{3\times 3}(x, y) = \frac{1}{9}\sum_{i=1}^{3}\sum_{j=1}^{3}[F_1(i,j) - H_1(i,j)]^2 \tag{8.6}$$

where $F_1(i,j)$ stands for the grey value of corresponding pixels in F_1 and so as $H_1(i,j)$ for H_1.

The experimental results indicate that when the difference of the light intensity in images is small, setting the standard value of the difference function to 30 is an appropriate value if the selected feature template is 3 × 3. When the selected standard value is greater than 30, the number of templates to meet the conditions is increasing rapidly, which will lead to a longer time spent in computation. If the chosen standard value is less than 30, it is difficult to find the templates meeting the criteria when there is high interference between two images and cause the failure of the algorithm.

When the difference calculated by the Eq. 8.6 is greater than the standard value 30, the difference between the two templates is considered to be very small and it is almost impossible to be a matched area. Hence, we should discard the template and go on with next calculation. When the difference is less than 30, it is considered that the template is very likely to be a matched template. However, the 3 × 3 template is too small to locate accurately. So according to the recorded distance information between the templates 3 × 3 and 5 × 5 in a template group, we can find the corresponding two 5 × 5 templates in the location of the same distance information around the 3 × 3 interest template in P_2. Then calculate the difference between the 5 × 5 templates corresponding to the feature template group. The function for the sum of differences is defined as below.

$$S_{5\times 5}(x, y) = \frac{1}{25}\sum_{i=1}^{5}\sum_{j=1}^{5}[F_2(i,j) - H_2(i,j)]^2 + \frac{1}{25}\sum_{i=1}^{5}\sum_{j=1}^{5}[F_3(i,j) - H_3(i,j)]^2 \tag{8.7}$$

Calculate all F_1 whose difference is less than 30 through Eq. 8.7 and save every result. Besides, saving the transverse and ordinate values of the most left upper corner pixel points of the template Fl at the same time. Finally, using Eq. 8.8

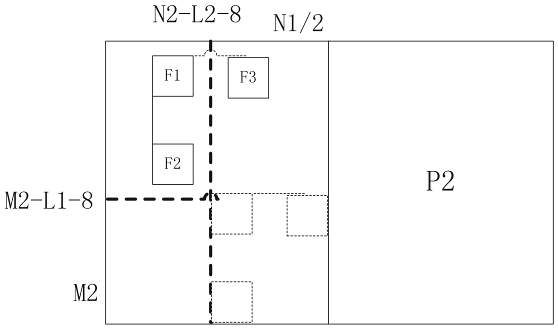


Fig. 8.7 Traversal matching of the feature template

to get the minimum difference sum, the transverse and ordinate values of the upper left corner of the Fl template corresponding to the minimum difference sum are the coordinates of the matching points obtained.

$$S_{find}(x, y) = MIN(S_{5\times 5}(x, y)) \tag{8.8}$$

This algorithm actually obtains the best matching template by filtering templates twice. First measure the relevance of the small 3×3 template and save each template with high correlation. Then calculate the relevant of two 5×5 templates that correspond to saved templates and collect the 5×5 template with highest relevance. This means to filter the templates obtained in the first step again for the best matching template (Fig. 8.7).

(3) Image stitching and image fusion. After finding the matching point, simple superposition will cause obvious borders in the picture which is undesirable. A smooth transition for image stitching is required to eliminate such undue influences. Gradated in-and-out algorithm can gain seamless images, but during the image fusion period, the overlapping areas of two images are superimposed by linear weighting and this certainly makes the overlapping areas more blurred than the original image. Hence, we use Gaussian fusion instead. By making the change of gradient factor from 0 to 1 follows the distribution characteristic of Gauss curve approximately, and achieves quick transition between two images. The overlapping area of the stitching result is clearer than that of gradated in-and-out approach.

Matlab programme of the algorithm mentioned above is shown as follows:

PROGRAMME 8.2: Image stitching using line and surface feature

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
A=imread('1.bmp');
subplot(1,2,1),imshow(A)
title(' Source Image A')
B=imread('2.bmp');
subplot(1,2,2),imshow(B)
title('Source Image B')
%%%%%%%%%%%%%%%%%%%%%%%%%%
[high,wid]=size(A);
A1=double(A);
B1=double(B);
sub_A=A1(high/2-39:high/2,3*wid/4:3*wid/4+39);
% sub_A=A1(high/2-39:high/2,end-39:end);
sub_B1=B1(11:50,11:50);
mod1=sub_A-sub_B1;
mat1=sum(sum(mod1.*mod1));
mat_best=mat1;
for x1=1:40:wid-40
    for y1=1:40:high-40
        sub_B=B1(y1:y1+39,x1:x1+39);
        mod=sub_A-sub_B;
        mat=sum(sum(mod.*mod));
        if   mat<=mat_best
            mat_best=mat;
            xx=x1;
            yy=y1;
        end
    end
end
x=xx;       % custom settings
y=yy;
for x2=xx-40:xx
    for y2=yy-20:yy+80
        sub_B2=B1(y2:y2+39,x2:x2+39);
        mod2=sub_A-sub_B2;
        mat2=sum(sum(mod2.*mod2));
        if mat2<=mat_best
           mat_best=mat2;
           x=x2;
           y=y2;
       end
```

```
        end
    end
% figure
% colormap(gray);
% subplot(2,1,1);imagesc(sub_A)
% subplot(2,1,2);imagesc(sub_B2)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x=55;y=1;
%************************
  if y==0
        AA=A(1:high-1,1:3*wid/4);
        BB=B(1:high-1,1:wid-1);
else if y>=high/2
                AA=A(1:high-y,1:3*wid/4);
                BB=B(y:high-1,x:wid-1);
                else
                AA=A(y:high-1,1:3*wid/4);
                BB=B(1:high-y,x:wid-20);
                end
end
C=[AA BB];
%imwrite(C,'Directly stitched image 34.bmp');
figure,imshow(C)
title(' Directly stitched image')
if y==0
        A2=A(1:high-1,:);
        B2=B(1:high,:);
else if y>=50
                A2=A(1:high-y,:);
                B2=B(y:high-1,:);
        else
                A2=A(y:high-1,:);
                B2=B(1:high-y,:);
        end
end
x=140;%************************
[high2,wid2]=size(A2);
a1=A2(1:high2,wid-x+1:wid);
b1=B2(1:high2,1:x);
a=double(a1);
```

```
b=double(b1);
d1O=linspace(1,0,x);
d=1:high2;
d1=d1O';
[X1,y1]=meshgrid(d1,d);
im1=a.*X1;%************************
d20=linspace(0,1,x);
d2=d20';
[X2,y2]=meshgrid(d2,d);
im2=b.*X2;
im11=uint8(im1);
im22=uint8(im2);
im3=imadd(im11,im22);
figure,imshow(im3)
title('Fusion zone with gradated')
a_b=imadd(im11,im22);
aa=A2(1:high2,1:wid-x);
bb=B2(1:high2,x:wid);
%D2=[aa a_b bb];
D2=[aa(:,1:wid2-x) a_b bb];%************************
imwrite(D2,'6.bmp');
figure,imshow(D2)
title('Image with gradated');
```

Figure 8.8 shows the result.

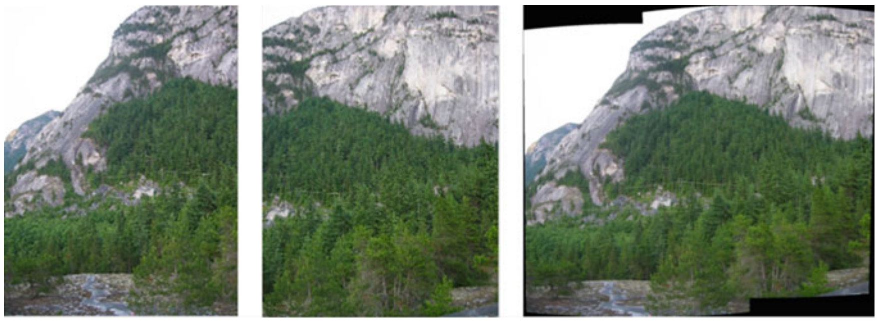

(a) Source image 1 **(b) Source image 2** **(c) Final output**

Fig. 8.8 Result of stitching

8.2.3 Image Stitching Based on FFT

Panorama refers to the formation of a full view, high resolution 360° image through image processing. It is an integrated reproduction of the view which observers looking around, and it can show better integral information of the surroundings.

Image stitching based on FFT first converts the image to the frequency domain and calculates the rotation amounts and offsets according to its phase cross power spectrum. Then reset the coordinate of the image and apply the movement. At last, the images are stitched together. When stitching a 360° panorama, conversions of the focal length and projections are needed before calculating with phases. Figure 8.9 presents the flowchart of image stitching based on FFT.

The approach applied for stitching cylindrical panoramic image can be divide into 3 parts:

(1) Construct a function with the phase correlation of frequency domain. This function will carry out the 2-D Fourier transformation on two input images and return the offset values between two adjacent images.
(2) Calculate the focal length values of a set of 360° live-action photos and apply the cylindrical projection to the image sequence.
(3) Call the function in part 1 one by one to stitch the images after the projection, and the lighting is processed to generate the cylindrical panoramic image. is generated.

Focal length f is a significant parameter when using the cylindrical projection formula for projection transformation. We set the translations between every two

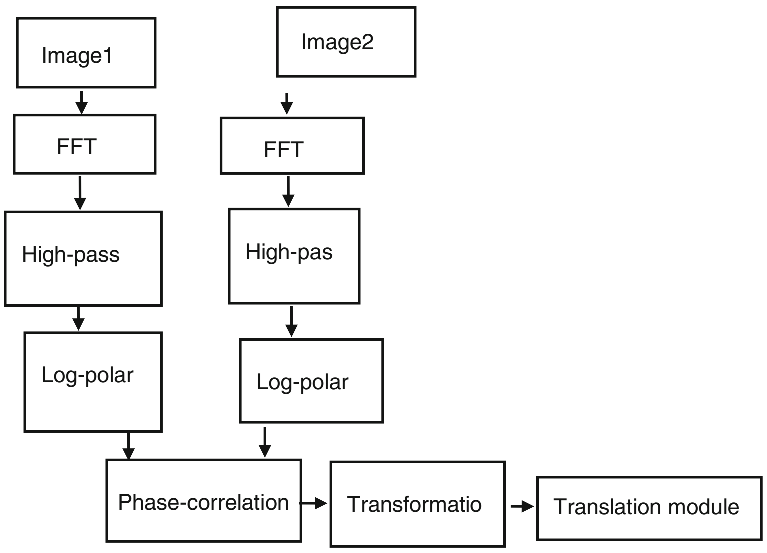


Fig. 8.9 Flowchart of image stitching based on FFT

adjacent images in the image sequence before projection as $C_1, C_2, \ldots, C_n$ respectively, where C_k represents the horizontal translation between the image k and the image $k + 1$. The initial value of focal length named f_0 can be calculated through Formula 8.9:

$$f_0 = \sum_{k=1}^{k=n} \frac{C_k}{2\pi} \tag{8.9}$$

The source code of image stitching based on FFT is shown as PROGRAMME 8.3.

PROGRAMME 8.3: Image stitching based on FFT

```
Function main()
clc;clear;
image1=(imread('D1.bmp'));[h,w,r]=size(image1);
image2=(imread('D2.bmp'));T0(:,:,:,1)=image1;T0(:,:,:,2)=image2;
subplot(121),imshow(image1);subplot(122),imshow(image2);
image11=multi_resolution(image1,2);
image22=multi_resolution(image2,2);
[r1,c1,d1]=size(image11);
[r2,c2,d2]=size(image22);
% calculate phase correlation offsets
tic
fprintf(' calculate phase correlation offsets...');
[i,j]=poc_2pow(image11,image22);
coor_shift(1,1)=i;
coor_shift(1,2)=j;
coor_shift(2,1)=0;coor_shift(2,2)=0;
coor_shift=coor_shift*2^2;%%% convert to offsets of original image
toc
%transform into cylindrical coordinate system
tic
f=sqrt(h^2+w^2);
[T1,coor_shift02]=coortransf(T0,f,coor_shift);
toc
%fuse overlapping areas
tic
fprintf('Fusing overlapping areas and stitching the image...');
panorama1=mosaic(T1(:,:,:,1),T1(:,:,:,2),coor_shift02(1,1),coor_shift02(1,2));
toc
```

```
%image reconstruction
tic
fprintf('Saving and displaying the result...');
imwrite(panorama1,'pic2.jpg','jpg');
imshow(panorama1,[]);
Toc

function T=multi_resolution(Xb,n)
% multiresolution decomposition
[r1,c1,d1,N]=size(Xb);
for i=1:N
    Xb(:,:,1,N)=filter2(fspecial('gaussian'),Xb(:,:,1,N));%%Default parameters of Gaussian filter [3 3]，
  sigma=0.5
    Xb(:,:,2,N)=filter2(fspecial('gaussian'),Xb(:,:,2,N));
    Xb(:,:,3,N)=filter2(fspecial('gaussian'),Xb(:,:,3,N));
    end
    step=2^n;
    for i=1:step:r1
        for j=1:step:c1
            T((i+step-1)/step,(j+step-1)/step,:,:)=Xb(i,j,:,:);
        end
    end

    function [dis,dm]=poc_2pow(imageL,imageR);
    %% phase correlation algorithm
    % imageL=image1;
    % imageR=image2;
    [H1,W1,d1]=size(imageL);
    [H2,W2,d2]=size(imageR);
    if d1==3 imageL=rgb2gray(imageL);end%%%grayscale
    if d2==3 imageR=rgb2gray(imageR);end
    % extract the binary outlines of the 2-exponential histogram
    [imageL,t1]=edge(imageL,'canny',[],1.2);%%%%sigma=1.2(Default 1)
    [imageR,t2]=edge(imageR,'canny',[],1.2);%%%% auto select the threshold value
    Xb=imageL;Yb=imageR;
    %2-exponential histogram
    for i=5:11
        index2=2^i;
        if index2<=H1 && index2<=W1 h1=index2;end
        if index2<=H2 && index2<=W2 h2=index2;end
```

```
end
% minhw1=min(h1,w1);
% minhw2=min(h2,w2);
minhw1=h1;minhw2=h2;
offset1=round((H1-minhw1)/2);
offset2=round((H2-minhw2)/2);
imageL=imageL(offset1:offset1+minhw1-1,W1-minhw1+1:W1);%%%choose right center in the left
    image
imageR=imageR(offset2:offset2+minhw2-1,1:minhw2);%%%% choose left center in the right image
% phase correlation algorithm for measuring offsets
A=fft2(im2double(imageL));%FFT in frequency domain
B=fft2(im2double(imageR));
AB=conj(A).*(B);%%% conjugated convolution, equals to phase transformation
modAB=abs(AB);
%peak value，（I，J）save peak coordinates which are offsets
COR=ifft2(AB);%%%unnormalize, reverse transformation for coeerlation
emin=100000;
% for i=1:10
     [maxC,sorti]=max(COR);
     [C,J]=max(maxC);
     I=sorti(J);
     if I<20 dis=I;
     elseif H2-I<20 dis=(I-H2);
     else dis=0;
     end
     dm=J;

function [T1,coor_shift02]=coortransf(T0,f,coor_shift)
%%transformation from image coordinate to cylindrical coordinate
%transform input image sequence T0 with focal length f
coor_shift02=coor_shift;%%%% the first dimension (row values) stay unchanged and the second
    dimension (column values) update after mapping
[H,W,r,N]=size(T0);
w2f=W/2/f;
h2=H/2;
constant2=f*atan(W/(2*f));
constant1=h2;
```

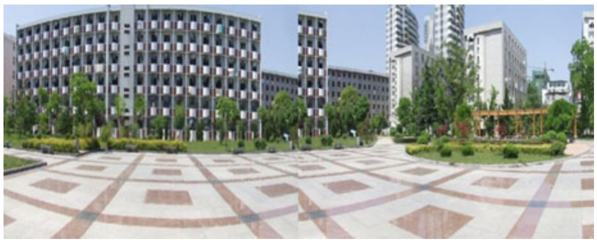
```
for y=1:W          %%%%%%     columns
       angle=atan(y/f-w2f);%%%%atan((y-W/2)/f);
         y1=uint16(f*angle+constant2);
         if   y1==0   y1=1;   end
       for x=1:H   %%%%%%%%%%%%     rows
         x1=uint16((x-h2)*cos(angle)+constant1);
         if   x1==0   x1=1;end
         if r==3      %%%%%%%%%%%%%color image
                for n=1:N   %%%%%%%%%
                       if (y==coor_shift(n,2)) coor_shift02(n,2)=y1; end%%%corresponding offsets
                       T1(x1,y1,:,n)=T0(x,y,:,n);%%%% mapping of points
                end
         elseif r==1
         end
       end
end
[h,w,a,N]=size(T1);
for i=1:60
       for j=1:w
              if (T1(i,j,:,:)==0)
                     T1(i,j,:;)=255;
              end
              if (T1(h-i,j,:,:)==0)
                     T1(i,j,:,:)=255;
              end
       end
end

function D=mosaic(image1,image2,i,j)
[ra,ca,a]=size(image1);
[rb,cb,b]=size(image2);
Xa=image1;Ya=image2;
% dis=i;%%% top and bottom offsets
dis=i;
EXa=zeros(abs(dis),ca,3)+255;
EXb=zeros(abs(dis),cb,3)+255;
if dis>1
   Xa=[EXa;Xa];
   Ya=[Ya;EXb];
elseif dis<-1
       Xa=[Xa;EXa];
       Ya=[EXb;Ya];
```

```
end
dm=j;%%% stitching crack width ,limited no more than 50 pel
     A=Xa(:,1:(ca-dm-1),:);
     B1=Xa(:,(ca-dm):ca,:);
     B2=Ya(:,1:dm,:);
     B=imagefusion02(B1,B2);%%partial overlapping(fusion)
     C=Ya(:,(dm+1):cb,:);%% cut out the rest part of the second image
     D=[A,B,C];%%merge and complete stitching
     %%%% eliminate accumulative errors
     [r,c]=size(D);
     if dis>1
            D=D(1:(r-dis),:,:);
     elseif dis<-1
            D=D((abs(dis)+1):r,:,:);
     end
     function [dis,dm]=phase_correlation(image1,image2);
     % phase correlation algorithm
     %%% C=phase_correlation(imagea,imageb)      input two images
     [H1,W1,d1]=size(image1);
     [H2,W2,d2]=size(image2);
     if d1==3 image1=rgb2gray(image1);end%%%grayscale
     if d2==3 image2=rgb2gray(image2);end
     Xb=image1;Yb=image2;
     [image1,t1]=edge(image1,'canny');
       [image2,t2]=edge(image2,'canny');
     A=fft2(im2double(image1));%FFT in frequency domain
     B=fft2(im2double(image2));
     AB=conj(A).*(B);%%% conjugated convolution, equals to phase transformation
     COR=ifft2(AB);%%% unnormalize, reverse transformation for coeerlation
     [C,i]=max(COR);[C,J]=max(C);I=i(J);%%% peak value，（I，J）save peak coordinates which are offsets
     if I<15 dis=I;
     elseif H2-I<15 dis=I-H2;
     else dis=0;
     end
     dm=J;
     function C=imagefusion02(A,B)
     %%%image fusion
     [M,N,D]=size(A);
```

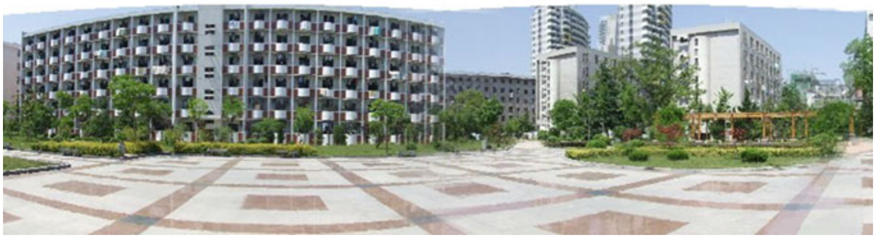
```
if D==3
for i=1:(N-1)
C(:,i,:)=round((double(A(:,i,:))*(N-i)+double(B(:,i,:))*i)/N);
end
elseif D==1
for i=1:(N-1)
C(:,i)=round((double(A(:,i))*(N-i)+double(B(:,i))*i)/N);
end
end
% figure,imshow(C/max(max(max(C))));
```

Figure 8.10 is the result of image stitching.

Image stitching based on FFT demands images to have the same size and more than 30% overlapping areas. Moreover, it is only applicable to image registration with translation, rotation and scaling. Non-linear distortions as tangential transformation is not applicable. This algorithm only utilizes the phase information in cross power spectrum for image registration hence it is insensitive to the changes of brightness among images.



(a) The source image A **(b) The source image B**



(c) The output panorama

Fig. 8.10 The sketch map for image stitching

8.3 Images Stitching Based on Feature Points

Instead of using pixel values of the images, image stitching [2] based on feature points [3] calculates features such as texture, edges, objects and etc., from pixels, and then uses features as standards and search matches for the corresponding feature areas of overlapping images. This kind of approaches are more robust. There are two processes for image stitching based on feature points: feature extraction and feature registration. First, extract points, lines and regions where gray scale changes obviously to form a feature set. Second, try to choose the paired features using feature matching algorithms between two feature sets. A series of image segment approaches have been applied to feature extraction and edge detection, such as canny descriptor, Laplace-gauss descriptor and region seeds growing (RSD). The extracted spatial features include closed edges, open edges, crossed lines and other features. Feature matching algorithms include cross correlation, distance transformation, dynamic programming, structure matching and chain code correlation algorithms.

8.3.1 SIFT Feature Points Detection

The process of image stitching based on SIFT feature points include: image acquisition, feature extraction and matching, image registration (calculating H) and finally image stitching.

(1) Image acquisition. Image acquisition is the precondition for image stitching. Different image acquisition methods can obtain different input image sequences and produce different image stitching effects. Currently, there are three different methods to obtain image sequences: (1) fix the camera to the tripod and rotate it to get the image data; (2) fix the camera on a movable platform, and the image data is obtained by parallel moving it; (3) Hand-held the camera for capturing image data by a fixed-point rotating or moving in the direction perpendicular to the camera's optical axis. This process utilizes given images.

(2) Feature extraction and matching. Extract SIFT feature points from the input image sequences. The algorithm calculates and extracts the feature points simultaneously in the spatial domain and the scale domain. Therefore, the obtained feature points have the scale invariance, which can correctly extract the feature points that exist in the image sequences with large scale and angle change. Euclidean distance is used to calculate the distance between two SIFT feature point descriptors.

(3) Image registration with calculation of H. Image registration based on feature points means that the transformation matrix between image sequences is constructed by matching points to complete the stitching of panoramic images. To improve the precision of image registration, RANSAC algorithm [4] is used to calculate and refine the transformation matrix. H algorithm to automatically calculate the transformation matrix: calculate feature points in each image;

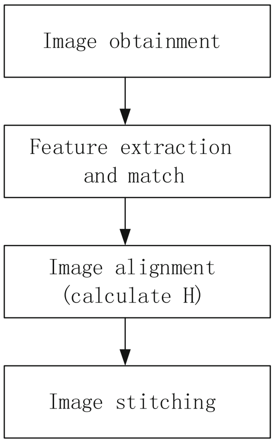


Fig. 8.11 The process of image stitching based on SIFT

match feature points; calculate initial value of the matrix; use iteration to refine H transformation matrix; boot matching; repeat iterations until the number of corresponding points is stable.

(4) Image fusion. According to the transformation matrix H of two images, the corresponding images can be transformed to determine the overlapping region of the images, and register the images to be merged into a new blank image to form a mosaic diagram. A quick and simple weighted smoothing algorithm is used to deal with the stitching seam problem.

The process of image stitching algorithm based on SIFT feature points is shown in Fig. 8.11:

The MATLAB source programme (main code) of image stitching algorithm based on SIFT feature points is shown in PROGRAMME 8.4. SIFT feature detection programme is shown in Chap. 4 PROGRAMME 4.9.

PROGRAMME 8.4: Image stitching based on SIFT feature points

```
function [ imgout ] = imMosaic( img1,img2,adjColor )
% use SIFT to find corresponding points
[matchLoc1 matchLoc2] = siftMatch(img1, img2);
% use RANSAC to find homography matrix
[H corrPtIdx] = findHomography(matchLoc2',matchLoc1');
H   %#ok
tform = maketform('projective',H');
```

```
img21 = imtransform(img2,tform); % reproject img2
%fis(img1)
%fis(img21)
% adjust color or grayscale linearly, using corresponding infomation
[M1 N1 dim] = size(img1);
[M2 N2 ~] = size(img2);
if exist('adjColor','var') && adjColor == 1
    radius = 2;
    x1ctrl = matchLoc1(corrPtIdx,1);
    y1ctrl = matchLoc1(corrPtIdx,2);
    x2ctrl = matchLoc2(corrPtIdx,1);
    y2ctrl = matchLoc2(corrPtIdx,2);
    ctrlLen = length(corrPtIdx);
    s1 = zeros(1,ctrlLen);
    s2 = zeros(1,ctrlLen);
    for color = 1:dim
      for p = 1:ctrlLen
            left = round(max(1,x1ctrl(p)-radius));
            right = round(min(N1,left+radius+1));
            up = round(max(1,y1ctrl(p)-radius));
            down = round(min(M1,up+radius+1));
            s1(p) = sum(sum(img1(up:down,left:right,color))); % calculate chroma of the around points
      end
      for p = 1:ctrlLen
            left = round(max(1,x2ctrl(p)-radius));
            right = round(min(N2,left+radius+1));
            up = round(max(1,y2ctrl(p)-radius));
            down = round(min(M2,up+radius+1));
            s2(p) = sum(sum(img2(up:down,left:right,color)));
      end
      sc = (radius*2+1)^2*ctrlLen;
      adjcoef = polyfit(s1/sc,s2/sc,1);
      img1(:,:,color) = img1(:,:,color)*adjcoef(1)+adjcoef(2);
    end
end
% do the mosaic
pt = zeros(3,4);
pt(:,1) = H*[1;1;1];
pt(:,2) = H*[N2;1;1];
pt(:,3) = H*[N2;M2;1];
pt(:,4) = H*[1;M2;1];
x2 = pt(1,:)./pt(3,:);
y2 = pt(2,:)./pt(3,:);
up = round(min(y2));
Yoffset = 0;
if up <= 0
    Yoffset = -up+1;
    up = 1;
end
left = round(min(x2));
Xoffset = 0;
if left<=0
    Xoffset = -left+1;
    left = 1;
end
[M3 N3 ~] = size(img21);
imgout(up:up+M3-1,left:left+N3-1,:) = img21;
    % img1 is above img21
imgout(Yoffset+1:Yoffset+M1,Xoffset+1:Xoffset+N1,:) = img1;
end
% [matchLoc1 matchLoc2] = siftMatch(img1, img2)
% This function reads two images, finds their SIFT features, and
%     displays lines connecting the matched keypoints.   A match is accepted
%     only if its distance is less than distRatio times the distance to the
%     second closest match.
% It returns the matched points of both images, matchLoc1 = [x1,y1;x2,y2;...]
% Example: match('scene.pgm','book.pgm');
function [matchLoc1 matchLoc2] = siftMatch(img1, img2)
% load matchdata
% load img1data
% load img2data
%{,
% Find SIFT keypoints for each image
[des1, loc1] = sift(img1);
[des2, loc2] = sift(img2);
% save img1data des1 loc1
% save img2data des2 loc2
% For efficiency in Matlab, it is cheaper to compute dot products between
%   unit vectors rather than Euclidean distances.   Note that the ratio of
%   angles (acos of dot products of unit vectors) is a close approximation
%    to the ratio of Euclidean distances for small angles.
% distRatio: Only keep matches in which the ratio of vector angles from the
%      nearest to second nearest neighbor is less than distRatio.
distRatio = 0.6;
% For each descriptor in the first image, select its match to second image.
des2t = des2';                                    % Precompute matrix transpose
matchTable = zeros(1,size(des1,1));
for i = 1 : size(des1,1)
    dotprods = des1(i,:) * des2t;                 % Computes vector of dot products
    [vals,indx] = sort(acos(dotprods));    % Take inverse cosine and sort results
% Check if nearest neighbor has angle less than distRatio times 2nd.
    if (vals(1) < distRatio * vals(2))
        matchTable(i) = indx(1);
    else
        matchTable(i) = 0;
    end
end
% save matchdata matchTable
%}
% Create a new image showing the two images side by side.
img3 = appendimages(img1,img2);
% Show a figure with lines joining the accepted matches.
figure('Position', [100 100 size(img3,2) size(img3,1)]);
colormap('gray');
imagesc(img3);
hold on;
cols1 = size(img1,2);
for i = 1: size(des1,1)
   if (matchTable(i) > 0)
      line([loc1(i,2) loc2(matchTable(i),2)+cols1], ...
              [loc1(i,1) loc2(matchTable(i),1)], 'Color', 'c');
   end
end
hold off;
num = sum(matchTable > 0);
fprintf('Found %d matches.\n', num);
idx1 = find(matchTable);
idx2 = matchTable(idx1);
x1 = loc1(idx1,2);
x2 = loc2(idx2,2);
```

```
y1 = loc1(idx1,1);
y2 = loc2(idx2,1);
matchLoc1 = [x1,y1];
matchLoc2 = [x2,y2];
end
% [descriptors, locs] = sift(img)
function [f inlierIdx] = ransac1( x,y,ransacCoef,funcFindF,funcDist )
%[f inlierIdx] = ransac1( x,y,ransacCoef,funcFindF,funcDist )
% Use RANdom SAmple Consensus to find a fit from X to Y.
% X is M*n matrix including n points with dim M, Y is N*n;
% The fit, f, and the indices of inliers, are returned.
%
% RANSACCOEF is a struct with following fields:
% minPtNum,iterNum,thDist,thInlrRatio
% MINPTNUM is the minimum number of points with whom can we
% find a fit. For line fitting, it's 2. For homography, it's 4.
% ITERNUM is the number of iteration, THDIST is the inlier
% distance threshold and ROUND(THINLRRATIO*n) is the inlier number threshold.
%
% FUNCFINDF is a func handle, f1 = funcFindF(x1,y1)
% x1 is M*n1 and y1 is N*n1, n1 >= ransacCoef.minPtNum
% f1 can be of any type.
% FUNCDIST is a func handle, d = funcDist(f,x1,y1)
% It uses f returned by FUNCFINDF, and return the distance
% between f and the points, d is 1*n1.
% For line fitting, it should calculate the dist between the line and the
% points [x1;y1]; for homography, it should project x1 to y2 then
% calculate the dist between y1 and y2.
minPtNum = ransacCoef.minPtNum;
iterNum = ransacCoef.iterNum;
thInlrRatio = ransacCoef.thInlrRatio;
thDist = ransacCoef.thDist;
ptNum = size(x,2);
thInlr = round(thInlrRatio*ptNum);
inlrNum = zeros(1,iterNum);
fLib = cell(1,iterNum);
for p = 1:iterNum
    % 1. fit using   random points
    sampleIdx = randIndex(ptNum,minPtNum);
    f1 = funcFindF(x(:,sampleIdx),y(:,sampleIdx));
    % 2. count the inliers, if more than thInlr, refit; else iterate
    dist = funcDist(f1,x,y);
    inlier1 = find(dist < thDist);
    inlrNum(p) = length(inlier1);
    if length(inlier1) < thInlr, continue; end
    fLib{p} = funcFindF(x(:,inlier1),y(:,inlier1));
end
% 3. choose the coef with the most inliers
[~,idx] = max(inlrNum);
f = fLib{idx};
dist = funcDist(f,x,y);
inlierIdx = find(dist < thDist);
end
clear
close all
f = 'a';
ext = 'jpg';
img1 = imread([f '1.' ext]);
img2 = imread([f '2.' ext]);
% img3 = imread('b3.jpg');
img0 = imMosaic(img2,img1,1);
% img0 = imMosaic(img1,img0,1);
fis(img0)
imwrite(img0,['mosaic_' f '.' ext],ext)
```

The result of image stitching is shown in Fig. 8.12:

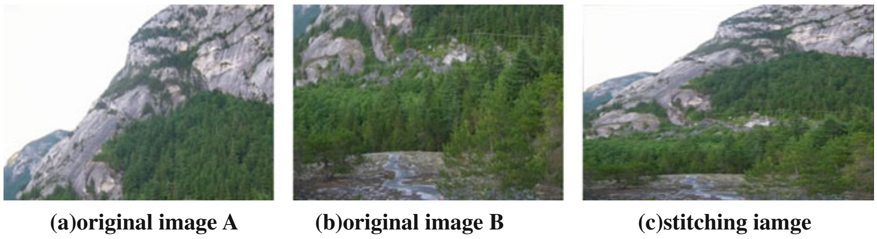


Fig. 8.12 Image stitching result

8.3.2 Image Stitching Based on Harris Feature Points

The process of image stitching based on Harris feature points is as follows.

(1) Detect Harris feature points of images;
(2) Connect the feature points between two images and complete image matching;
(3) Filter all matching points and obtain points which are needed for image stitching;
(4) Calculate the distance between feature points of two images and smooth the overlapping parts of the images.

The core code is shown in PROGRAMME 8.5.

PROGRAMME 8.5: Image stitching based on Harris feature points

```
clc
clear all
% read image
pic1=imread('lena1.jpg');
pic2=imread('lena2.jpg');
% Harris feature points detection
points1=myHarris(pic1);
points2=myHarris(pic2);
% draw Harris feature points
figure(1)
drawHarrisCorner(pic1,points1,pic2,points2);
% describe Harris feature
des1=myHarrisCornerDescription(pic1,points1);
des2=myHarrisCornerDescription(pic2,points2);
% coarse match
matchs=myMatch(des1,des2);
% obtain position of match points
matchedPoints1=points1(matchs(:,1),:);
matchedPoints2=points2(matchs(:,2),:);
% line coarse match points
figure(2)
drawLinedCorner(pic1,matchedPoints1,pic2,matchedPoints2);
% Harris feature points fine matching
[newLoc1,newLoc2]=pointsSelect(matchedPoints1,matchedPoints2);
% line fine match points
figure(3)
drawLinedCorner(pic1,newLoc1,pic2,newLoc2);
% stitch images
```

```
im=picMatched(pic1,newLoc1,pic2,newLoc2);
% show the stitching image
figure(4)
imshow(im);
set(gcf,'Color','w');
function points=myHarris(pic)
% function:Harris feature points detection
% input: RGB image or gray scale image
% output:the row and col N×2 matrix of the feature point
if length(size(pic))==3
pic=rgb2gray(pic);
end
pic=double(pic);
hx=[-1 0 1];
Ix=filter2(hx,pic);
hy=[-1;0;1];
Iy=filter2(hy,pic);
Ix2=Ix.*Ix;
Iy2=Iy.*Iy;
Ixy=Ix.*Iy;
h=fspecial('gaussian',[7 7],2);
Ix2=filter2(h,Ix2);
Iy2=filter2(h,Iy2);
Ixy=filter2(h,Ixy);
[heigth,width]=size(pic);
alpha=0.06;
R=zeros(heigth,width);
for i=1:heigth
for j=1:width
M=[Ix2(i,j) Ixy(i,j);Ixy(i,j) Iy2(i,j)];
R(i,j)=det(M)-alpha*(trace(M)^2);
end
end
Rmax=max(max(R));
pMap=zeros(heigth,width);
for i=2:heigth-1
for j=2:width-1
if R(i,j)>0.01*Rmax
tm=R(i-1:i+1,j-1:j+1);
tm(2,2)=0;
```

```
if R(i,j)>tm
pMap(i,j)=1;
end
end
end
end
[row,col]=find(pMap==1);
points=[row,col];
function drawHarrisCorner(pic1,points1,pic2,points2)
% function:draw Harris feature points' match connection
% input:
% pic1、 pic2:the images need stitching
% points1、 points2: Harris feature points position
X1=points1(:,2);
Y1=points1(:,1);
X2=points2(:,2);
Y2=points2(:,1);
dif=size(pic1,2);
imshowpair(pic1,pic2,'montage');
hold on
plot(X1,Y1,'b*');
plot(X2+dif,Y2,'b*');
set(gcf,'Color','w');
function des=myHarrisCornerDescription(pic,points)
% Function: describe Harris feature points
% Input:
% pic:source image
% points:feature points' position
% Output:
% des: 8×N matrix describing Harris feature points
if length(size(pic))==3
pic=rgb2gray(pic);
end
len=length(points);
des=zeros(8,len);
for k=1:len
p=points(k,:);
pc=pic(p(1),p(2));
des(1,k)=pic(p(1)-1,p(2)-1)-pc;
des(2,k)=pic(p(1),p(2)-1)-pc;
des(3,k)=pic(p(1)+1,p(2)-1)-pc;
des(4,k)=pic(p(1)+1,p(2))-pc;
des(5,k)=pic(p(1)+1,p(2)+1)-pc;
des(6,k)=pic(p(1),p(2)+1)-pc;
des(7,k)=pic(p(1)-1,p(2)+1)-pc;
des(8,k)=pic(p(1)-1,p(2))-pc;
des(:,k)=des(:,k)/sum(des(:,k));
end
function matchs=myMatch(des1,des2)
% Function:feature point bidirectional match
% input:
% des1、 des2:feature point description matrix
% Output:
% matchs:correspondence relation of match points
len1=length(des1);
len2=length(des2);
match1=zeros(len1,2);
cor1=zeros(1,len2);
for i=1:len1
d1=des1(:,i);
for j=1:len2
d2=des2(:,j);
cor1(j)=(d1'*d2)/sqrt((d1'*d1)*(d2'*d2));
end
[~,indx]=max(cor1);
match1(i,:)=[i,indx];
end
match2=zeros(len2,2);
cor2=zeros(1,len1);
for i=1:len2
d2=des2(:,i);
for j=1:len1
d1=des1(:,j);
cor2(j)=(d1'*d2)/sqrt((d1'*d1)*(d2'*d2));
end
[~,indx]=max(cor2);
match2(i,:)=[indx,i];
end
```

```
matchs=[];
for i=1:length(match1)
for j=1:length(match2)
if match1(i,:)==match2(j,:)
matchs=[matchs;match1(i,:)];
end
end
end
function drawLinedCorner(pic1,loc1,pic2,loc2)
% Function:draw connections between match points
% Input:
% pic1、 pic2:image to need stitching
% loc1、 loc2:position of the paired points
X1=loc1(:,2);
Y1=loc1(:,1);
X2=loc2(:,2);
Y2=loc2(:,1);
dif=size(pic1,2);
imshowpair(pic1,pic2,'montage');
hold on
for k=1:length(X1)
plot(X1(k),Y1(k),'b*');
plot(X2(k)+dif,Y2(k),'b*');
line([X1(k),X2(k)+dif],[Y1(k),Y2(k)],'Color','r');
end
set(gcf,'Color','w');
function [newLoc1,newLoc2]=pointsSelect(loc1,loc2)
% Filter:filter the paired match points and obtain the fine match points
% Input:
% loc1、 loc2:position of coarse match points
% Output:
% newLoc1、newLoc2:position of fine match points
slope=(loc2(:,1)-loc1(:,1))./(loc2(:,2)-loc1(:,2));
for k=1:3
slope=slope-mean(slope);
len=length(slope);
t=sort(abs(slope));
thresh=t(round(0.5*len));
ind=abs(slope)<=thresh;
slope=slope(ind);
loc1=loc1(ind,:);
```

```
loc2=loc2(ind,:);
end
newLoc1=loc1;
newLoc2=loc2;
function im=picMatched(pic1,newLoc1,pic2,newLoc2)
% Function: obtain the stitching image
% Input:
% pic1、 pic2: images need stitching
% newLoc1、newLoc2:new position of feature points
% Output:
% im: the stitching image
if length(size(pic1))==2
pic1=cat(3,pic1,pic1,pic1);
end
if length(size(pic2))==2
pic2=cat(3,pic2,pic2,pic2);
end
SZ=2000;
X1=newLoc1(:,2);
Y1=newLoc1(:,1);
X2=newLoc2(:,2);
Y2=newLoc2(:,1);
sel=randperm(length(newLoc1),3);
x=X2(sel)';
y=Y2(sel)';
X=X1(sel)';
Y=Y1(sel)';
U=[x;y;ones(1,3)];
V=[X;Y;ones(1,3)];
T=V/U;
cntrX=SZ/2;
cntrY=SZ/2;
im=zeros(SZ,SZ,3);
for i=1:size(pic2,1)
for j=1:size(pic2,2)
tmp=T*[j;i;1];
nx=round(tmp(1))+cntrX;
ny=round(tmp(2))+cntrY;
if nx>=1 && nx<=SZ && ny>=1 && ny<=SZ
im(ny,nx,:)=pic2(i,j,:);
```

```
end
end
end
im=imresize(im,1,'bicubic');
tpic1=zeros(SZ,SZ,3);
tpic1(1+cntrY:size(pic1,1)+cntrY,1+cntrX:size(pic1,2)+cntrX,:)=pic1;
re=rgb2gray(uint8(im))-rgb2gray(uint8(tpic1));
for k=1:3
ta=im(:,:,k);
tb=tpic1(:,:,k);
ta(re==0)=tb(re==0);
im(:,:,k)=ta;
end
clear ta tb re tpic1
im=getPicture(im,SZ);
im=uint8(im);
if length(size(pic1))==2
im=rgb2gray(im);
end
function im=getPicture(pic,SZ)
% Function: obtain the useful image region
% Input
% pic: the stitching image
% SZ: given size
% Output:
% im: useful image region
if length(size(pic))==2
pic=cat(3,pic,pic,pic);
end
k=1;
while k<SZ
if any(any(pic(k,:,:)))
break
end
k=k+1;
end
ceil=k; % Upper boundary
k=SZ;
while k>0
if any(any(pic(k,:,:)))
```

```
break
end
k=k-1;
end
bottom=k; % Lower boundary
k=1;
while k<SZ
if any(any(pic(:,k,:)))
break
end
k=k+1;
end
left=k; %left boundary
k=SZ;
while k>0
if any(any(pic(:,k,:)))
break
end
k=k-1;
end
right=k; %right boundary
%%obtain image
im=pic(ceil:bottom,left:right,:);
```

8.3.3 Auto-Sorting for Image Sequence

In order to stitch the images, the input image sequence [5] is ordered according to the actual scene content, that is to say, each adjacent two images must have overlapping parts, so that the correct panoramic image can be spliced. However, in the process of capturing images and storage or input, sequence of images could be confusing and couldn't stitch directly. In order to implement image sequence auto-sorting, there are 3 problems to solve firstly:

(1) Determine whether there is overlapping regions between two images, that is, whether two images are related;
(2) Determine the head and tail images of the sequence of images;
(3) Determine the relationship between the left and right position of two overlapping images.

In this section, we use phase correlation method to sort the image sequence.
The principle of phase correlation method is as follows:
Suppose there is an offset $(\Delta x, \Delta y)$ between image 1 $I_1(x, y)$ and image 2 $I_2(x, y)$:

$$I_1(x, y) = I_2(x - \Delta x, y - \Delta y) \tag{8.10}$$

According to shift characteristics of Fourier transformation, here is:

$$\hat{I}_1(u, v) = e^{-j2\pi(u\Delta x + v\Delta y)}\hat{I}_2(u, v) \tag{8.11}$$

Its normalized mutual power spectrum is represented as:

$$C\tilde{o}rr(u, v) = \frac{\hat{I}_1(u, v)\hat{I}_2^*(u, v)}{\left|\hat{I}_1(u, v)\hat{I}_2(u, v)\right|} = e^{-j2\pi(u\Delta x + v\Delta y)} \tag{8.12}$$

$\hat{I}_1$ and $\hat{I}_2$ are I_1 and I_2's Fourier transformation separately, $\hat{I}_2^*$ is the complex conjugate of I_2.

The phase of cross power spectrum density equals the phase difference of two images. Normalized cross power spectrum density is operated to get an impulse function through the inverse Fourier transformation:

$$Corr(x, y) = F^{-1}[e^{-j2\pi(u\Delta x + v\Delta y)})] = \delta(x - \Delta x, y - \Delta y) \tag{8.13}$$

This function takes max value at relative displacement (Δx, Δy) (match point) of two images, anywhere else near 0. relative displacement (Δx, Δy) was determined by finding out the position of the peek point in formula (4.4). In the case of only translation between images, the magnitude of the peak of the impulse function reflects the correlation between the two images and take a value in an interval [0, 1]. The larger the overlapping region between two images, the larger the value is. If two images have the same content, the value is 1, and the value is 0 when it is completely different. If there is still perspective, noise or moving target between two images, the energy of the impulse function will be distributed from a single peak to other small peaks, but its maximum peak position has some robustness.

According to the principle of phase correlation method, the automatic sorting algorithm is as follows:

(1) Determine the head and tail images (leftmost and rightmost images) and adjacent image.
For a given image sequence with n images, any image can be computed by the remaining $N - 1$ images to get the $N - 1$ correlation degree. Since the image will be adjacent to up to two images (an intermediate image), it will at least be adjacent to one of the images (head and tail images). Therefore, if the first two largest is selected from the $N - 1$ correlation calculated from the image, the image will overlap with the two or one of the two correlations. Operated on the left $N - 1$ images, $2N$ largest correlation degrees are obtained. For the head image and tail image, their corresponding two correlation degrees will have one not eligible. Obviously, the corresponding correlation degrees of the head image and the tail image are smaller than the other degrees. When finding out the smallest correlation degrees of the $2N$ degrees, the head and tail image are obtained correspondly. Finally, the head and tail image differ from the adjacent images.

(2) Determine the relationship between the left and right positions of two adjacent images.
In the method of phase correlation algorithm, the measure results show the δ pulse function with very sharp correlation peaks when two images are really relevant. The horizontal translation parameters of two images can be calculated by the corresponding pixel points of the peak. When horizontal translation parameter x is greater than half of the image width, you can subtract it from the image width and then take the negative. If the horizontal translation between image **A** and **B** is negative, Image **A** is on the left side of image **B**, and conversely, image **A** is on the right side of image **B**.

Therefore, the automatic sorting of image sequences is completed.

The MATLAB code is shown in PROGRAMME 8.6, where the poc_2pow function has described in PROGRAMME 8.3:

PROGRAMME 8.6: Image sequence automatic sorting

```
clear;clc;
%%%%%%%%%%%%%%%%%%%input images, tectonic image pyramid, store in an array
tic
fprintf('image input, image pyramid building...');
level=2;
T0=uint8([]);T=uint8([]);
%%%%%%%%%%%%%%%%%%%%%%%
i1=imread('X1.jpg');T0(:,:,:,1)=i1;
i2=imread('X2.jpg');T0(:,:,:,2)=i2;
i3=imread('X3.jpg');T0(:,:,:,3)=i3;
[h,w,d]=size(T0(:,:,:,1));%%% same size of images is required
% [T,Terr]=multi_resolution(T0,level);
T=multi_resolution(T0,level);
toc
% %%%%%%%%%%%%%%%%%%%%%%%% calculate offset through phase correlation
tic
fprintf(' calculate offset through phase correlation...');
M=3;%%number of images
% L=M*w;%%total length of images
% suml=0;%%total length of the overlapping region
for N=1:M
if N<M [i,j]=poc_2pow(T(:,:,:,N),T(:,:,:,N+1));
elseif N==M [i,j]=poc_2pow(T(:,:,:,N),T(:,:,:,1));
end
```

```
coor_shift(N,1)=i;
coor_shift(N,2)=j;
%        suml=suml+j;
end
coor_shift=coor_shift*2^level;%%% convert the offsets in the pyramid hierarchy to the offset of the
    original
toc
%%%%%%%%%%%%%%%%%%%% Transform to cylindrical coordinate system
tic
fprintf(' Transform to cylindrical coordinate system...');
f=sqrt(h^2+w^2);
[T1,coor_shift02]=coortransf(T0,f,coor_shift);
toc
%%%%%%%%%%%%%%%merge overlapping parts
tic
fprintf(' merge overlapping parts, stitching image...');
panorama1=T1(:,:,:,1);
for N=1:M
if N<M panorama1=mosaic(panorama1,T1(:,:,:,N+1),coor_shift02(N,1),coor_shift02(N,2));
end
end
toc
%%%%%%%%%%%%%%%%%%reconstruct image
tic
fprintf('save and show result...');
imwrite(panorama1,'pic1.jpg','jpg');
imshow(panorama1,[]);
toc
```

A complete picture is obtained after sorting the input images.

8.3.4 Harris Point Registration Based on RANSAC Algorithm

Harris point registration based on Random Sample Consensus (RANSAC) is a kind of matching method based on features. Harris points detection is firstly performed and then make a rough matching according to local characteristics of extracted points to find out correspondence between sets of points to be matched. After rough matching, most wrong matching points pairs are removed, but there remain many points missing the requirements. These point pairs with large errors in geometrical relationship remains mainly because gray scale information similarity. These points are called pseudo matching pairs. The RANSAC algorithm is used to remove the pseudo matching pairs.

RANSAC is a kind of iteration algorithm to estimate mathematical model parameters. The main idea is to calculate the parameters to make the majority of samples (feature points) can meet the mathematical model. At iteration, the minimum number of samples is used to sample the model and calculate the parameters and the number of samples confirming to the model is counted. And the maximum sample parameters are considered as the values of the final model. The sample point that conforms to the model is called the inliers, and the sample point that does not conform to the model is called the outer point or the wild point.

RANSAC's basic ideas are as follows:

Consider a model required a minimum sampling set with n samples (n for the minimum number of samples required to initializing the model parameters) and a sample set P, numbers of samples of set P #(P) > n. Subset S with n samples which are random extracted from P is used to initialize model M:

The samples in Complement set $SC = P/S$ whose error is less than a set threshold t, along with set S constitute S^*. S^* is considered as inliers set and construct S's Consensus Set.

If #(S^*) $* N$, right parameters are considered obtained and Least Squares method and so on are used to estimate new model M^* on inliers set S^*; or resample new S and repeat.

After a certain number of sampling, if no consistent set is found, the algorithm fails, otherwise the maximal consensus set obtained after sampling, and the algorithm ends. The code is shown in PROGRAMME 8.7.

PROGRAMME 8.7: Harris point registration based on RANSAC algorithm

```
function points = kp_harris(im)
    % Extract keypoints using Harris algorithm (with an improvement
    % version)
    % Author :: Vincent Garcia
    % Date    :: 05/12/2007
    % INPUT
  % im         : the graylevel image
    % OUTPUT
    % points : the interest points extracted
    % REFERENCES
    % C.G. Harris and M.J. Stephens. "A combined corner and edge detector",
    % Proceedings Fourth Alvey Vision Conference, Manchester.
    % pp 147-151, 1988.
    % Alison Noble, "Descriptions of Image Surfaces", PhD thesis, Department
    % of Engineering Science, Oxford University 1989, p45.
    % C. Schmid, R. Mohrand and C. Bauckhage, "Evaluation of Interest Point Detectors",
    % Int. Journal of Computer Vision, 37(2), 151-172, 2000.
    % EXAMPLE
    % points = kp_harris(im)
    % only luminance value
    im = double(im(:,:,1));
    sigma = 1.5;
    % derivative masks
    s_D = 0.7*sigma;
    x    = -round(3*s_D):round(3*s_D);
    dx = x .* exp(-x.*x/(2*s_D*s_D)) ./ (s_D*s_D*s_D*sqrt(2*pi));
    dy = dx';
    % image derivatives
    Ix = conv2(im, dx, 'same');
    Iy = conv2(im, dy, 'same');
    % sum of the Auto-correlation matrix
    s_I = sigma;
    g = fspecial('gaussian',max(1,fix(6*s_I+1)), s_I);
    Ix2 = conv2(Ix.^2, g, 'same'); % Smoothed squared image derivatives
    Iy2 = conv2(Iy.^2, g, 'same');
    Ixy = conv2(Ix.*Iy, g, 'same');
    % interest point response
    cim = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps);                    % Alison Noble measure.
    % k = 0.06; cim = (Ix2.*Iy2 - Ixy.^2) - k*(Ix2 + Iy2).^2;      % Original Harris measure.
    % find local maxima on 3x3 neighborgood
    [r,c,max_local] = findLocalMaximum(cim,3*s_I);
    % set threshold 1% of the maximum value
    t = 0.1*max(max_local(:));
    % find local maxima greater than threshold
    [r,c] = find(max_local>=t);
    % build interest points
    points = [r,c];
end
```

```
function [row,col,max_local] = findLocalMaximum(val,radius)
    % Determine the local maximum of a given value
    %
    % Author :: Vincent Garcia
    % Date      :: 09/02/2007
    %
    % INPUT
    % val        : the NxM matrix containing values
    % radius : the radius of the neighborhood
    % OUTPUT
    % row              : the row position of the local maxima
    % col              : the column position of the local maxima
    % max_local : the NxM matrix containing values of val on unique local maximum
    % EXAMPLE
    % [l,c,m] = findLocalMaximum(img,radius);
    % FIND LOCAL MAXIMA BY DILATION (FAST) /!\ NON UNIQUE /!\
    % mask = fspecial('disk',radius)>0;
    % val2 = imdilate(val,mask);
    % index = val==val2;
    % [row,col] = find(index==1);
    % max_local = zeros(size(val));
    % max_local(index) = val(index);
    % FIND UNIQUE LOCAL MAXIMA USING FILTERING (FAST)
    mask    = fspecial('disk',radius)>0;
    nb        = sum(mask(:));
    highest                    = ordfilt2(val, nb, mask);
    second_highest      = ordfilt2(val, nb-1, mask);
    index                        = highest==val & highest~=second_highest;
    max_local                = zeros(size(val));
    max_local(index) = val(index);
    [row,col]                = find(index==1);
    % FIND UNIQUE LOCAL MAXIMA (FAST)
    % val_height    = size(val,1);
    % val_width      = size(val,2);
    % max_local      = zeros(val_height,val_width);
    % val_enlarge = zeros(val_height+2*radius,val_width+2*radius);
    % val_mask        = zeros(val_height+2*radius,val_width+2*radius);
    % val_enlarge( (1:val_height)+radius , (1:val_width)+radius ) = val;
    % val_mask(        (1:val_height)+radius , (1:val_width)+radius ) = 1;
    % mask    = fspecial('disk',radius)>0;
% row = zeros(val_height*val_width,1);
% col = zeros(val_height*val_width,1);
% index = 0;
% for l = 1:val_height
%       for c = 1:val_width
%             val_ref = val(l,c);
%             neigh_val    = val_enlarge(l:l+2*radius,c:c+2*radius);
%             neigh_mask = val_mask(      l:l+2*radius,c:c+2*radius).*mask;
%             neigh_sort = sort(neigh_val(neigh_mask==1));
%             if val_ref==neigh_sort(end) && val_ref>neigh_sort(end-1)
%                   index                  = index+1;
%                   row(index,1)      = l;
%                   col(index,1)      = c;
%                   max_local(l,c) = val_ref;
%             end
%       end
% end
    % row(index+1:end,:) = [];
    % col(index+1:end,:) = [];
end
```

RANSAC code:

```
function [final_inliers flag bestmodel] = AffinePairwiseRansac(frames_a1, frames_a2, all_matches)
% iterations = 0
% bestfit = nil
% besterr = something really large
% while iterations < k {
%       maybeinliers = n randomly selected values from data
%       maybemodel = model parameters fitted to maybeinliers
%       alsoinliers = empty set
%       for every point in data not in maybeinliers {
%             if point fits maybemodel with an error smaller than t
%                   add point to alsoinliers
%       }
%       if the number of elements in alsoinliers is > d {
%             this implies that we may have found a good model
%             now test how good it is
%             bettermodel = model parameters fitted to all points in maybeinliers and alsoinliers
%             thiserr = a measure of how well model fits these points
%             if thiserr < besterr {
%                   bestfit = bettermodel
%                   besterr = thiserr
%           }
%       }
%       increment iterations
% }
% return bestfit
%
%first decide how many matches we have
MIN_START_VALUES = 4;
num_matches = size(all_matches,2);
if (num_matches < MIN_START_VALUES)
    final_inliers = [];
    bestmodel = [];
    flag = -1;
    return
end
% Todo? These might have to be changed if the values are different.
Z_OFFSET = 640;
COND_THRESH = 45;
% RANSAC parameters
NUM_START_VALUES = 3;   % only 3 corrospondences needed for determining affine model
K = 50;
ERROR_THRESHOLD = 10; % fairly high threshold - this is in number of pixels
D = 1; % additional points must fit any given affine model
N = NUM_START_VALUES;
RADIUS = 30; %changed to 30 by vijay
MIN_NUM_OUTSIDE_RADIUS = 1;
%best error, best fit
iteration = 0;
besterror = inf;
bestmodel = [];
final_inliers = [];
max_inliers = 0;
while (iteration < K)
    %start with NUM_START_VALUES unique values
    uniqueValues = [];
    max_index = size(all_matches, 2);
    while (length(uniqueValues) < NUM_START_VALUES)
        value = ceil(max_index*rand(1,1));
        if (length(find(value == uniqueValues)) == 0)
            %unique non-zero value
            uniqueValues = [uniqueValues value];
        end
  end
  %uniqueValues are the indices in all_matches
  maybeinliers = all_matches(:, uniqueValues);    %start with NUM_START_VALUES unique
random values
  % make sure points are well distributed
  point_matrix = [frames_a1(:, maybeinliers(1, :)); Z_OFFSET*ones(1, NUM_START_VALUES)];
  if (cond(point_matrix) > COND_THRESH)
        iteration = iteration + 1;
        continue;
  end
  M_maybemodel = getModel(maybeinliers, frames_a1, frames_a2);
  if (prod(size(M_maybemodel)) == 0)
        iteration = iteration + 1;
        continue;
  end
  alsoinliers = [];
  %figure out other inliers
  for i = 1:size(all_matches, 2)
        temp = find(all_matches(1,i) == maybeinliers(1,:));
        if (length(temp) == 0)
              %this means, point not in maybeinlier
              a1 = frames_a1(1:2, all_matches(1,i));
              a2 = frames_a2(1:2, all_matches(2,i));
              if (getError(M_maybemodel, a1, a2) < ERROR_THRESHOLD )
                    alsoinliers = [alsoinliers all_matches(:,i)];
              end
        end
  end
  if (size(alsoinliers,2) > 0)
        num = 0;
        dist = [];
        for i = 1:NUM_START_VALUES
              diff =      frames_a1(1:2,   alsoinliers(1, :)) - ...
                    repmat(frames_a1(1:2, maybeinliers(1, i)), [1, size(alsoinliers, 2)]);
              dist = [dist; sqrt(sum(diff.^2))];
        end
        num = sum(sum(dist > RADIUS) == NUM_START_VALUES);
        if (num < MIN_NUM_OUTSIDE_RADIUS)
              iteration = iteration+1;
        continue;
    end
end
%see how good the model is
%fprintf('Number of elements in also inliers %d\n', size(alsoinliers,2));
if (size(alsoinliers,2) > D)
    %this implies that we have found a good model
    %now let's see how good it is
    %find new model
    all_inliers = [maybeinliers alsoinliers];
    M_bettermodel = getModel(all_inliers, frames_a1, frames_a2);
    %the new model could be bad
    if (prod(size(M_bettermodel)) == 0)
        iteration = iteration+1;
        continue;
        end
        %find error for the model
        thiserror = getModelError(M_bettermodel,all_inliers, frames_a1, frames_a2);
        if max_inliers < size(all_inliers, 2) | (thiserror < besterror & max_inliers == size(all_inliers, 2))
            bestmodel = M_bettermodel;
            besterror = thiserror;
            final_inliers = all_inliers;
            max_inliers = size(final_inliers, 2);
        end
    end
    %do it K times
    iteration = iteration + 1;
end
%bestmodel has the best Model
if (prod(size(bestmodel)) ~= 0)
    % a model was found
    fprintf('Error of best_model ~%f pixels\n', besterror);
    flag = 1;
else
    flag = -1;
    final_inliers = [];
    bestmodel = [];
    fprintf('No good model found !\n');
end
end
```

```
function error = getModelError(M ,matches, frames_a1, frames_a2)
        nummatches = size(matches,2);
        error = 0;
        for i = 1:nummatches
                a1 = frames_a1(1:2, matches(1,i));
                a2 = frames_a2(1:2, matches(2,i));
                error = error + getError(M, a1, a2);
        end
        error = error/nummatches;
 end
function M = getModel(matches, frames_a1, frames_a2)
        %let's go from 1 to 2 -- changed on Apr 28 to be consistent
        %with epipolar and perspective models
        singular_thresh = 1e-6;
        scaling_ratio_thresh = 5;
        scale_thresh = 0.005;
        % approximate M
        M = zeros(3,3);
        Y = []; X = [];
          for i = 1:size(matches,2)
            a1 = frames_a1(1:2, matches(1,i));
            a2 = frames_a2(1:2, matches(2,i));
            Y = [Y; a2];
            X = [X; a1(1) a1(2) 1 0 0 0; 0 0 0 a1(1) a1(2) 1];
        end
          %to check if matrix is singular
        if (1/cond(X) < singular_thresh)
            M = [];
            return
        end
          M = X\Y;
        %we need to return M - a 3X3 matrix, where the last row is (0 0 1)
        M = [reshape(M, 3,2)'; 0 0 1];
        %let's add some rules to remove any crazy map
        %we definitely cannot have reflection
        [u, s, v] = svd(M(1:2,1:2));
        if (det(u*v') < 0)
            %==> there is a reflection
            M=[];
```

```
%                fprintf('Special case to avoid reflection\n');
              return
         end
         %we cannot have crazy ratios of scaling in the two dimensions.
         if (cond(M(1:2,1:2)) > scaling_ratio_thresh)
              %==> the matches are bad
              M=[];
%                fprintf('Special case to avoid crazy scaling ratio\n');
              return
         end

         %check for crazy zoom
         if (s(1,1) < scale_thresh | s(2,2) < scale_thresh)
             M = [];
         end
end
function error = getError(M, a1, a2)
         %a2_model is the value of a2 that comes from the model
         %calculate mapping error
         a2_model    = M*([a1;1]);    %3x1 vector, only the first two values matter
         error = dist(a2, a2_model(1:2));
end
function d =    dist(one, two)
     d = sqrt(sum((one-two).^2));
end
function [final_inliers flag bestmodel] = PerspectivePairwiseRansac(frames_a1, frames_a2, all_matches)
     % iterations = 0
     % bestfit = nil
     % besterr = something really large
     % while iterations < k {
     %        maybeinliers = n randomly selected values from data
     %        maybemodel = model parameters fitted to maybeinliers
     %        alsoinliers = empty set
     %        for every point in data not in maybeinliers {
     %             if point fits maybemodel with an error smaller than t
     %                   add point to alsoinliers
     %        }
     %        if the number of elements in alsoinliers is > d {
     %             this implies that we may have found a good model
     %             now test how good it is
```

```
    %               bettermodel = model parameters fitted to all points in maybeinliers and alsoinliers
    %               thiserr = a measure of how well model fits these points
    %               if thiserr < besterr {
    %                   bestfit = bettermodel
    %                   besterr = thiserr
    %               }
    %           }
    %       increment iterations
    % }
    % return bestfit
    %
    %finds the model from the first image to the second image
%
%       [XW] = [a b c][x]
%       [YW] = [d e f][y]
%       [W]   = [g h 1][1]
%(x,y,1) are points in the first image and they map to (XW, YW, W) in
%the second image
%first decide how many matches we have
MIN_START_VALUES = 20;
num_matches = size(all_matches,2);
if (num_matches < MIN_START_VALUES)
    final_inliers = [];
    bestmodel = [];
    flag = -1;
    return
end
%RANSAC parameters
K = 150;
NUM_START_VALUES = 4;   % using 4 point least squares solution
ERROR_THRESHOLD = 10; % this is in the number of pixels.
D = 8; %start with 4 points and fit atleast 8 more fit the model
%best error, best fit
iteration = 0;
besterror = inf;
bestmodel = [];
final_inliers = [];
max_inliers = 0;
while (iteration < K)
    %start with NUM_START_VALUES unique values
    uniqueValues = [];
    max_index = size(all_matches, 2);
    while (length(uniqueValues) < NUM_START_VALUES)
        value = ceil(max_index*rand(1,1));
        if (length(find(value == uniqueValues)) == 0)
            %unique non-zero value
            uniqueValues = [uniqueValues value];
        end
    end
    %uniqueValues are the indices in all_matches
    maybeinliers = all_matches(:, uniqueValues);   %start with NUM_START_VALUES unique
random values
    M_maybemodel = getModel(maybeinliers, frames_a1, frames_a2);
    if (prod(size(M_maybemodel)) == 0)
        iteration = iteration + 1;
        continue;
end
alsoinliers = [];
%figure out other inliers
for i = 1:size(all_matches, 2)
    temp = find(all_matches(1,i) == maybeinliers(1,:));
    if (length(temp) == 0)
        %this means, point not in maybeinlier
        a1 = frames_a1(1:2, all_matches(1,i));
        a2 = frames_a2(1:2, all_matches(2,i));
        if (getError(M_maybemodel, a1, a2) < ERROR_THRESHOLD )
            alsoinliers = [alsoinliers all_matches(:,i)];
        end
    end
end
%see how good the model is
%fprintf('Number of elements in also inliers %d\n', size(alsoinliers,2));
if (size(alsoinliers,2) > D)
    %this implies that we have found a good model
    %now let's see how good it is
    %find new model
    all_inliers = [maybeinliers alsoinliers];
    M_bettermodel = getModel(all_inliers, frames_a1, frames_a2);
    %the new model could be bad
    if (prod(size(M_bettermodel)) == 0)
```

```
                iteration = iteration+1;
                continue;
            end
            %find error for the model
            thiserror = getModelError(M_bettermodel,all_inliers, frames_a1, frames_a2);
            if max_inliers < size(all_inliers, 2) | (thiserror < besterror & max_inliers ==
size(all_inliers, 2))
                bestmodel = M_bettermodel;
                besterror = thiserror;
                final_inliers = all_inliers;
                max_inliers = size(final_inliers, 2);
            end
        end
        %do it K times
        iteration = iteration + 1;
  end
        if (prod(size(bestmodel)) ~= 0)
            % a model was found
            fprintf('Error of best_model ~%f pixels\n', besterror);
            flag = 1;
          else
            flag = -1;
            final_inliers = [];
            bestmodel = [];
            fprintf('No good model found !\n');
        end
  end
  function error = getModelError(M ,matches, frames_a1, frames_a2)
            nummatches = size(matches,2);
            error = 0;
              for i = 1:nummatches
                    a1 = frames_a1(1:2, matches(1,i));
                    a2 = frames_a2(1:2, matches(2,i));
                    error = error + getError(M, a1, a2);
            end
            error = error/nummatches;
  end
  function P = getModel(matches, frames_a1, frames_a2)
        %goes from 1 to 2
        %let's use a least squares approach. Referenced from
```

```
    %http://alumni.media.mit.edu/~cwren/interpolator
    nummatches = size(matches,2);
    LHS= [];
    for i = 1:nummatches
        a1 = frames_a1(1:2, matches(1,i)); x = a1(1); y=a1(2);
        a2 = frames_a2(1:2, matches(2,i)); X = a2(1); Y=a2(2);
        LHS = [LHS; x y 1 0 0 0 -X*x -X*y; 0 0 0 x y 1 -Y*x -Y*y];
    end
    RHS = reshape(frames_a2(1:2, matches(2, :)), nummatches*2, 1);
    P = reshape([(LHS\RHS);1], 3,3)';
    %to get P in the form
%       [a b c; d e f; g h 1];
end
function error = getError(P, a1, a2)
    %the model F goes from image 1 to image 2
    %the corresponding point for a1
    temp = P*[a1;1];
    a2_model(1) = temp(1)/temp(3);
    a2_model(2) = temp(2)/temp(3);
    error = dist(a2_model', a2);
end
  function d =   dist(one, two)
    d = sqrt(sum((one-two).^2));
end
```

8.4 Panoramic Image Stitching

Panoramic image stitching aims to do seamless stitching on image sequence taken from the same scene, different perspective, different focal lengths, from the same optical center with partially overlapping. This means that image registration algorithm is used to calculate the motion parameters between each frame and then synthetic a large static wide-angle image. Moreover, the stitching image requires to be as close as the real scene without obvious seam. According to different viewpoints, image stitching can be divided into algorithm based on single viewpoint and algorithm based on multiple viewpoints. To obtain single viewpoint image sequences, a camera is fixed in a position and rotate it around; or, set cameras in a circle, the optical axis of the camera is on the same plane and intersects with one point, and the video is collected in real time. To obtain multiple viewpoints image sequences, usually use a camera capture a set of image sequences on a horizontal level or set multiple cameras in different positions and capture at the same time. Image stitching algorithm based on multiple viewpoints is commonly used in stitching banded panoramic images.

This section introduces an image stitching algorithm based on image projection transformation without active objects.

Discrete image information can only express information on a part of the visual environment. The panoramic view based on image rendering is to show the discrete image information in an image completely. Build a complete graphics environment for better 3D visual effects.

(1) Image positioning

Automatically find overlapping locations for images. Proposing there are two rectangle regions A and B. B contains a region A_2. A_2 and A are same module, the position of A_2 in B is to solve. The typical algorithm is to search from the lower left corner of B, where each piece is compared to a with the same area C and A, and the value of the evaluation function, the smallest area is A_2.

(2) Image stitching

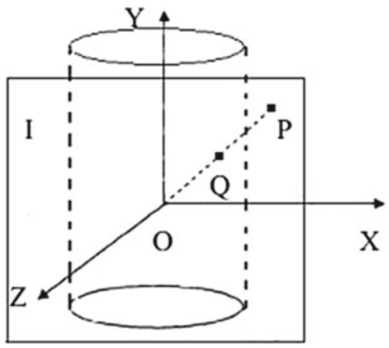
After image positioning, if splice the two images simply, there will be a clear seam due to the difference of brightness. The color fitting method can be used to reconcile the brightness of adjacent images and produce seamless synthetic images.

(3) Implementation of cylindrical projection

Shot at the same point as the rotating camera, the cylindrical panoramic images are not in the same coordinate system. There is a certain angle on the projection surface. In order to generate a panoramic image, we must transform these images into a unified cylindrical coordinate system and use image stitching technology to remove the overlap of every two images. In this way, a complete cylindrical panoramic picture is obtained.

As shown in Fig. 8.13, it's the positive projection diagram of cylindrical surface. I is a frame extracted from the video, P is any point on the captured image, Q is the point where P maps to the cylinder coordinate.

Fig. 8.13 The positive projection diagram of cylindrical surface



Assuming that W and H are the width and height of the Image I respectively, f is the radius of the cylinder, so that P's coordinate in 3D coordinate system is represented as $(x - W/2, y - H/2, f)$. Using the combination of the parametric equation and the cylindrical equation, assuming Q's coordinate is (x', y', z') and P as well as Q are in the same line, which satisfies the parametric equation:

$$\begin{cases} x' = t(x - W/2) \\ y' = t(y - H/2) \\ z' = tf \end{cases} \tag{8.14}$$

where t is the parameter, coupled with the cylindrical surface equation: $x'^2 + z'^2 = f^2$.

Thus, we can get the coordinates of Q point because the coordinates of Q point is three-dimensional, we convert it into two dimensions to get:

$$\begin{cases} x' = f * \arctan\left[(x - W/2)/f\right] + f * \arctan(W/2f) \\ y' = f(y - H/2)/\sqrt{(x - W/2)^2 + f^2} + H/2 \end{cases} \tag{8.15}$$

After the image is projected to the cylindrical plane, the images of the same coordinate system are obtained. Then by looking for the transformation between adjacent images, the sequence images are spliced together to form a cylindrical panoramic image under the same scene.

The steps of using IBR method to splice video panoramic image are as follows:

(1) Extract key frames of the video and use images to represent information in videos.
(2) Find out the overlapping region of images, this means that extract feature position.
(3) Image registration, match feature points. Use fine match algorithm to remove wrong point pairs and moving corner point pairs. The coordinate transformation function is obtained by calculating the transformation matrix between the datum image and the image to be matched. Finally, the coordinate transformation function is used to transform the image to the datum coordinate system and realize the registration of the image to be matched with the datum image in the same coordinate system.
(4) The final step is image stitching, involving the fusion of two images and the elimination of seams.

The code for reading a video and extracting key frames is shown in PROGRAMME 8.8.

PROGRAMME 8.8: Read video and extract the key frames

```
xyloObj = VideoReader('test.avi');%read video
nFrames = xyloObj.NumberOfFrames;      %number of frames
count=1;        %count the extracted frames
letter='a'; %tag，to make subsequent read sequences normal, precede the Arabic numerals with letters
for k = 1 : 5: nFrames
    mov(k).cdata =    read(xyloObj,k);        %image color data
    strtemp = strcat('images/',letter+count/10);
    strtemp = strcat(strtemp, int2str(count));
    strtemp = strcat(strtemp,'.jpg');
    count=count+1;
    imwrite(mov(k).cdata,strtemp);%save as strtemp.jpg
end
%feature transformation method based on color——using color histogram to measure color feature
%pop up a few key frames
filenames=dir('images/*.jpg');           %image source
num=size(filenames,1);                       %number of images
key=zeros(1,num);                              % (0,1) key frame array
count=0;                                         %save a few key frames
threshold=0.75;                                %set threshold
if num==0
    error('Sorry, there is no pictures in images folder!');
else                                             %set the first frame as key frame
    img=imread(strcat('images/',filenames(1).name));
    key(1)=1;
    count=count+1;
    %obtain RGB histogram
    [preCountR,x]=imhist(img(:,:,1));      %red histogram
    [preCountG,x]=imhist(img(:,:,2));      %green histogram
    [preCountB,x]=imhist(img(:,:,3));      %blue histogram
    %show first key frame
    figure(count);
    imshow('images/a1.jpg');
    for k=2:num
        img=imread(strcat('images/',filenames(k).name));
        [newCountR,x]=imhist(img(:,:,1));      %red histogram
        [newCountG,x]=imhist(img(:,:,2));      %green histogram
        [newCountB,x]=imhist(img(:,:,3));      %blue histogram
        sR=0;
        sG=0;
        sB=0;
```

```
%use method of color histograms
for j=1:256
   sR=min(preCountR(j),newCountR(j))+sR;
   sG=min(preCountG(j),newCountG(j))+sG;
   sB=min(preCountB(j),newCountB(j))+sB;
end
dR=sR/sum(newCountR);
dG=sG/sum(newCountG);
dB=sB/sum(newCountB);
            %YUV,persons are sensitive to Y
            d=0.30*dR+0.59*dG+0.11*dB;
            if d<threshold              %small similarity, new keyframes found
                  key(k)=1;              %set as keyframes
                  count=count+1;
                  figure(count);
                  imshow(strcat('images/',filenames(k).name));
                  %nearest update color histogram
                  preCountR=newCountR;
                  preCountG=newCountG;
                  preCountB=newCountB;
            end
      end
 end
 keyFrameIndexes=find(key)
```

Panoramic image stitching based on IBR method using key frames, whose code is programme 8.9, and the functions involved are described in programme 8.3.

PROGRAMME 8.9: Panoramic image stitching based on IBR method

```
clear;clc;
%%%%%%%%%%%%%%%%input images, tectonic image pyramid, save an array
tic
fprintf(' input images, tectonic image pyramid,...');
level=3;
T0=uint8([]);T=uint8([]);
%%%%%%%%%%%%%%%%%%%%%
i1=imread('1.jpg');T0(:,:,:,1)=i1;
i2=imread('2.jpg');T0(:,:,:,2)=i2;
i3=imread('3.jpg');T0(:,:,:,3)=i3;
```

```
i4=imread('4.jpg');T0(:,:,:,4)=i4;
i5=imread('5.jpg');T0(:,:,:,5)=i5;
i6=imread('6.jpg');T0(:,:,:,6)=i6;
i7=imread('7.jpg');T0(:,:,:,7)=i7;
i8=imread('8.jpg');T0(:,:,:,8)=i8;
[h,w,d]=size(T0(:,:,:,1));%%%same image size is required
% [T,Terr]=multi_resolution(T0,level);
T=multi_resolution(T0,level);
toc
% %%%%%%%%%%%%%%%%%%%%%% calculate offset through phase correlation
tic
fprintf(' calculate offset through phase correlation..');
M=8;%% number of images
% L=M*w;%% total length of images
for N=1:M
    if N<M [i,j]=poc_2pow(T(:,:,:,N),T(:,:,:,N+1));
    elseif N==M [i,j]=poc_2pow(T(:,:,:,N),T(:,:,:,1));
    end
    coor_shift(N,1)=i;
    coor_shift(N,2)=j;
end
coor_shift=coor_shift*2^level;%%% convert the offsets in the pyramid hierarchy to the offset of the
  original
toc
%%%%%%%%%%%%%%%%%% Transform to cylindrical coordinate system
tic
fprintf(' Transform to cylindrical coordinate system..');
f=sqrt(h^2+w^2);
[T1,coor_shift02]=coortransf(T0,f,coor_shift);
toc
%%%%%%%%%%%%% merge overlapping parts
tic
fprintf(' merge overlapping parts, stitching image....');
panorama1=T1(:,:,:,1);
for N=1:M
    if N<M
panorama1=mosaic(panorama1,T1(:,:,:,N+1),coor_shift02(N,1),coor_shift02(N,2));
    end
```

```
end
toc
%%%%%%%%%%%%%%%% reconstruct image
tic
fprintf(' save and show result..');
toc
image1=rgb2gray(panorama1);
index=find(image1(:,1)>=255);
aa=max(index);
[r,c]=size(image1)
image1=imcrop(panorama1,[1,aa,c,r]);
imshow(image1);
imwrite(image1,'pic1.jpg','jpg');
```

Key frames extracted is shown in Fig. 8.14.
Panoramic stitching image is shown in Fig. 8.15.

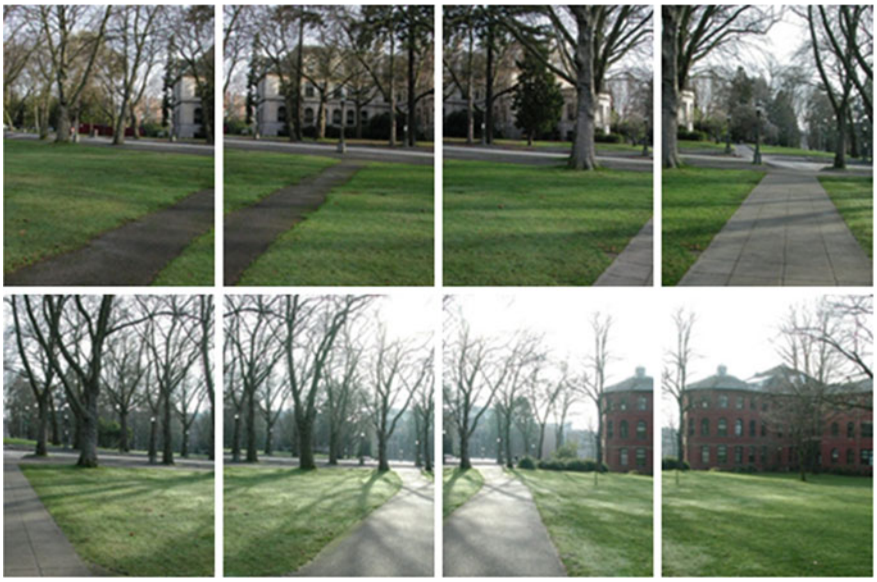


Fig. 8.14 Key frames extracted

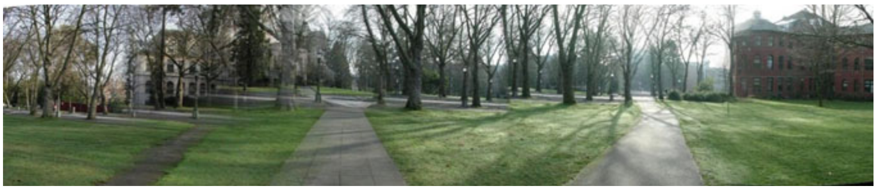


Fig. 8.15 The panoramic stitching image

References

1. Yanyan L, Xu S (2008) Study of image stitching algorithm based on ratio matching. Electron Meas Technol
2. Le-Fu WU, Ding GT (2010) Region-based images stitching algorithm. Comput Eng Design 31(18):4043–4044
3. Zhou DF, Ming-Yi HE, Yang Q (2009) A robust seamless image stitching algorithm based on feature points. Meas Control Technol 28(6):32–36
4. Chandratre R, Chakkarwar VA (2014) Image stitching using Harris and RANSAC. Int J Comput Appl 89(15):14–19
5. Zhao WJ, Gong SR, Liu Q et al (2007) An auto-sorting arithmetic for image sequence used in image Mosaics. J Image Graph 12(10):1861–1864