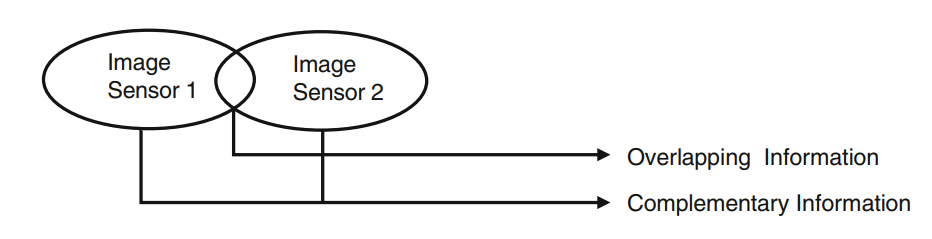
Chương 8: Ghép ảnh

Tóm tắt - Trong chương này, trước tiên chúng tôi giới thiệu nền ứng dụng và quy trình cơ bản của việc ghép ảnh, sau đó mô tả một số phương pháp ghép ảnh dựa trên vùng, các phương pháp ghép ảnh dựa trên điểm đặc trưng và kỹ thuật ghép ảnh video toàn cảnh.

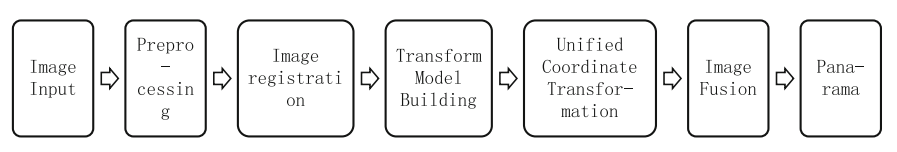
8.1 Giới Thiệu

Trong thực tế, nó thường cần hình ảnh toàn cảnh có góc nhìn rộng và độ phân giải cao, nhưng kích thước của hình ảnh phụ thuộc vào hiệu suất của máy ảnh do giới hạn của thiết bị chụp ảnh. Do đó, một cách tiếp cận sử dụng phần mềm máy tính để ghép ảnh sau đó đã được đưa ra để tạo ảnh toàn cảnh. Ghép hình ảnh đề cập đến việc ghép một số hình ảnh có các phần chồng lên nhau thành một hình ảnh lớn, liền mạch và có độ phân giải cao. Hình 8.1 hiển thị bản đồ phác thảo của ghép hình ảnh. Nói chung, ghép ảnh chủ yếu bao gồm các 5 bước:

1. Xử lý trước hình ảnh. Nó chứa các hoạt động cơ bản của xử lý hình ảnh kỹ thuật số (chẳng hạn như khử nhiễu, trích xuất cạnh và xử lý histogram), thiết lập các mẫu ảnh phù hợp, biến đổi hình ảnh (FT, WT, v.v.) và các hoạt động khác.
2. Đăng ký hình ảnh. Nó áp dụng một số loại thuật toán đối sánh để xác định vị trí tương ứng của các mẫu hoặc điểm đặc trưng trong các hình ảnh ghép để xác định mối quan hệ chuyển đổi giữa hai hình ảnh.
3. Xây dựng mô hình biến đổi. Một mô hình biến đổi toán học có thể được xây dựng giữa hai hình ảnh bằng cách tính toán các thông số của mô hình dựa trên sự tương ứng của các mẫu hình ảnh hoặc các đặc trưng.
4. Phép biến đổi tọa độ thống nhất. Theo mô hình biến đổi toán học được xây dựng ở bước 3, ảnh được ghép sẽ được chuyển vào hệ tọa độ của ảnh tham chiếu để thực hiện phép biến đổi tọa độ thống nhất.



Hình 1: Sơ đồ phác thảo ghép ảnh



Hình 2: Lưu đồ ghép ảnh

(5) Hợp nhất và tái tạo hình ảnh. Hợp nhất các phần chồng chéo của hình ảnh được ghép thành một bức ảnh toàn cảnh được tái tạo liền mạch và mượt mà.

Đăng ký hình ảnh là chìa khóa cho các thuật toán ghép hình ảnh. Theo các phương pháp đăng ký ảnh khác nhau, các thuật toán ghép ảnh có thể được phân loại thành hai loại: ghép ảnh dựa trên khu vực và ghép ảnh dựa trên các điểm đặc trưng.

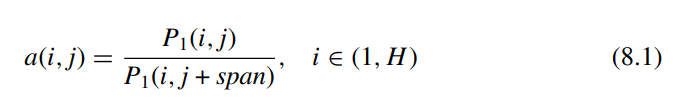
8.2 Ghép ảnh dựa trên khu vực

Việc ghép ảnh dựa trên khu vực bắt đầu từ việc so sánh các giá trị thang độ xám của một vùng trong ảnh sẽ được ghép với vùng trong ảnh được tham chiếu có cùng kích thước bằng các phương pháp bình phương nhỏ nhất và các phương pháp toán học khác. Từ các phép so sánh, chúng ta có thể đo lường mức độ giống nhau giữa các vùng chồng chéo trong hình ảnh được ghép và lấy phạm vi và vị trí của vùng chồng chéo trong hình ảnh sẽ được ghép để hoàn thành nhiệm vụ ghép ảnh. Chúng tôi cũng có thể chuyển đổi hình ảnh từ miền không gian thành miền tần số với FFT và vận hành đăng ký hình ảnh sau đó. Đối với các ảnh có độ dịch chuyển lớn, chúng ta có thể điều chỉnh việc xoay ảnh và sau đó thiết lập ánh xạ giữa hai ảnh. Khi lấy sự khác biệt giữa các giá trị thang độ xám của pixel ở hai vùng làm tiêu chí, cách tiếp cận đơn giản nhất là cộng trực tiếp sự khác biệt theo từng pixel. Một cách khác là tính toán hệ số tương quan giữa các giá trị thang độ xám pixel của hai khu vực. Hệ số tương quan càng lớn thì mức độ khớp của hai hình ảnh càng cao và cách này cho thấy hiệu suất tốt hơn cũng như tỷ lệ thành công cao hơn. Ngày nay, các thuật toán ghép ảnh dựa trên vùng thường được sử dụng bao gồm Khớp tỷ lệ, Khớp dựa trên khối, Khớp dòng và Khớp lưới.

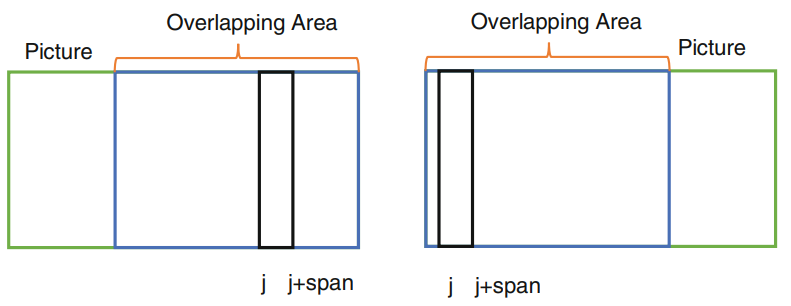
8.2.1 Ghép ảnh dựa trên khớp tỷ lệ

Việc ghép ảnh dựa trên phù hợp tỷ lệ, trước tiên chọn tỷ lệ hai cột pixel với khoảng cách nhất định giữa các phần được chồng lên nhau của hình ảnh làm mẫu [ 1 ]. Sau đó, tìm kiếm kết quả phù hợp nhất cho vùng chồng chéo trong hình ảnh thứ hai và tìm hai cột tương ứng với mẫu được lấy từ hình ảnh đầu tiên để hoàn thành việc ghép ảnh. Hình 8,3 là một bản phác thảo của thuật toán. Hình 1 là viết tắt của một hình ảnh ( W 1 × H) tính bằng pixel và Hình ảnh 2 là ( W 2 × H). W 1 và W 2 có thể bằng nhau hoặc không. Hình 1 nằm bên trái Hình 2. Một tình huống khác mà hình ảnh là chồng chéo theo chiều dọc sẽ không được thảo luận trong chương này vì chúng ta có thể xử lý nó theo cách tương tự. Sau đây là các bước của thuật toán này:

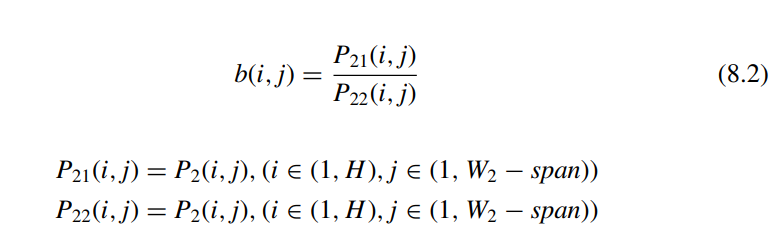
1. Chọn hai cột pixel với khoảng cách từ khu vực chồng chéo của hình 1, tính toán tỷ lệ pixel tương ứng làm mẫu a.



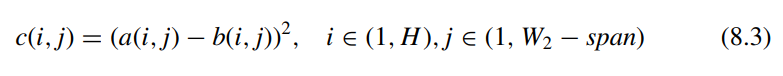
1. Trong Hình 2, lần lượt chọn hai cột với khoảng cách khoảng từ cột đầu tiên, tỷ lệ các pixel tương ứng của nó được tính dưới dạng mẫu b.



Hình 3: Sơ đồ chọn mẫu



1. Tính toán sự khác biệt giữa các mẫu a và b như mẫu c.



1. c là một mảng hai chiều. Thêm từng vectơ cột vào một mảng khác gọi là tổng: . Giá trị của sum (j) tái tạo ra sự khác biệt của các cột đã chọn trong hai hình ảnh. Tọa độ cột của sum (j) tối thiểu Sum(min) là vùng tốt nhất.

Code:

clear**;**

clc**;**

%A=imread('lenna\_left.png');% read Picture 1 A represents the pixel array of Picture 1

%B=imread('lenna\_right.png');% read Picture 2

A**=**imread**(**'left1.jpg'**);**% read Picture 1 A represents the pixel array of Picture 1

B**=**imread**(**'right1.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

zz**=**b**(**1**,:);**

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'**);**

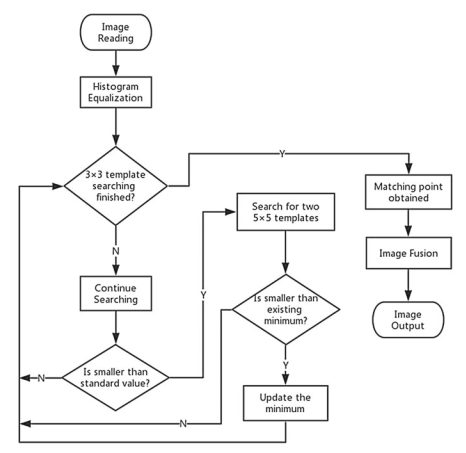
imshow**(**mat2gray**(**C**));**

Kết quả:

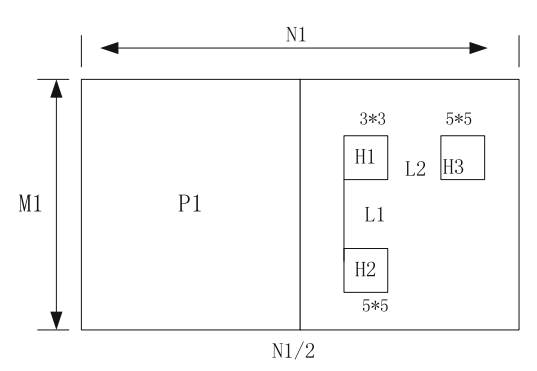
8.2.2 Ghép ảnh dựa trên đặc điểm đường thẳng và mặt phẳng

Ghép ảnh dựa trên của đường và mặt phẳng bao gồm: xử lý hình ảnh, tìm kiếm khối tính năng, ghép hình ảnh và kết hợp hình ảnh. Hình 8.5 hiển thị biểu đồ.

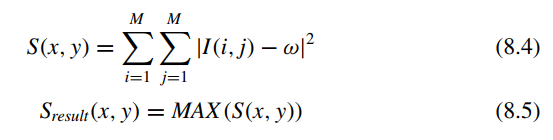


Hình 4: Lưu đồ ghép ảnh sử dụng đặc trưng cạnh và mặt phẳng

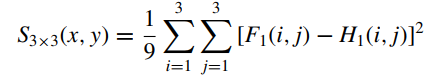
1. Xử lý hình ảnh. Do độ chiếu sáng khác nhau nên rất dễ làm lỗi ghép ảnh nếu ảnh thô thu được trực tiếp từ máy ảnh được ghép. Quá trình cân bằng histogram là một cách hiệu quả để làm giảm bớt các tác động của ánh sáng. Sau khi áp dụng cân bằng histogram cho hai hình ảnh được ghép, biểu đồ thang độ xám của hai hình ảnh được trải rộng thành tất cả các phạm vi mức xám và sự khác biệt về độ chiếu sáng trong các hình ảnh liền kề được giảm đi một cách đáng kể, điều này sẽ giúp việc ghép hình ảnh dễ dàng nhận ra.



Hình 5: Trích xuất nhóm tính năng của mẫu



1. Tìm kiếm khu vực tính năng. Chúng ta lấy P 1 như hình ảnh được tham chiếu và P 2 như hình ảnh được ghép. Kích thước của P 1 Là ( M 1 × N 1) và Hình 2 là ( M 2 × N 2). Thuật toán này sẽ chọn 3 mẫu tính năng nhỏ trong P 1 để phù hợp. Trước hết, giới hạn vùng dùng để chọn mẫu là từ dòng 1 đến dòng M 1 và cột N 1/2 đến cột N 1 trong P 1. Trước tiên, chúng tôi sẽ chọn một 3 × 3 mẫu nhỏ có tên H 1 trong khu vực này. Sau đó, theo các tính năng của hình ảnh, chúng tôi chọn hai mẫu còn lại 5 × 5 có tên H 2 và H 3 tương ứng trong khu vực. Như thể hiện trong Hình. 8.6 , một nhóm mẫu tính năng bao gồm 3 mẫu nhỏ được tạo thành. (Lưu ý: Chúng tôi giả sử rằng H 2 và H 1 ở cùng một mức theo hướng ngang và như vậy H 3 và H 1 theo phương thẳng đứng và khoảng cách từ Hl tương ứng là Ll và L2.) Chúng tôi áp dụng phương pháp tính toán các giá trị phương sai của pixel trong các mẫu khi chọn mẫu tính năng. Chúng tôi chọn mẫu có tổng phương sai tối đa làm mẫu tiêu chuẩn vì các đặc điểm chi tiết trong hình ảnh được xác định bởi các đặc điểm cạnh hoặc trong các điểm chỉnh sửa của giá trị thang độ xám. Trong đó tổng phương sai lớn nhất bằng với vị trí của các cạnh hoặc tại điểm tạo ra dao động hiển thị nhiều nhất trong các đường cong của mức xám. Chúng tôi có thể đo lường có bao nhiêu tính năng chi tiết trong một mẫu bằng các tổng các giá trị pixel. Các chi tiết hơn và thông tin kết cấu P 1, càng dễ dàng tìm thấy các khu vực tương tự trong P 2. Dưới đây là các phương trình để chọn mẫu tính năng. Trong Eq. ( 8,4 ), I(i, j) đại diện cho giá trị pixel và ω là viết tắt của giá trị màu xám trung bình của mẫu. Khi đặt M là 3, chúng ta có thể thu được mẫu 3 × 3 tốt nhất thông qua Eqs. 8,4 và 8.5 . Lặp lại phép tính hai lần để lầy 2 mẫu 5 × 5. Bằng cách này, 3 mẫu tính năng có chi tiết tốt nhất sẽ được trích xuất. Ghi lại thông tin khoảng cách xung quanh chúng để tạo nhóm mẫu tính năng. Sau khi trích xuất các nhóm mẫu tính năng thích hợp, tìm kiếm từ trên xuống dưới cùng và từ trái sang phải với mẫu 3 × 3 Fl trong P 2, và tính toán pixel sự khác biệt giữa F 1 và H 1 từng cái một. Hàm MSE được sử dụng để xác định hàm khác biệt. Đây là định nghĩa:

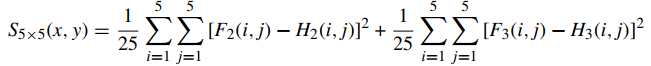


Với F1 ( i, j) là viết tắt của giá trị màu xám của các pixel tương ứng trong F1 và như vậy H1 ( (i, j) cho H1.

Kết quả thí nghiệm chỉ ra rằng khi sự chênh lệch của cường độ ánh sáng trong hình ảnh có kích thước nhỏ, việc đặt giá trị tiêu chuẩn của hàm khác biệt thành 30 là một giá trị thích hợp nếu mẫu tính năng đã chọn là 3 × 3. Khi giá trị tiêu chuẩn đã chọn lớn hơn 30, số lượng mẫu thiết lập các điều kiện đang tăng lên nhanh chóng, điều này sẽ dẫn đến thời gian tính toán lâu hơn. Nếu giá trị tiêu chuẩn đã chọn nhỏ hơn 30, thì sẽ rất khó để tìm các mẫu đáp ứng tiêu chí khi có sự giao thoa cao giữa hai hình ảnh và gây ra lỗi thuật toán.

Khi sự khác biệt được tính toán bởi Eq. 8.6 lớn hơn giá trị tiêu chuẩn 30, sự khác biệt giữa hai tiêu bản được coi là rất nhỏ và hầu như không thể là một khu vực phù hợp. Do đó, chúng ta nên loại bỏ mẫu và tiếp tục tính toán tiếp theo. Khi sự khác biệt nhỏ hơn 30, được coi là mẫu rất có thể là một mẫu phù hợp. Tuy nhiên, 3 × 3 mẫu quá nhỏ để định vị chính xác. Vì vậy, theo thông tin khoảng cách được ghi lại giữa các mẫu 3 × 3 và 5 × 5 trong một nhóm mẫu, chúng ta có thể tìm hai mẫu 5 × 5 ở một vị trí có cùng

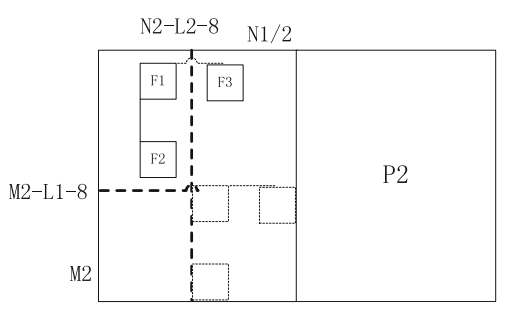
thông tin khoảng cách xung quanh mẫu 3x3 trong P 2. Sau đó, tính toán sự khác biệt giữa mẫu 5 × 5 tương ứng với mẫu tính năng nhóm. Hàm tổng của các chênh lệch được định nghĩa như bên dưới.



Tính toán tất cả F1 có sự khác biệt nhỏ hơn 30 thông qua Eq. 8.7 và lưu mọi kết quả. Bên cạnh đó, lưu các giá trị ngang và sắp xếp của phần trên bên trái các điểm pixel góc của mẫu Fl tại cùng một thời điểm. Cuối cùng, sử dụng Eq. 8.8 để có được tổng chênh lệch nhỏ nhất, các giá trị ngang và sắp xếp của góc trên bên trái của mẫu Fl tương ứng với tổng chênh lệch nhỏ nhất là tọa độ của các điểm phù hợp thu được.

Sfind( x, y) = MIN (S 5 × 5 ( x, y)) (8,8)

Thuật toán này thu được một cách chính xác mẫu phù hợp nhất bằng cách chọn lọc các mẫu hai lần. Đầu tiên đo mức độ liên quan của mẫu nhỏ 3 × 3 và lưu từng mẫu với độ tương quan cao. Sau đó tính giá trị liên quan của hai mẫu 5 × 5 tương ứng với các mẫu đã lưu và thu thập mẫu 5 × 5 có mức độ liên quan cao nhất. Điều này có nghĩa là phải lọc lại các mẫu thu được ở bước đầu tiên để có mẫu phù hợp nhất (Hình. 8.7 ).



Hình 6: Đối sánh ngang của các mẫu tính năng

1. Ghép hình ảnh và kết hợp hình ảnh. Sau khi kết thúc điểm khớp, sự chồng chéo đơn giản sẽ gây ra các đường viền rõ ràng trong ảnh là điều không mong muốn. Cần phải có một quá trình chuyển đổi mượt mà cho việc ghép ảnh để loại bỏ những điểm không đáng có như vậy. Thuật toán trong và ngoài được phân cấp có thể thu được hình ảnh liền mạch, nhưng trong thời kỳ hợp nhất hình ảnh, các vùng chồng chéo của hai hình ảnh được chồng lên nhau bằng trọng số tuyến tính và điều này chắc chắn làm cho các vùng chồng chéo bị mờ hơn so với hình ảnh gốc. Do đó, chúng tôi sử dụng tổng hợp Gauss để thay thế. Bằng cách thực hiện sự thay đổi của hệ số gradient từ 0 đến 1 tuân theo đặc tính phân phối của đường cong Gauss một cách xấp xỉ, và đạt được sự chuyển đổi nhanh chóng giữa hai hình ảnh. Khu vực chồng chéo của kết quả ghép ảnh rõ ràng hơn so với phương pháp tiếp cận “in-and-out” .

Code:

%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all**;**

clc**;**

A1**=**imread**(**'1.jpg'**);**

subplot**(**1**,** 2**,** 1**),** imshow**(**A1**)**

title**(**' Source Image A'**)**

B1**=**imread**(**'2.jpg'**);**

subplot**(**1**,** 2**,** 2**),** imshow**(**B1**)**

title**(**'Source Image B'**)**

**[**x**,**y**,**rgb**]=**size**(**A1**);**

% covert jpg to gray

A **=** rgb2gray**(**A1**);**

B **=** rgb2gray**(**B1**);**

%%%%%%%%%%%%%%%%%%%%%%%%%%

**[**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 **=** 50**;** y **=** 1**;**

zz **=** 0**;**

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**if** y **==** 0

zz **=** 1**;**

AA **=** A**(**1**:**high **-** 1**,** 1**:**3 **\*** wid **/** 4**);**

BB **=** B**(**1**:**high **-** 1**,** 1**:**wid **-** 1**);**

**else**

**if** y **>=** high **/** 2

zz **=** 2**;**

AA **=** A**(**1**:**high **-** y**,** 1**:**3 **\*** wid **/** 4**);**

BB **=** B**(**y**:**high **-** 1**,** x**:**wid **-** 1**);**

**else**

zz **=** 3**;**

yy **=** y**;**

xx**=**x**;**

AA **=** A**(**y**:**high **-** 1**,** 1**:**3 **\*** wid **/** 4**);**

BB **=** B**(**y**:**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**];** %\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

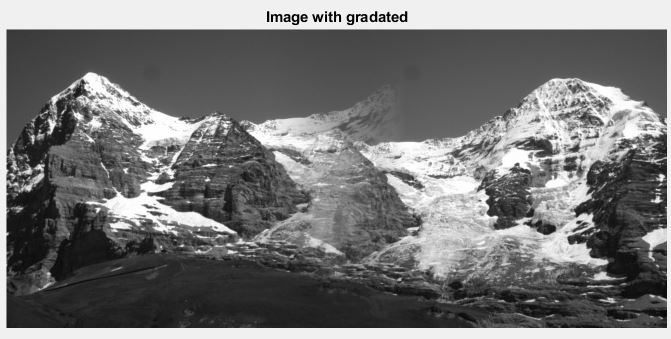
imwrite**(**D2**,** '6.bmp'**);**

figure**,** imshow**(**D2**)**

title**(**'Image with gradated'**);**

Kết Quả:



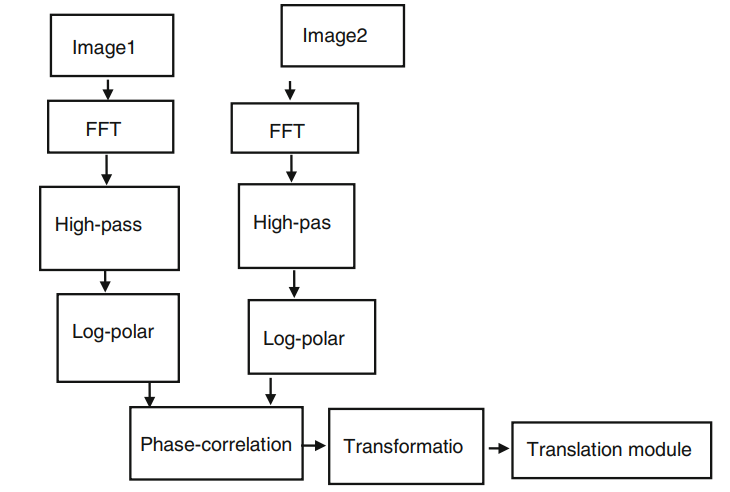
8.2.3 Ghép ảnh dựa trên FFT

Ảnh Toàn cảnh đề cập đến sự hình thành của chế độ xem đầy đủ, hình ảnh 360 ° có độ phân giải cao thông qua xử lý hình ảnh. Nó là một sự tái tạo tích hợp đối với tầm nhìn xung quanh và nó có thể hiển thị thông tin tích hợp tốt hơn về môi trường xung quanh.

Khâu hình ảnh dựa trên FFT, đầu tiên chuyển đổi hình ảnh sang miền tần số và tính toán lượng quay và độ lệch theo phổ công suất chéo pha của nó. Sau đó đặt lại tọa độ của hình ảnh và áp dụng chuyển động. Cuối cùng, các hình ảnh được ghép lại với nhau. Khi ghép ảnh toàn cảnh 360 °, cần chuyển đổi độ dài tiêu cự và hình chiếu trước khi tính toán với các giai đoạn. Hình 8.9 trình bày sơ đồ ghép ảnh dựa trên FFT.

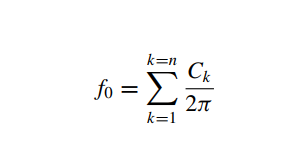
Phương pháp áp dụng để ghép ảnh toàn cảnh hình trụ có thể được chia thành 3 phần:

1. Xây dựng một hàm với tương quan pha của miền tần số. Hàm này sẽ thực hiện phép biến đổi 2-DFourier trên hai hình ảnh đầu vào và trả về các giá trị offset giữa hai hình ảnh liền kề.
2. Tính toán các giá trị tiêu cự của một bộ ảnh thực tế 360 ° và áp dụng phép chiếu hình trụ cho chuỗi ảnh.
3. Gọi lần lượt hàm trong phần 1 để ghép các hình ảnh sau khi chiếu và xử lý ánh sáng để tạo ra hình ảnh toàn cảnh hình trụ. được tạo ra.



Hình 7: Lưu đồ ghép ảnh dựa trên FFT

Tiêu cự f là một tham số đáng kể khi sử dụng công thức chiếu hình trụ để biến đổi phép chiếu. Chúng tôi đặt các bản dịch giữa hai hình ảnh liền kề trong chuỗi hình ảnh trước khi chiếu như C1, C 2,. . . , C n tương ứng, C k đại diện cho phép dịch ngang giữa hình ảnh k và hình ảnh k + 1. Giá trị ban đầu của tiêu cự có tên là f 0 có thể được tính toán thông qua Công thức:



Code:

**function** code8\_2\_3**()**

clc**;**

clear**;**

%bpham debug

**global** image1

**global** image2

**global** image11

**global** image22

image1 **=** **(**imread**(**'1.jpg'**));**

**[**h**,** w**,** r**]** **=** size**(**image1**);**

image2 **=** **(**imread**(**'2.jpg'**));**

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**);**

% bPHAM DEBUG

% aa = T0(:, :, :, 1);

% bb = T0(:, :, :, 2);

% aaa = T1(:, :, :, 1);

% bbb = T1(:, :, :, 2);

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]ˈ

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 valueˈ˄IˈJ˅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 valueˈ˄IˈJ˅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))))

Kết quả:



8.3 Ghép ảnh dựa trên điểm đặc trưng

Thay vì sử dụng các giá trị pixel của hình ảnh, ghép hình ảnh [ 2 ] dựa trên điểm tính năng [ 3 ] tính toán các đặc điểm như kết cấu, cạnh, đối tượng, v.v., từ các pixel, sau đó sử dụng các tính năng làm tiêu chuẩn và tìm kiếm đối sánh cho các vùng đặc trưng tương ứng của hình ảnh chồng chéo. Loại phương pháp tiếp cận này mạnh mẽ hơn. Có hai quy trình để ghép ảnh dựa trên các điểm đặc trưng: trích xuất tính năng và đăng ký tính năng. Đầu tiên, trích xuất các điểm, đường và vùng mà thang màu xám thay đổi rõ ràng thành tập hợp tính năng. Thứ hai, cố gắng chọn các tính năng được ghép nối bằng cách sử dụng các thuật toán đối sánh tính năng giữa hai tập hợp tính năng. Một loạt các phương pháp tiếp cận phân đoạn hình ảnh đã được áp dụng để trích xuất đặc trưng và phát hiện cạnh, chẳng hạn như bộ mô tả canny, bộ mô tả Laplace-gauss và lõi vùng đang phát triển (RSD). Các đặc điểm không gian được trích xuất bao gồm các cạnh đóng, các cạnh mở, các đường chéo và các đặc trưng khác.

8.3.1 Phát hiện điểm tính năng SIFT

Quá trình ghép ảnh dựa trên các điểm đặc trưng của SIFT bao gồm: thu nhận ảnh, trích xuất và đối sánh đặc điểm, đăng ký ảnh (tính toán H) và ghép ảnh cuối cùng.

1. Thu nhận hình ảnh. Thu nhận hình ảnh là điều kiện tiên quyết để ghép hình ảnh. Các bộ thu thập hình ảnh khác nhau có thể thu được các chuỗi hình ảnh đầu vào khác nhau và tạo ra các hiệu ứng ghép hình ảnh khác nhau. Hiện tại, có ba phương pháp khác nhau để thu được chuỗi hình ảnh: (1) x đặt máy ảnh vào chân máy và xoay nó để lấy dữ liệu hình ảnh; (2) cố định máy ảnh trên nền có thể di chuyển và dữ liệu hình ảnh thu được bằng cách di chuyển song song nó; (3) Cầm máy ảnh để chụp dữ liệu hình ảnh bằng điểm fi xed xoay hoặc di chuyển theo hướng vuông góc với trục quang học của máy ảnh. Quá trình này sử dụng các hình ảnh đã cho.
2. Trích xuất tính năng và sự tương quan. Trích xuất các điểm đặc trưng của SIFT từ chuỗi hình ảnh đầu vào. Thuật toán tính toán và trích xuất các điểm đặc trưng đồng thời trong miền không gian và miền tỷ lệ. Do đó, các điểm đặc trưng thu được có tỷ lệ bất biến, có thể trích xuất chính xác các điểm đặc trưng tồn tại trong chuỗi ảnh với tỷ lệ lớn và thay đổi góc. Khoảng cách Euclide được sử dụng để tính toán khoảng cách giữa hai bộ mô tả điểm đặc trưng SIFT.
3. Đăng ký hình ảnh với tính toán H. Đăng ký hình ảnh dựa trên các điểm đặc trưng có nghĩa là ma trận chuyển đổi giữa các chuỗi hình ảnh được xây dựng bằng các điểm phù hợp để hoàn thành việc ghép các hình ảnh toàn cảnh. Để cải thiện độ chính xác của đăng ký hình ảnh, thuật toán RANSAC [ 4 ] được sử dụng để tính toán và xác định lại ma trận biến đổi. Thuật toán H tự động tính toán ma trận biến đổi: tính điểm đặc trưng trong từng ảnh; phù hợp với các điểm tính năng; tính giá trị ban đầu của ma trận; sử dụng phép lặp để tái lập ma trận biến đổi H; kết hợp khởi động; lặp lại các bước cho đến khi ổn định số điểm tương ứng.
4. Hợp nhất hình ảnh. Theo ma trận biến đổi H của hai hình ảnh, các hình ảnh tương ứng có thể được chuyển đổi để xác định vùng chồng chéo của các hình ảnh và đăng ký các hình ảnh sẽ được hợp nhất vào một hình ảnh trống mới để tạo thành một sơ đồ mosaic. Một thuật toán làm mịn có trọng số nhanh chóng và đơn giản được sử dụng để giải quyết vấn đề về ghép ảnh.

Quy trình của thuật toán ghép ảnh dựa trên các điểm đặc trưng SIFT được thể hiện trong hình 8.11

Code:

**function** **[** imgout **]** **=** imMosaic**(** img1**,**img2**,**adjColor **)**

%[ imgout ] = imMosaic( img1,img2,adjColor )

% img1 and img2 can (both) be rgb or gray, double or uint8.

% If you have more than 2 images to do mosaic, call this function several

% times.

% If you set adjColor to 1, imMosaic will try to try to adjust the

% color(for rgb) or grayscale(for gray image) of img1 linearly, so the 2

% images can join more naturally.

% Yan Ke @ THUEE, 20110123, xjed09@gmail.com

% 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

figure**,**imshow**(**img1**)**

figure**,**imshow**(**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**)));** % ȡ���ܵ�ɫ��

**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

function [descriptors, locs] = sift(img)

% If you have the Image Processing Toolbox, you can uncomment the following

% lines to allow input of color images, which will be converted to grayscale.

if size(img,3)

img = rgb2gray(img);

end

[rows, cols] = size(img);

% Convert into PGM imagefile, readable by "keypoints" executable

f = fopen('tmp.pgm', 'w');

if f == -1

error('Could not create file tmp.pgm.');

end

fprintf(f, 'P5\n%d\n%d\n255\n', cols, rows);

fwrite(f, img', 'uint8');

fclose(f);

% Call keypoints executable

if isunix

command = './sift ';

else

command = 'siftWin32 ';

end

command = [command ' <tmp.pgm >tmp.key'];

system(sprintf(command));

% Open tmp.key and check its header

g = fopen('tmp.key', 'r');

if g == -1

error('Could not open file tmp.key.');

end

[header, count] = fscanf(g, '%d %d', [1 2]);

if count ~= 2

error('Invalid keypoint file beginning.');

end

num = header(1);

len = header(2);

if len ~= 128

error('Keypoint descriptor length invalid (should be 128).');

end

% Creates the two output matrices (use known size for efficiency)

locs = double(zeros(num, 4));

descriptors = double(zeros(num, 128));

% Parse tmp.key

for i = 1:num

[vector, count] = fscanf(g, '%f %f %f %f', [1 4]); %row col scale ori

if count ~= 4

error('Invalid keypoint file format');

end

locs(i, :) = vector(1, :);

[descrip, count] = fscanf(g, '%d', [1 len]);

if (count ~= 128)

error('Invalid keypoint file value.');

end

% Normalize each input vector to unit length

descrip = descrip / sqrt(sum(descrip.^2));

descriptors(i, :) = descrip(1, :);

end

%delete tmp.key

fclose(g);

end

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

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

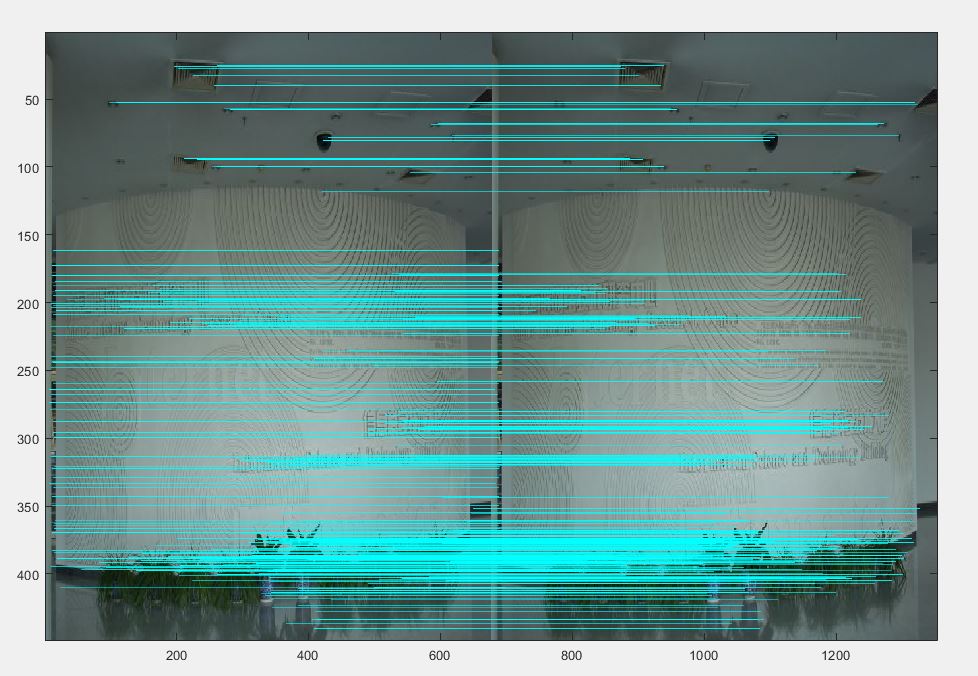
[~,idx] = max(inlrNum);

f = fLib{idx};

dist = funcDist(f,x,y);

inlierIdx = find(dist < thDist);

end



8.3.2 Khâu ghép ảnh dựa trên các điểm đặc trưng của Harris

Quá trình ghép ảnh dựa trên các điểm đặc trưng của Harris như sau.

1. Phát hiện các điểm đặc trưng Harris của hình ảnh;
2. Kết nối các điểm đặc trưng giữa hai hình ảnh và hoàn thành ghép ảnh;
3. Lọc tất cả các điểm phù hợp và các điểm phù hợp cần thiết để ghép ảnh;
4. Tính khoảng cách giữa các điểm đặc trưng của hai ảnh và làm mịn các phần chồng chéo của hình ảnh.

Code:

%clc;

clear all**;**

% read image

pic1 **=** imread**(**'a1.jpg'**);**

pic2 **=** imread**(**'a2.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];

end

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');

end

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

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

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');

end

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;

end

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

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

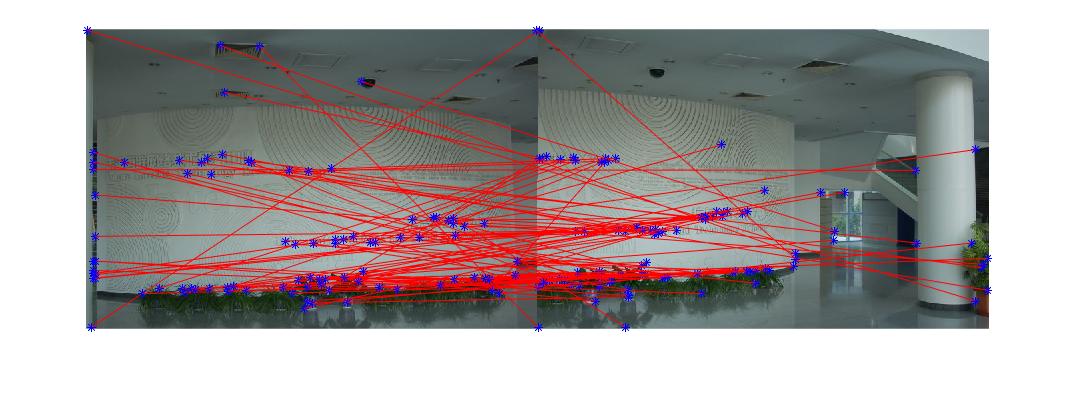
right = k; %right boundary

%%obtain image

im = pic(ceil:bottom, left:right, :);

end



8.3.3 Tự động sắp xếp cho chuỗi hình ảnh

Để ghép ảnh, chuỗi ảnh đầu vào [ 5 ] được đặt hàng theo nội dung cảnh thực tế, nghĩa là mỗi hai hình ảnh liền kề phải có các phần chồng lên nhau, để có thể ghép hình ảnh toàn cảnh chính xác. Tuy nhiên, trong quá trình chụp ảnh và lưu trữ hoặc nhập liệu, chuỗi ảnh có thể gây nhầm lẫn và không thể ghép trực tiếp. Để thực hiện tự động sắp xếp trình tự hình ảnh, có 3 vấn đề cần giải quyết trước hết:

1. Xác định xem có vùng chồng chéo giữa hai hình ảnh hay không, nghĩa là liệu hai hình ảnh có liên quan với nhau không;
2. Xác định ảnh đầu và ảnh đuôi của dãy ảnh;
3. Xác định mối quan hệ giữa vị trí bên trái và bên phải của hai vùng trùng hình ảnh.

Trong phần này, chúng tôi sử dụng phương pháp tương quan pha để sắp xếp chuỗi ảnh. Nguyên tắc của phương pháp tương quan pha như sau:

(Thêm hình nha anh)

Theo đặc điểm dịch chuyển của phép biến đổi Fourier, đây là:

Thêm Công thức

Phổ công suất tương hỗ chuẩn hóa của nó được biểu diễn như sau:

Thêm Công thức

Pha của mật độ quang phổ công suất chéo bằng độ lệch pha của hai hình ảnh. Mật độ phổ công suất chéo chuẩn hóa được vận hành để có được hàm xung thông qua phép biến đổi Fourier nghịch đảo:

Hàm này nhận giá trị lớn nhất ở độ dịch chuyển tương đối ( x, y) ( điểm trùng nhau) của hai hình ảnh, bất kỳ nơi nào khác gần 0. độ dịch chuyển tương đối ( x, y) được xác định bằng cách loại bỏ vị trí của điểm nhìn lén trong công thức ( 4.4 ). Trong trường hợp chỉ dịch giữa các ảnh, độ lớn của đỉnh của hàm xung tạo ra mối tương quan giữa hai ảnh và nhận một giá trị trong một khoảng [0, 1]. Vùng chồng lấn giữa hai ảnh càng lớn thì giá trị càng lớn. Nếu hai hình ảnh có cùng nội dung giá trị là 1 và giá trị là 0 khi nó hoàn toàn khác nhau. Nếu vẫn có phối cảnh, nhiễu hoặc mục tiêu chuyển động giữa hai hình ảnh, năng lượng của hàm xung sẽ được phân phối từ một đỉnh duy nhất đến các đỉnh nhỏ khác, nhưng vị trí đỉnh tối đa của nó có độ bền nhất định.

Theo nguyên tắc của phương pháp tương quan pha, thuật toán sắp xếp tự động như sau:

1. Xác định hình ảnh đầu và đuôi (hình ảnh ngoài cùng bên trái và ngoài cùng bên phải) và liền kề hình ảnh. Đối với một chuỗi hình ảnh nhất định có n hình ảnh, bất kỳ hình ảnh nào cũng có thể được tính toán bởi phần còn lại N - 1 hình ảnh để lấy N - 1 mức độ tương quan. Vì hình ảnh sẽ tiếp giáp với tối đa hai hình ảnh (một hình ảnh trung gian), nên ít nhất nó sẽ tiếp giáp với một trong các hình ảnh (hình ảnh đầu và đuôi). Do đó, nếu hai giá trị lớn nhất đầu tiên được chọn từ N - 1 tương quan tính từ ảnh thì ảnh sẽ chồng lên 2 hoặc 1 trong 2 tương quan. Hoạt động ở bên trái N - 1 hình ảnh, 2 N mức độ tương quan lớn nhất thu được. Đối với hình ảnh đầu và hình ảnh đuôi, hai mức độ tương quan tương ứng của chúng sẽ có một mức độ không đủ điều kiện. Rõ ràng, độ tương quan tương ứng của ảnh đầu và ảnh đuôi nhỏ hơn các độ khác. Khi xác định mức độ tương quan nhỏ nhất của 2 N độ, hình ảnh đầu và đuôi thu được tương ứng. Cuối cùng, hình ảnh đầu và đuôi khác với các hình ảnh bên cạnh.
2. Xác định mối quan hệ giữa vị trí bên trái và bên phải của hai hình kề nhau. Trong phương pháp thuật toán tương quan pha, kết quả đo cho thấy δ hàm xung với các đỉnh tương quan rất sắc nét khi hai hình ảnh thực sự có liên quan. Các tham số dịch ngang của hai ảnh có thể được tính bằng các điểm pixel tương ứng của đỉnh. Khi tham số dịch ngang x lớn hơn một nửa chiều rộng hình ảnh, bạn có thể trừ nó khỏi chiều rộng hình ảnh và sau đó lấy giá trị âm. Nếu dịch ngang giữa hình ảnh A và B là tiêu cực, Hình ảnh A ở bên trái của hình ảnh B, và ngược lại, hình ảnh A ở bên phải của hình ảnh B.

Code:

clear all**;**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

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

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;

end

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

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

end

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))))

end



8.3.4 Đăng ký điểm Harris dựa trên thuật toán RANSAC

Đăng ký điểm Harris dựa trên Đồng thuận mẫu ngẫu nhiên (RANSAC) là một loại phương pháp đối sánh dựa trên các tính năng. Trước tiên, việc phát hiện điểm Harris được thực hiện và sau đó thực hiện đối sánh thô theo đặc điểm cục bộ của các điểm được trích xuất để tìm ra sự tương ứng giữa các tập hợp điểm được so khớp. Sau khi so sánh thô, hầu hết các cặp điểm khớp sai được loại bỏ, nhưng vẫn còn nhiều điểm thiếu so với yêu cầu. Các cặp điểm này có sai số lớn trong mối quan hệ hình học chủ yếu vẫn là do sự giống nhau về thông tin thang xám. Những điểm này được gọi là các cặp đối sánh giả. Thuật toán RANSAC được sử dụng để loại bỏ các cặp đối sánh giả. RANSAC là một loại thuật toán lặp để ước tính các tham số mô hình toán học. Ý tưởng chính là tính toán các thông số để làm cho phần lớn các mẫu (điểm đặc trưng) có thể thiết lập mô hình toán học. Tại lần lặp lại, số lượng mẫu tối thiểu được sử dụng để lấy mẫu mô hình và tính toán các thông số và số lượng mẫu chuẩn bị cho mô hình được đếm. Và các tham số mẫu tối đa được coi là giá trị của mô hình cuối cùng. Điểm mẫu phù hợp với mô hình được gọi là phần trong và điểm mẫu không phù hợp với mô hình được gọi là điểm ngoài hoặc điểm hoang dã.

Các ý tưởng cơ bản của RANSAC như sau:

Hãy xem xét một mô hình yêu cầu một bộ lấy mẫu tối thiểu với n mẫu ( n cho số lượng mẫu tối thiểu cần thiết để khởi tạo các thông số mô hình) và một bộ mẫu P, số mẫu của tập P # (P)> n. Tập hợp con S với n các mẫu được trích xuất ngẫu nhiên từ P được sử dụng để khởi tạo mô hình M:

Các mẫu trong bộ bổ sung SC = P / S có lỗi nhỏ hơn ngưỡng đã đặt t, cùng với bộ S cấu tạo S ∗. S ∗ được coi là bộ nội tại thiết lập và xây dựng S Bộ đồng thuận.

Nếu #( S ∗) ∗ N, các tham số phù hợp được coi là thu được và phương pháp Bình phương nhỏ nhất, v.v. được sử dụng để ước tính mô hình mới M ∗ trên bộ nội quy S ; hoặc lấy lại mẫu mới S và lặp lại.

Sau một số lần lấy mẫu nhất định, nếu không tìm thấy tập hợp nhất quán nào thì thuật toán không thành công, nếu không thì tập hợp đồng thuận thập phân thu được sau khi lấy mẫu và thuật toán kết thúc. Mã được hiển thị trong CHƯƠNG TRÌNH 8.7.

Code:

%{

How to run this test:

clear all;

clc;

im=imread('lena256.pgm');

code8\_3\_4(im)

%}

function points = code8\_3\_4(im)

% clear all;

% clc;

% im=imread('lena256.pgm');

% 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=imread('lena256.pgm');

im=im2double(im);

% 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];

%bpham add

figure, imshow(im),title('Harris Feature Points');

hold on

plot(points(:,1),points(:,2),'r\*');

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 --------------------------------------

% 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);

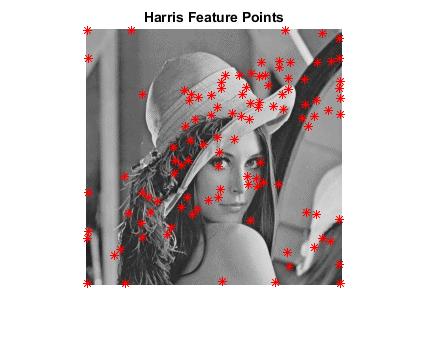
% error = dist(a2\_model', a2);

% end

% function d = dist(one, two)

% d = sqrt(sum((one - two).^2));

% end



8.4 Ghép ảnh toàn cảnh

Ghép ảnh toàn cảnh nhằm mục đích ghép nối liền mạch trên chuỗi ảnh được chụp từ cùng một cảnh, góc nhìn khác nhau, độ dài tiêu cự khác nhau, từ cùng một trung tâm quang học với một phần chồng lên nhau. Điều này có nghĩa là thuật toán đăng ký hình ảnh được sử dụng để tính toán các thông số chuyển động giữa mỗi khung hình và sau đó tổng hợp một hình ảnh góc rộng tĩnh lớn. Hơn nữa, hình ảnh đường khâu yêu cầu phải gần như cảnh thật, không có đường may rõ ràng. Theo các quan điểm khác nhau, ghép ảnh có thể được chia thành thuật toán dựa trên một góc nhìn và thuật toán dựa trên nhiều góc nhìn. Để có được chuỗi hình ảnh điểm nhìn đơn lẻ, một máy ảnh được bố trí ở một vị trí và xoay nó xung quanh; hoặc, đặt máy ảnh theo hình tròn, trục quang học của máy ảnh nằm trên cùng một mặt phẳng và giao với một điểm và video được thu theo thời gian thực ở các vị trí khác nhau và chụp cùng một lúc. Thuật toán ghép ảnh dựa trên nhiều điểm nhìn thường được sử dụng để ghép ảnh toàn cảnh theo dải.

Phần này giới thiệu thuật toán ghép ảnh dựa trên phép biến đổi phép chiếu ảnh mà không có đối tượng hoạt động.

Thông tin hình ảnh rời rạc chỉ có thể thể hiện thông tin trên một phần của môi trường trực quan. Chế độ xem toàn cảnh dựa trên kết xuất hình ảnh là hiển thị thông tin hình ảnh rời rạc trong một hình ảnh hoàn toàn. Xây dựng một môi trường đồ họa hoàn chỉnh để có hiệu ứng hình ảnh 3D tốt hơn.

(1) Định vị hình ảnh, Tự động tìm các vị trí chồng chéo cho hình ảnh. Đề xuất có hai vùng A và B. B chứa một khu vực A 2. A 2 và A là cùng một mô-đun, vị trí của A 2 trong B là để giải quyết. Thuật toán điển hình là tìm kiếm từ góc dưới bên trái của B, trong đó mỗi mảnh được so sánh với a có cùng diện tích C và A, và giá trị của hàm đánh giá, diện tích nhỏ nhất là A 2.

(2) Ghép hình ảnh

Sau khi định vị hình ảnh, nếu ghép đơn giản hai hình ảnh sẽ có một đường nối rõ ràng do sự chênh lệch về độ sáng. Phương pháp thử màu có thể được sử dụng để điều hòa độ sáng của các hình ảnh liền kề và tạo ra hình ảnh tổng hợp liền mạch.

(3) Thực hiện phép chiếu hình trụ

Được chụp tại cùng một điểm với camera quay, các hình ảnh toàn cảnh hình trụ không nằm trong cùng một hệ tọa độ. Có một góc nhất định trên mặt chiếu. Để tạo ra một hình ảnh toàn cảnh, chúng ta phải biến đổi những hình ảnh này thành một hệ tọa độ hình trụ duy nhất và sử dụng công nghệ ghép hình ảnh để loại bỏ sự chồng chéo của mỗi hai hình ảnh. Bằng cách này, một bức tranh toàn cảnh hình trụ hoàn chỉnh sẽ thu được.

Như thể hiện trong Hình. 8.13 , đó là sơ đồ hình chiếu dương của bề mặt hình trụ. I là một khung được trích xuất từ video, P là bất kỳ điểm nào trên hình ảnh đã chụp, Q là điểm mà P bản đồ đến tọa độ hình trụ.

Thêm hình nha anh ( 8.13)

Giả sử rằng W và H là tầm nhìn của hình ảnh I tương ứng, f là bán kính của

hình trụ, sao cho P Tọa độ của trong hệ tọa độ 3D được biểu diễn dưới dạng ( x - W / 2, y - H / 2, f). Sử dụng sự kết hợp của phương trình tham số và phương trình hình trụ,

giả sử Q tọa độ của là x ( ′, y ′, z ′) và P cũng như Q nằm trên cùng một đường thẳng, thỏa mãn phương trình tham số:

Thêm công thuc nha anh

Với t là tham số, cùng với phương trình bề mặt hình trụ: Thêm công thuc nha anh

Như vậy, chúng ta có thể nhận được tọa độ của điểm Q vì tọa độ của điểm Q là ba chiều, chúng tôi chuyển đổi nó thành hai chiều để nhận được: Thêm công thuc nha anh

Do đó, chúng ta có thể nhận được tọa độ của Q điểm vì tọa độ của Q điểm là ba chiều, chúng tôi chuyển đổi nó thành hai chiều để nhận được:

Sau khi chiếu ảnh xuống mặt phẳng trụ ta thu được các ảnh của cùng một hệ trục tọa độ. Sau đó, bằng cách tìm kiếm sự chuyển đổi giữa các hình ảnh liền kề, các hình ảnh chuỗi được nối với nhau để tạo thành một hình ảnh toàn cảnh hình trụ trong cùng một cảnh.

Các bước sử dụng phương pháp IBR để ghép hình ảnh toàn cảnh video như sau:

(1) Trích xuất các khung chính của video và sử dụng hình ảnh để thể hiện thông tin trong video.

(2) Tìm ra vùng chồng chéo của hình ảnh, điều này có nghĩa là trích xuất vị trí của đối tượng địa lý.

(3) Đăng ký hình ảnh, điểm tính năng phù hợp. Sử dụng thuật toán đối sánh để loại bỏ các cặp điểm sai và các cặp điểm góc di động. Chức năng chuyển đổi tọa độ có được bằng cách tính toán ma trận chuyển đổi giữa hình ảnh dữ liệu và hình ảnh được so khớp. Cuối cùng, chức năng biến đổi tọa độ được sử dụng để chuyển đổi hình ảnh sang hệ tọa độ dữ liệu và nhận ra việc đăng ký hình ảnh được khớp với hình ảnh dữ liệu trong cùng một hệ tọa độ.

(4) Bước cuối cùng là ghép ảnh, liên quan đến việc hợp nhất hai ảnh và loại bỏ các đường nối.

Code:

%xyloObj = VideoReader('test.avi'); %read video -> Can not find test.avi

%replace: rhinos.avi

xyloObj **=** VideoReader**(**'rhinos.avi'**);**

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)



Ghép ảnh toàn cảnh dựa trên phương pháp IBR sử dụng các khung hình chính, có mã là chương trình 8.9 và các chức năng liên quan được mô tả trong chương trình 8.3.

Code:

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');

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

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;

end

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

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

end

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))))

end



