**Báo cáo chương 3**

1. **Homographies**

* Homographies là một phép biến đổi ánh xạ 2D ánh xạ các điểm trong một mặt phẳng này tới một mặt phẳng khác
* Về bản chất một phép biến đổi đồng nhất H ánh xạ các điểm 2D theo

x’ = Hx với H là một ma trận 3 \* 3

* Toạ độ đồng nhất là một biểu diễn hữu ích cho các điểm trong mặt phẳng hình ảnh
* Một trường hợp quan trọng trong phép biến đổi này là Affine transformation. Được sử dụng cho ví dụ làm cong ảnh

A picture containing text, clock

Description automatically generated

* Similarity transformation là một biến đổi 2D cứng nhắc cũng bao gồm các phép thay đổi tỉ lệ. s chỉ tỉ lệ, R là mộ độ xoay góc teta và t là một phép tịnh tiến, với s = 1 là một rigid transformation ( phép chuyển đổi cứng nhắc). Biến đổi tương đồng được ứng dụng trong đăng kí hình ảnh

A picture containing text, clock

Description automatically generated

* 1. The direct linear transformation algorithm
* Phép đồng nhất có thể được tính toán trực tiếp từ các điểm tương ứng trong hai bức ảnh hoặc là mặt phẳng. Cần 4 điểm tương ứng ddeer có thể tính được H
* DLT là một thuật toán cho việc tính H được cho bởi 4 điểm tương ứng hoặc hơn. Tính H sử dụng phương pháp bình phương tối thiểu bằng cách sử dụng singular value decomposition (SVD).
* Thuật toán này sẽ chuẩn hoá các điểm để chúng có trung bình bằng 0 và độ lệch chuẩn. Một ma trận A được tính bằng các điểm tương ứng. giải một ma trận với linalg.svd(A) và trả về một U, S, V, t reshape V[8] thành một ma trận 3 \* 3 và được de-conditioned, chuẩn hoá rồi trả về H
  1. Affine Transformations
* Có sáu bậc tự do nên cần 3 điểm tương ứng để ước tính H. Nó cx có thể được tính sử dụng phương pháp trên với h7, h8 bằng 0.

1. **Warping images**

* Áp dụng H affine transformation trên một mảng các hình ảnh thì được gọi là warping hoặc là affine warping. Nó được thực hiện dễ dàng với ndimage.affine\_transform(im, A, b, size) biến đổi bản vá hình ảnh im, A là một phép chuyển đổi tuyến tính, b là một tranformation vector, size có thể là option như là kích thước của ảnh đầu ra. Mặc định là cùng size với ảnh ban đầu.

Graphical user interface

Description automatically generated

**Hình 1 Affine Transformations**

* **Image to Image**

Sử dụng hàm ginput() trong pylab để lấy ra toạ độ các điểm một cách thủ công

A row of houses

Description automatically generated with low confidence

A row of houses

Description automatically generated with medium confidence

* Tạo các anpha map cho mỗi tam giác và tổng hợp chúng lại, làm cho các góc bây giờ match với nhau (fit hoàn toàn các góc và cạnh)

A picture containing text, house, outdoor, white

Description automatically generated

* **Piecewise affine warping**

Chart

Description automatically generated

Hình 1 Ví dụ về tam giác hoá các điểm ngẫu nhiên trong toạ độ 2D

* Thuật toán Delaunay triangulation chọn các hình tam giác sao cho góc nhỏ nhất trong tất cả các góc của các tam giác trong triangulations là lớn nhất. Nó sẽ trả ra một list các triangles

Graphical user interface

Description automatically generated

* **Registering images**
* Đăng kí hình ảnh ( Registering images) là quá trình chuyển hình ảnh để chúng được căn chỉnh trong một khung toạ độ chung. Đăng kí có thể là cứng nhắc hoặc không cứng nhắc và là một bước quan trọng để có thể so sánh ảnh và phân tích tinh vi hơn
* Sử dụng phương pháp bình phương tối thiểu để giải quyết bài toán

Text, table

Description automatically generated with medium confidence

Sau đó sẽ trả ra một ma trận R, và vector tịnh tiến t

A picture containing text, clock

Description automatically generated

Sau đó áp dụng phép affine transformation trên từng kênh màu.

* Thuận toán để thực hiện Registering images(Sử dụng openCV do không lấy được dữ liệu theo thuật toán trong sách:
* Chuyển cả hai ảnh sang ảnh xám
* Matching các thuộc tính từ ảnh được căn chỉnh so với ảnh tham chiếu và lưu lại toạ độ của các Keypoint tương ứng. Keypoints chỉ đơn giản là chọn một vài điểm để tính toán chuyển đổi ( từ các điểm nổi bật) và bộ mô tả (descriptors) là các histogram của gradients của các điểm xunh quanh một điểm keypoints. Ở đây sử dụng thuật toán ORB
* Match các keypoint giữa hai ảnh với nhau
* Tìm một phép biến đổi hình thái phù hợp

Ảnh aligned:

A close-up of a book

Description automatically generated with low confidence

Ảnh sau khi thực hiện registing:

A close-up of a book

Description automatically generated with medium confidence

1. **Tạo ảnh toàn cảnh (Creating Panoramas)**

Hai hay nhiều ảnh được chụp cùng một vị trí thì có quan hệ đồng nhất. Thường được sử dụng để tạo ảnh toàn cảnh, ghép nhiều ảnh thành một ảnh lớn hơn

* 1. **RANSAC ( Random sample consensus)**

Đây là một phương pháp hi vọng có thể fit toàn bộ dữ liệu khi train trong khi dữ liệu có thể có một số ngoại lai mà phương pháp binh phương tối thiểu có thể sẽ không làm được. Sẽ được sử dụng để ước tính đồng nhất (homography estimation)

* 1. **Robust homography estimation**
* Tìm tự động một phép đồng nhất cho ảnh toàn cảnh sử dụng các điểm tương ứng. SIFT thì phát hiện descriptors rất là mạnh mẽ và matches lỗi thì ít hơn Harris với một patch các tương quan. Nhưng vẫn chưa đủ hoàn toàn tốt
* Class RansacModel bao gồm một method fit() nhận 4 corespondences (x, y) và (x’, y’) được chọn bởi ransac.py (4 cái đầu tiên trong data) và fit một phép tương đồng (4 điểm là số tối thiểu để tính đồng nhất). Hàm get\_error() áp dụng phép tương đồng và trả về căn bậc hai tổng khoảng cách cho mỗi cặp correspondences để ransac có thể chọn các điểm nào để giữ là điểm ngoại lai và điểm bất thường. Điều này được thực hiện với một ngưỡng trên khoảng cách.
* Hàm H\_from\_ransac(fp, tp, model, maxiter=1000, match\_theshold=10) cung cấp một ngưỡng và số điểm tối thiểu mong muốn. Tham số maxiter cho số lần lặp tối đa, nếu ít quá thì có thể có một giải pháp tồi tệ, nếu nhiều quá thì mất thời gian, và trả ra cùng các điểm bất thường.
  1. **Stitching images together**

1. ### Panorama Ảnh toàn cảnh
2. def panorama(H, fromim, toim, padding=2400, delta=2400):
3. """ Tạo một ảnh toàn cảnh bằng cách kết hợp hai ảnh
4. sử dụng một phép đồng nhất H ( đã được ước tính bằng
5. cách sử dụng RANSAC) Kết quả là một bức ảnh với chiều
6. cao bằng chiều cao của toim. Padding chỉ định số pixel
7. lấp đầy và delta dịch chuyển bổ sung """
8. # Kiểm tra ảnh có là ảnh xám hay ảnh màu
9. is\_color = len(fromim.shape) == 3
10. # Phép đồng nhất cho phép biến đổi hình  học
11. def transf(p):
12. p2 = dot(H, [p[0], p[1], 1])
13. return (p2[0] / p2[2], p2[1] / p2[2])
14. # Nếu fromim là bên phải
15. if H[1, 2] < 0:
16. print('warp - right')
17. # Biến đổi fromim
18. if is\_color:
19. # padding tới ảnh gốc với 0 về bên phải
20. toim\_t = hstack((toim, zeros((toim.shape[0], padding, 3))))
21. fromim\_t =  zeros((toim.shape[0], toim.shape[1] + padding, toim.shape[2]))
22. for col in range(3):
23. fromim\_t[:, :, col] = ndimage.geometric\_transform(fromim[:, :, col], transf, (toim.shape[0], toim.shape[1] + padding))
24. else:
25. # padding ảnh đích với không về phía bên phải
26. toim\_t = hstack((toim, zeros((toim.shape[0], padding))))
27. fromim\_t = ndimage.geometric\_transform(fromim, transf, ((toim.shape[0], toim.shape[1] + padding)))
28. else: # Bên trái
29. print('warp - left')
30. # Thêm một dịch chuyển để bù cho padding ở bên trái
31. H\_delta = array([[1, 0, 0], [0, 1, -delta], [0, 0, 1]])
32. H = dot(H, H\_delta)
33. # Chuyển đổi fromim
34. if is\_color:
35. # thêm vào ảnh đích với không tới bên trái
36. toim\_t = hstack((zeros((toim.shape[0], padding, 3))), toim)
37. fromim\_t = zeros((toim.shape[0], toim.shape[1] + padding, 3))
38. for col in range(3):
39. fromim\_t[:, :, col] = ndimage.geometric\_transform(fromim\_t, transf, (toim.shape[0], toim.shape[1]) + padding)
40. else:
41. # pad the destination image with zeros to the left
42. toim\_t = hstack((zeros((toim.shape[0], padding)), toim))
43. fromim\_t = ndimage.geometric\_transform(fromim,
44. transf, (toim.shape[0], toim.shape[1] + padding))
45. # Kết hợp và trả lại kết quả ( đặt fromim trên toim)
46. if is\_color:
47. # Lấy pixel không phải màu đen:
48. alpha = ((fromim\_t[:, :, 0] \* fromim\_t[:, :, 1] \* fromim\_t[:, :, 2]) > 0)
49. for col in range(3):
50. toim\_t[:, :, col] = fromim\_t[:, :, col]\*alpha + toim\_t[:, :, col]\*(1 - alpha)
51. else:
52. alpha = (fromim\_t > 0)
53. toim\_t = fromim\_t\*alpha + toim\_t\*(1 - alpha)
55. return toim\_t

* Sử dụng hàm geometric\_transform() để ánh xạ từ pixel tới pixel. Hàm transf() thực hiện nhân với H và chuẩn hoá các toạ độ đồng nhất. Check giá trị dịch chuyển trong H chúng ta sẽ quyết định xem bức ảnh thêm vào bên trái hay bên phải
* Khi ảnh được thêm vào bên trái, các toạ độ các điểm trong ảnh đích phải thay đổi để một bản dịch được thêm vào một bản dịch đồng nhất. Ở đây sử dụng pixel 0 bằng việc sử dụn anpha map để kết hợp hay hình ảnh ( Pixel nào có giá trị dương thì lấy ảnh fromim, với pixel bằng không thì lấy giá trị pixel của ảnh toim

**Dưới đây là cách ứng dụng trong OpenCV**

* Ước tính homography matrix với RANSAC, từ một tập điểm tương ứng(matches) tới tập điểm đích, thuật toán ransac sẽ chọn ra ngẫu nhiên 4 điểm một, và chọn H sao cho tổng khoảng cách giữa điểm target với điểm input \* H là nhỏ nhất
* Sau đó ta sẽ sử dụng H này để thực hiện ghép ảnh, tạo ảnh toàn cục. Kết quả ở dưới. Code hàm ImageStitching.py

A picture containing text, sky, outdoor, screenshot

Description automatically generated

Hình 2 Hai hình ảnh gốc ban đầu

A picture containing text, screenshot

Description automatically generated

Hình 3 200 Matches using SIFT

A picture containing text, outdoor

Description automatically generated

Figure 1 Kết quả Stitching Image