**TỪ TRANG 63 ĐẾN TRANG 68**

là một phép biến đổi 2D bao gồm sự thay đổi về tỷ lệ. Một con số vô hướng s xác định tỷ lệ, R là góc quay và t = [tx, ty] là một phép biến đổi dịch chuyển đơn thuần. Với s = 1 khoảng cách được bảo toàn và sau đó là một biến đổi cứng. Các biến đổi tương tự được sử dụng trong các ví dụ về đồng nhất tọa độ hình ảnh.

Hãy cùng xem các thuật toán để ước tính các phép đo đồng nhất và sau đó đi vào các ví dụ về việc sử dụng các phép biến đổi affine để nắn hình ảnh, các phép biến đổi tương tự để hiển thị và cuối cùng là các phép biến đổi ánh xạ để tạo ảnh toàn cảnh.

**3.1.1. Thuật toán biến đổi tuyến tính trực tiếp**

Tọa độ đồng nhất có thể được tính trực tiếp từ các điểm tương ứng trong hai hình ảnh (hoặc mặt phẳng). Như đã đề cập trước đó, một biến đổi ánh xạ đầy đủ có tám bậc tự do. Mỗi điểm tương ứng cho hai phương trình, mỗi phương trình cho tọa độ x và y, và do đó cần có bốn điểm tương ứng để tính H.

Phép biến đổi tuyến tính trực tiếp (The direct linear transformation-DLT) là một thuật toán để tính toán H cho trước bốn hoặc nhiều điểm tương ứng hơn. Bằng cách viết lại phương trình ánh xạ dùng ma trận H cho một số điểm tương ứng, ta được một phương trình như sau:

(3.7)

hoặc Ah = 0 trong đó A là ma trận có số hàng gấp đôi số điểm tương ứng. Bằng cách xếp chồng tất cả các điểm tương ứng, dùng phương pháp bình phương tối thiểu để tính H bằng cách sử dụng Singular Value Decomposition (SVD). Thêm hàm dưới đây vào homography.py.

def H\_from\_points(fp, tp):  
 *""" Find homography H, such that fp is mapped to tp  
 using the linear DLT method. Points are conditioned  
 automatically. """* if fp.shape != tp.shape:  
 raise RuntimeError('number of points do not match')  
  
 # condition points (important for numerical reasons)  
 # - from points -  
 m = mean(fp[:2], axis=1)  
 maxstd = max(std(fp[:2], axis=1)) + 1e-9  
 C1 = diag([1/maxstd, 1/maxstd, 1])  
 C1[0][2] = -m[0]/maxstd  
 C1[1][2] = -m[1]/maxstd  
 fp = dot(C1, fp)  
  
 # - to points -  
 m = mean(tp[:2], axis=1)  
 maxstd = max(std(tp[:2], axis=1)) + 1e-9  
 C2 = diag([1/maxstd, 1/maxstd, 1])  
 C2[0][2] = -m[0]/maxstd  
 C2[1][2] = -m[1]/maxstd  
 tp = dot(C2, tp)  
  
 # create matrix for linear method, 2 rows for each correspondence pair  
 nbr\_correspondences = fp.shape[1]  
 A = zeros((2\*nbr\_correspondences, 9))  
 for i in range(nbr\_correspondences):  
 A[2\*i] = [-fp[0][i], -fp[1][i], -1, 0, 0, 0,  
 tp[0][i]\*fp[0][i], tp[0][i]\*fp[1][i], tp[0][i]]  
 A[2\*i+1] = [0, 0, 0, -fp[0][i], -fp[1][i], -1,  
 tp[1][i]\*fp[0][i], tp[1][i]\*fp[1][i], tp[1][i]]  
  
 U, S, V = linalg.svd(A)  
 H = V[8].reshape((3, 3))  
  
 # decondition  
 H = dot(linalg.inv(C2), dot(H, C1))  
  
 # normalize and return  
 return H / H[2, 2]

Ta cần kiểm tra số lượng điểm bằng nhau. Điều này hữu ích để viết chương trình nhưng sẽ chỉ sử dụng các điểm dị biệt trong rất ít trường hợp trong cuốn sách này để làm cho các mẫu chương trình đơn giản hơn và dễ theo dõi hơn. Có thể đọc thêm về các loại điểm dị biệt tại <http://docs.python.org/library/exceptions.html> và cách sử dụng tại <http://docs.python.org/tutorial/errors.html>.

Các điểm được bắt theo điều kiện bằng cách chuẩn hóa sao cho chúng có độ lệch chuẩn trung bình và sai lệch bằng 0. Điều này rất quan trọng vì tính ổn định của thuật toán phụ thuộc vào biểu diễn tọa độ. Sau đó, ma trận A được tạo bằng cách sử dụng các điểm tương ứng. Dùng phương pháp bình phương tối thiểu, hàng cuối cùng trong ma trận V của SVD được định dạng lại để tạo H. Ma trận này sau đó được bỏ điều kiện và chuẩn hóa trước khi được trả về.

**3.1.2. Biến đổi Affine**

Một phép biến đổi affine có sáu bậc tự do và do đó cần có ba điều kiện để ước tính H. Biến đổi affine có thể được ước tính bằng thuật toán DLT bằng cách đặt hai phần từ cuối bằng 0, = = 0. Ở đây sẽ sử dụng một cách tiếp cận khác, được mô tả chi tiết trong [13] (trang 130). Thêm hàm sau vào homography.py, tính toán ma trận biến đổi affine từ các điểm tương ứng.

def Haffine\_from\_points(fp, tp):  
 *""" Find H, affine transformation, such that  
 tp is affine transf of fp. """* if fp.shape != tp.shape:  
 raise RuntimeError('number of points do not match')  
  
 # condition points  
 # -- from points --  
 m = mean(fp[:2], axis=1)  
 maxstd = max(std(fp[:2], axis=1)) + 1e-9  
 C1 = diag([1/maxstd, 1/maxstd, 1])  
 C1[0][2] = -m[0]/maxstd  
 C1[1][2] = -m[1]/maxstd  
 fp\_cond = dot(C1, fp)  
  
 # -- to points --  
 m = mean(tp[:2], axis=1)  
 C2 = C1.copy() # must use same scaling for both point sets  
 C2[0][2] = -m[0]/maxstd  
 C2[1][2] = -m[1]/maxstd  
 tp\_cond = dot(C2, tp)  
  
 # conditioned points have mean zero, so translation is zero  
 A = concatenate((fp\_cond[:2], tp\_cond[:2]), axis=0)  
 U, S, V = linalg.svd(A.T)  
  
 # create B and C matrices as Hartley-Zisserman (2nd ed) p 130.  
 tmp = V[:2].T  
 B = tmp[:2]  
 C = tmp[2:4]  
  
 tmp2 = concatenate((dot(C, linalg.pinv(B)), zeros((2, 1))), axis=1)  
 H = vstack((tmp2, [0, 0, 1]))  
  
 # decondition  
 H = dot(linalg.inv(C2), dot(H, C1))  
  
 return H / H[2, 2]

Một lần nữa, các điểm được bắt theo điều kiện và khử điều kiện như trong thuật toán DLT. Hãy cùng xem những biến đổi affine này có thể làm gì với hình ảnh trong phần tiếp theo.

**3.2. NẮN ẢNH**

Áp dụng ma trận biến đổi affine H trên các hình ảnh dị biệt được gọi là nắn ảnh và không chỉ được sử dụng trong đồ họa máy tính mà còn được sử dụng trong một số thuật toán thị giác máy tính khác. Một trạng thái nắn ảnh hay uốn cong hình ảnh có thể dễ dàng được thực hiện với SciPy bằng cách sử dụng gói thư viện ndimage. Sử dụng câu lệnh sau:

transformed\_im = ndimage.affine\_transform(im,A,b,size)

biến đổi khung hình ảnh im với A là một phép biến đổi tuyến tính và b một vectơ chuyển đổi như trên. Kích thước đối số tùy chọn có thể được sử dụng để chỉ định kích thước của hình ảnh đầu ra. Mặc định là một hình ảnh có cùng kích thước với bản gốc. Để xem cách thức hoạt động, chạy các lệnh sau:

from scipy import ndimage  
  
im = array(Image.open('empire.jpg').convert('L'))  
H = array([[1.4, 0.05, -100], [0.05, 1.5, -100], [0, 0, 1]])  
im2 = ndimage.affine\_transform(im, H[:2, :2], (H[0, 2], H[1, 2]))  
  
figure()  
gray()  
imshow(im2)  
show()

Kết quả như hình ảnh bên phải trong Hình 3.1. Như có thể thấy, các giá trị pixel bị thiếu trong hình ảnh kết quả được lấp đầy bằng các số 0.

 A tall building with a pointed top

Description automatically generated

1. (b)

*Hình 3.1: Một ví dụ về nắn ảnh một hình ảnh bằng cách sử dụng một biến đổi affine. (a) là ảnh gốc, (b) ảnh sau khi được nắn với hàm ndimage.affine\_transform()*

**3.2.1. Đặt một ảnh trong một ảnh khác**

Một ví dụ đơn giản về nắn ảnh theo phép biến đổi affine là đặt một hình ảnh hoặc một phân của hình ảnh bên trong một hình ảnh khác sao cho trùng khớp với các vùng hoặc điểm mốc cụ thể nào đó.

Thêm hàm image\_in\_image() vào warp.py. Hàm này có hai hình ảnh và tọa độ góc của nơi để đặt hình ảnh đầu tiên vào trong hình ảnh thứ hai.

def image\_in\_image(im1,im2,tp):  
 """Put im1 in im2 with an affine transfomation  
 such that corners are as close to tp as possible.  
 tp are homogenerous and counter-clockwise from top left."""  
  
 # points to warp from  
 m,n = im1.shape[:2]  
 fp = array([[0,m,n,0],[0,0,m,n],[1,1,1,1]])  
  
 # compute affine transform and apply  
 H = homography.Haffine\_from\_points(tp,fp)  
  
 im1\_t = ndimage.affine\_transform(im1,H[:2,:2],  
 (H[0,2],H[1,2]),im2.shape[:2])  
 alpha = (im1\_t > 0)  
  
 return (1-alpha) \* im2 + alpha \* im1\_t

Như có thể thấy, điều này được thực hiện khá dễ dàng tử chương trình mẫu bên trên. Khi trộn một hình ảnh bị nắn và hình ảnh thứ hai, tạo ra một bản đồ alpha xác định số lượng từng pixel sẽ lấy từ mỗi hình ảnh. Ở đây sử dụng thực tế là hình ảnh bị nắn chứa đầy các số 0 bên ngoài đường viền của khu vực chênh lệch để tạo ra bản đồ alpha nhị phân. Để thực sự chặt chẽ, có thể thêm một số lượng nhỏ các pixel có giá trị bằng 0 vào hình ảnh đầu tiên. Lưu ý rằng tọa độ hình ảnh ở dạng đồng nhất.

Để thử hàm này, hãy chèn một hình ảnh vào một hình ảnh khác. Các dòng chương trình sau đây sẽ đặt hình ảnh ngoài cùng bên trái của Hình 3.2 vào hình ảnh thứ hai. Các tọa độ được xác định thủ công bằng cách nhìn vào một ô của hình ảnh (trong hình PyLab, tọa độ chuột được hiển thị gần phía dưới). Cũng có thể sử dụng PyLab's ginput() để thực hiện điều này.

import warp  
  
# example of affine warp of im1 onto im2  
im1 = array(Image.open('beatles.jpg').convert('L'))  
im2 = array(Image.open('billboard\_for\_rent.jpg').convert('L'))  
  
# set to points  
tp = array([[264,538,540,264],[40,36,605,605],[1,1,1,1]])  
  
im3 = warp.image\_in\_image(im1,im2,tp)  
  
figure()  
gray()  
imshow(im3)  
axis('equal')  
axis('off')  
show()

* A large white square sign

Description automatically generated  *

*Hình 3.2: Một ví dụ về việc đặt một hình ảnh bên trong một hình ảnh khác bằng cách sử dụng phép biến đổi affine*

Điều này đặt hình ảnh ở phần trên bảng quảng cáo. Một lần nữa lưu ý rằng tọa độ mốc *tp* nằm trong tọa độ đồng nhất. Thay đổi tọa độ sẽ đặt hình ảnh ở phía dưới bên trái vị trí có chữ "for rent" trong ảnh.

tp = array([[675,826,826,677],[55,52,281,277],[1,1,1,1]])

Hàm *Haffine\_from\_point()* thực hiện phép biến đổi affine tốt nhất đối với các điểm tương ứng biết trước. Trong ví dụ trên, đó là các góc hình ảnh và các góc của bảng quảng cáo. Hàng trên t cùng của Hình 3.3 cho thấy điều gì xảy ra nếu cố gắng sử dụng phép biến đổi affine thành hình ảnh biển quảng cáo với nhiều phối cảnh hơn. Không thể chuyển đổi tất cả bốn điểm góc thành các vị trí mục tiêu của chúng với cùng một biến đổi affine. Nếu muốn sử dụng nắn ảnh theo biến đổi affine để tất cả các điểm góc khớp với nhau, có thể thực hiện theo đề xuất sau đây.

Đối với ba điểm, một phép biến đổi affine có thể nắn một hình ảnh sao cho ba cặp điểm đối sánh tốt nhất. Điều này là do một phép biến đổi affine có sáu bậc tự do và ba tương ứng đưa ra chính xác sáu ràng buộc (tọa độ x và y phải khớp với cả ba). Vì vậy, nếu thực sự muốn hình ảnh phù hợp với bảng quảng cáo bằng cách sử dụng các phép biến đổi affine, có thể chia hình ảnh thành hai hình tam giác và nằn chúng một cách riêng biệt. Để làm điều đó, thực hiện như trong chương trình sau: