COMPUTER VISION

**Linear Filtering:**

* Sửa đổi các pixel trong một hình ảnh dựa trên một số chức năng của một vùng lân cận cục bộ của từng pixel.
* Thay thế từng pixel bằng tổ hợp tuyến tính (tổng trọng số) của các lân cận.

**RANSAC:**   
Cho toạ độ của N điểm trên mặt phẳng.

* Giả sử có một số điểm nằm trên đường thẳng, còn các điểm khác là nhiễu (nghĩa là biết chắc có 2 tập hợp con, còn gọi là cluster).
* Hãy tìm phương trình đường thẳng, và cho biết những điểm nào được coi là nằm trên đường thẳng, những điểm nào được coi là nhiễu.

Đầu vào:  
 data - tập hợp các điểm  
 k - số lần lặp  
 t - ngưỡng (threshold) sai số để xác định điểm nào đó có khớp mô hình không  
  
Đầu ra:  
 best\_model - mô hình tốt nhất  
 best\_consensus\_set - tập hợp các điểm khớp với best\_model  
  
best\_model = nil  
best\_consensus\_set = nil  
best\_num\_points = 0  
  
loop k lần  
 consensus\_set = tập hợp 2 điểm ngẫu nhiên thuộc data  
 model = mô hình đường thẳng suy ra từ 2 điểm trên  
  
 với mỗi điểm point thuộc data nhưng không thuộc consensus\_set  
 distance = khoảng cách từ point đến đường thẳng  
 if distance < t (point được coi là khớp với model nếu sai số nhỏ hơn t)  
 thêm point vào consensus\_set  
   
 num\_points = số lượng phần tử trong consensus\_set  
 if num\_points > best\_num\_points  
 best\_model = model  
 best\_consensus\_set = consensus\_set  
 best\_num\_points = num\_points  
  
return best\_model và best\_consensus\_set

Code python cho bài toán: Phát sinh ngẫu nhiên tập điểm S = {M\_i} với i=1..n bao gồm 70% là tập điểm xung quanh đường thẳng y = 3 x + 7 và 30% điểm nhiễu xung quanh đường thẳng này.

Dùng thuật toán RANSAC (random sampling) để ước lượng tham số phương trình đường thẳng. So sánh với hệ số của phương trình đường thẳng gan đầu.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***HOUGH transformation:**

2.2. Cài đặt thuật toán phát hiện đường thẳng sử dụng biến đổi Hough (Hough transformation).

Bước 1: Chọn một tấm ảnh có đường biên cạnh dạng đường thẳng. Ví dụ: Bảng toạ độ, sân bóng.

Bước 2: Lọc lấy biên cạnh của ảnh đầu vào (thuật toán Roberts Cross, Sobel hoặc Canny)

Bước 3: Nhị phân hoá ảnh để xác định tập điểm màu trắng (điểm thuộc biên cạnh)

Bước 4: Cài đặt thuật toán Hough để xác định đường thẳng trong ảnh ở bước 3

Ý tưởng chính của giải thuật phát hiện đường thẳng Hough Transform đó là:

* Dựa trên kết quả phát hiện cạnh để tiến hành phát hiện đường thẳng. Giải thuật phát hiện cạnh phổ biến là [Canny Edge Detection](https://minhng.info/tutorials/xu-ly-anh-opencv-hien-thuc-canny-edge.html).
* Trên mỗi **pixel cạnh** (pixel thuộc cạnh được phát hiện trong ảnh), ta lần lượt thử các phương trình đường thẳng đi qua pixel đó. Số phương trình đường thẳng ta thử càng nhiều thì sẽ cho ra kết quả phát hiện đường thẳng càng tốt (ít bỏ lỡ đường thẳng có trong ảnh hơn). **Pixel cạnh** đó sẽ "vote" thêm 1 giá trị vào ma trận thống kê.
* Sau khi duyệt hết tất cả các **pixel cạnh**, ta sẽ lọc theo một **giá trị ngưỡng** (xác định trước) trên ma trận thống kê để giữ lại (để xác định được) các phương trình đường thẳng có trong ảnh.

Sau khi xác định được các đường thẳng, việc còn lại đơn giản ta chỉ việc vẽ các đường thẳng đó lên ảnh.

*Ưu điểm chính của kỹ thuật biến đổi Hough là nó có thể chịu được các khoảng trống trong các mô tả đặc trưng biên giới và tương đối không bị ảnh hưởng bởi nhiễu hình ảnh.*

*Kỹ thuật Hough đặc biệt hữu ích để tính toán mô tả toàn cầu về (các) đặc trưng (trong đó số lượng lớp giải pháp không cần phải biết trước), được đưa ra (có thể là nhiễu) phép đo cục bộ.*

Đầu tiên, phương trình một đường thẳng có thể được mô tả bằng **rho** và **theta**. Góc theta có giá trị từ 0 -> 180, do đó nếu ta muốn thử các đường thẳng mà mỗi đường thẳng lệch nhau chỉ 1 độ (tức đường thẳng 1, theta = 0 độ; đường thẳng 2, theta = 1 độ, đường thẳng 3, theta = 2 độ; …) thì ta sẽ có 180 đường thẳng có thể thử. Góc theta để đặc tả một đường thẳng là chưa đủ, ta cần thêm khoảng cách **rho** (đọc là rô) để xác định khoảng cách từ đường thẳng đến gốc tọa độ O (từ O ta chiếu vuông góc đến đường thẳng). Giá trị tối đa của rho bằng đường chéo của ảnh (**diagonal\_length**), mà rho lại có thể mang giá trị âm, do đó rho có giá trị trong đoạn [-diagonal\_length, diagonal\_length]. Dĩ nhiên, để đạt kết quả tốt, ta có thể set độ khít của rho = 1 pixel thì ta sẽ có tổng số rho có thể thử trên ảnh là **2 x int(diagonal\_length / rho) + 1** (cộng thêm một là do ta phải tính luôn cả giá trị zero).

Từ các thông tin trên, ta xác định được **ma trận thống kê** (khởi tạo các phần tử trong ma trận này có giá trị zero) có kích thước là: số trường hợp rho x số trường hợp theta. Tiếp theo, ta cần phải duyệt trên mỗi pixel cạnh xem có thể có bao nhiêu phương trình đường thẳng (xác định bởi cặp số (rho, theta)) đi qua được nó. Ý tưởng cho việc hiện thực đó là với mỗi pixel cạnh (x,y), ta tính giá trị rho tương ứng cho chúng lần lượt cho các giá trị theta (180 giá trị theta). Ví dụ cụ thể cho dễ hiểu, giả sử ta có 4000 pixel được cho là cạnh bởi giải thuật Canny, thì ta sẽ có 4000 cặp giá trị (x,y) –> mô tả bằng ma trận A có kích thước [4000, 2]. Sau đó, với **mỗi cặp (x,y)** ta lại muốn thử 180 giá trị theta theo phương trình đường thẳng **ρ = xcosθ + ysinθ** để tính ra **180 giá trị rho**. Để làm được điều đó, ta chỉ cần dựng ma trận B có kích thước [2, 180] chứa các **cosθ** (dòng 1) và **sinθ** (dòng 2) để thu được một ma trận chứa các giá trị **rho** là C có kích thước [4000, 180].

Ta chỉ cần duyệt mỗi phần tử trên ma trận C để "vote" vào **ma trận thống kê X** bằng cách xác định dòng và cột phù hợp để +1 đơn vị. Giá trị rho nằm từ [-num\_rho, num\_rho] trong khi index các dòng của X bắt đầu từ 0, vì vậy **rho\_pos** đối với X sẽ bằng **int(round(vote\_matrix[vr, vc]))+num\_rho**. Sau khi duyệt xong để vote vào X thì ta lọc theo ngưỡng xác định trước để giữ lại các đường thẳng có trong ảnh. Lọc theo ngưỡng trên ma trận X (edge\_matrix) ta sẽ được các index của dòng và index của cột. Từ các giá trị index này ta phải convert về **rho** và **theta** (radian) để hàm cv2.line có thể vẽ đúng!

**MÔ TẢ CÁCH HOẠT ĐỘNG CỦA THUẬT TOÁN**Vì mỗi đường thẳng được xác dịnh bởi 2 giá trị (ρ, θ) nên thuật toán tạo 1 mảng 2 chiều (trong bài gốc gọi là accumulator). Dòng ứng với ρ và cột ứng với θ, kích thước của mảng phụ thuộc vào bạn chọn, và tất nhiên là mảng càng lớn thì càng chính xác và tính toán càng lâu, còn mảng nhỏ thì nhanh hơn nhưng không chính xác bằng.

Khi sử dụng người dùng sẽ truyền giá trị ρ và θ họ mong muốn. Thuật toán sẽ vẽ ρ x θ đường thẳng trên ảnh, với mỗi đường thẳng thuật toán đếm số pixel nằm trên đường thẳng đó, cứ mỗi pixel tìm thấy thì cộng thêm giá trị vào ô ứng với (ρ, θ).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***SIFT:**

Ý tưởng:

* Từ ảnh tìm ra các điểm ảnh đặc biệt, gọi là feature point hay keypoint. Đầu vào và đầu ra của phép biến đổi SIFT: ảnh -> SIFT -> các keypoint
* Để có thể phân biệt keypoint này với keypoint khác cần tìm ra tham số gì đó, gọi là descriptor. 2 keypoint khác nhau thì phải descriptor khác nhau. Thường thì descriptor là chuỗi số gồm 128 số (vector 128 chiều).
* Sau khi áp dụng biến đổi SIFT, ứng với mỗi keypoint, thu được (1) toạ độ keypoint (2) scale và orientation của keypoint (3) descriptor. Các mũi tên trong hình dưới vẽ nhờ vào scale và orientation.

Đặc điểm:

* Keypoint phụ thuộc rất ít vào cường độ sáng, nhiễu, che khuất (một phần ảnh bị che), góc xoay (ảnh bị xoay trong mặt phẳng 2D), thay đổi của tư thế (pose thay đổi trong không gian 3D).
* Nhanh, tốc độ xử lí gần như thời gian thực (realtime).

Như vậy việc khớp ảnh mẫu với ảnh thường chính là việc khớp các keypoint trong 2 ảnh với nhau:

* Áp dụng SIFT cho từng tấm.
* So sánh descriptor của keypoint trong ảnh này với descriptor của keypoint trong ảnh kia. Nếu 2 keypoint có descriptor [gần giống nhau](http://en.wikipedia.org/wiki/Nearest_neighbor_search) thì coi như 2 điểm trên 2 tấm ảnh cùng thuộc về một vật (hoặc vật giống nhau).
* Dựa vào thương của scale và hiệu của orientation của 2 keypoint, có thể biết thêm vật trong hình này to hơn vật trong hình kia bao nhiêu, vật trong hình này xoay bao nhiêu độ so với vật trong hình kia bao nhiêu.