**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA ĐIỆN TỬ – VIỄN THÔNG**

**BỘ MÔN MÁY TÍNH – HỆ THỐNG NHÚNG**

⁃⁃⁃🙞🕮🙜⁃⁃



**BÁO CÁO MÔN HỌC**

CANNY EDGE DETECTION

**GIÁO VIÊN HƯỚNG DẪN:**

**ĐẶNG TẤN PHÁT**

**NHÓM SINH VIÊN THỰC HIỆN:**

Lê Đình Huy 20200219

Tôn Đức Phú Vĩnh 20200420

Thành phố Hồ Chí Minh, tháng 1 năm 2024

1. **Giới thiệu:**

**Canny Edge Detector** là toán tử phát hiện cạnh sử dụng thuật toán nhiều giai đoạn để phát hiện nhiều loại cạnh trong hình ảnh. Nó được phát triển bởi John F. Canny vào năm 1986. Mặc dù đã khá cũ nhưng nó đã trở thành một trong những phương pháp phát hiện cạnh tiêu chuẩn và vẫn được sử dụng trong nghiên cứu.

**Canny Edge Detection** là một kỹ thuật để trích xuất thông tin cấu trúc hữu ích từ các đối tượng thị giác khác nhau và giảm đáng kể lượng dữ liệu cần xử lý. Nó đã được áp dụng rộng rãi trong các hệ thống thị giác máy tính khác nhau. Canny đã phát hiện ra rằng các yêu cầu cho việc áp dụng phát hiện cạnh trên các hệ thống thị giác đa dạng là tương đối tương tự nhau. Do đó, một giải pháp phát hiện cạnh để đáp ứng các yêu cầu này có thể được triển khai trong một loạt các tình huống. Tiêu chí chung cho phát hiện cạnh bao gồm:

* Phát hiện cạnh với tỷ lệ lỗi thấp, có nghĩa là việc phát hiện nên chính xác bắt được càng nhiều cạnh hiển thị trong ảnh càng tốt.
* Điểm cạnh được phát hiện từ toán tử nên định vị chính xác ở trung tâm của cạnh.
* Một cạnh cho trước trong ảnh chỉ nên được đánh dấu một lần, và nếu có thể, nhiễu ảnh không nên tạo ra các cạnh giả.

1. **Thuật toán Canny Edge Detection:**

Quá trình của thuật toán Canny Edge Detection có thể được chia thành **5 bước** khác nhau:

* *Giảm nhiễu (Noise reduction)*
* *Tính toán Gradient độ xám của ảnh (Gradient Calculation)*
* *Áp dụng Non-maximum suppression (Non-maximum suppression)*
* *Ngưỡng kép (Double threshold)*
* *Theo dõi cạnh bằng độ trễ (Edge Tracking by Hysteresis)*

1. **Giảm nhiễu bằng lọc Gaussian**

Không thể tránh khỏi việc tất cả các hình ảnh được chụp từ máy ảnh sẽ có một lượng nhiễu nhất định. Tránh để nhiễu bị nhầm lẫn với các cạnh, phải giảm nhiễu. Để làm mịn hình ảnh, nhân bộ lọc Gaussian được tích hợp với hình ảnh. Bước này sẽ làm mịn hình ảnh một chút để giảm ảnh hưởng của nhiễu rõ ràng lên bộ dò cạnh. Phương trình cho hạt nhân bộ lọc Gaussian có kích thước (2k+1)×(2k+1) như sau:

Trong bài này, ta sẽ áp dụng bộ lọc Gaussian 5x5 với độ lệch chuẩn là 1.4. Áp dụng công thức vào code Python để tạo hạt nhân Gaussian 5x5:

|  |
| --- |
| def gaussian\_kernel(size, sigma):      size = int(size) // 2      x, y = np.mgrid[-size:size+1, -size:size+1]      normal = 1 / (2.0 \* np.pi \* sigma\*\*2)      g =  np.exp(-((x\*\*2 + y\*\*2) / (2.0\*sigma\*\*2))) \* normal  return g |

Điều quan trọng là phải hiểu rằng việc lựa chọn kích thước của nhân Gaussian sẽ ảnh hưởng đến hiệu suất của máy dò. Kích thước càng lớn thì độ nhạy của máy dò với nhiễu càng thấp. Kích thước 5×5 là kích thước phù hợp cho hầu hết các trường hợp.



Hình 1: Ảnh gốc được làm mịn bằng bộ lọc Gaussian để khử nhiễu.

1. **Tính toán Gradient độ xám**

Thuật toán Canny về cơ bản tìm ra các cạnh có cường độ thang độ xám của hình ảnh thay đổi nhiều nhất. Những khu vực này được tìm thấy bằng cách xác định độ dốc của hình ảnh. Độ dốc ở mỗi pixel trong ảnh được làm mịn được xác định bằng cách áp dụng cái được gọi là toán tử Sobel. Bước đầu tiên là ước tính độ dốc theo hướng x và y tương ứng bằng cách áp dụng các hạt nhân:

Từ đó có thể xác định được độ dốc và hướng của cạnh:

Áp dụng công thức vào code Python:

|  |
| --- |
| def sobel\_filters(img):      gx = np.zeros((len(img),len(img[0])))      gy = np.zeros((len(img),len(img[0])))      g = np.zeros((len(img),len(img[0])))  theta = np.zeros((len(img),len(img[0])))      for i in range(1, len(img)-1):          for j in range(1, len(img[0]) - 1):              gx[i][j] = (img[i - 1][j - 1] + 2\*img[i][j - 1] + img[i + 1][j - 1]) - (img[i - 1][j + 1] + 2\*img[i][j + 1] + img[i +1][j + 1])              gy[i][j] = (img[i - 1][j - 1] + 2\*img[i - 1][j] + img[i - 1][j + 1]) - (img[i + 1][j - 1] + 2\*img[i + 1][j] + img[i +1][j + 1])              g[i][j] = min(255, np.sqrt(gx[i][j]\*\*2 + gy[i][j]\*\*2))              theta[i][j] = np.arctan2(gy[i][j], gx[i][j])      return (g, theta) |



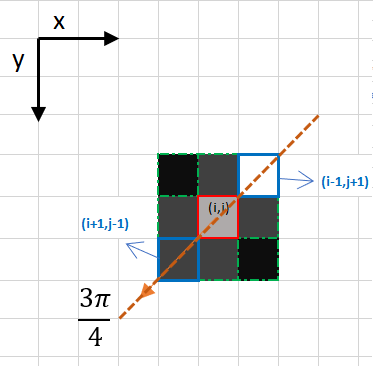
Hình 2: Độ lớn của ảnh sau khi áp dụng bộ lọc Sobel

Kết quả gần như như mong đợi, ta có thể thấy rằng một số cạnh dày và một số khác thì mỏng. Bước Non-Max Suppression sẽ giúp chúng ta giảm thiểu những điều trên.

1. **Non-maximum suppression**

Mục đích của bước này là chuyển đổi các cạnh “mờ” trong ảnh có độ dốc thành các cạnh “sắc nét”. Về cơ bản, điều này được thực hiện bằng cách giữ nguyên tất cả các cực đại cục bộ trong ảnh gradient và xóa mọi thứ khác. Thuật toán dành cho từng pixel trong ảnh gradient:

1. So sánh cường độ cạnh của pixel hiện tại với cường độ cạnh của pixel theo hướng gradient dương và âm.
2. Nếu cường độ cạnh của pixel hiện tại là lớn nhất so với các pixel khác trong mặt nạ có cùng hướng (ví dụ: pixel trỏ theo hướng y sẽ được so sánh với pixel ở trên và bên dưới nó theo trục dọc ), giá trị sẽ được giữ nguyên. Nếu không, giá trị sẽ bị loại bỏ.



Lấy ví dụ từ ảnh trên, hướng là đường chéo màu cam, pixel (i, j) viền đỏ là pixel đang xét và các pixel cùng hướng với pixel trên có viền xanh (i+1, j-1), ( i-1, j +1). Nếu một trong hai pixel đó có cường độ cao hơn pixel đang được xét, thì chỉ có một pixel có cường độ cao hơn được giữ lại. Pixel ( i-1, j +1) có cường độ cao hơn, vì nó có màu trắng (giá trị 255). Do đó, giá trị cường độ của pixel hiện tại (i, j) được đặt thành 0. Nếu không có pixel nào ở hướng cạnh có giá trị cường độ cao hơn thì giá trị của pixel hiện tại được giữ nguyên.

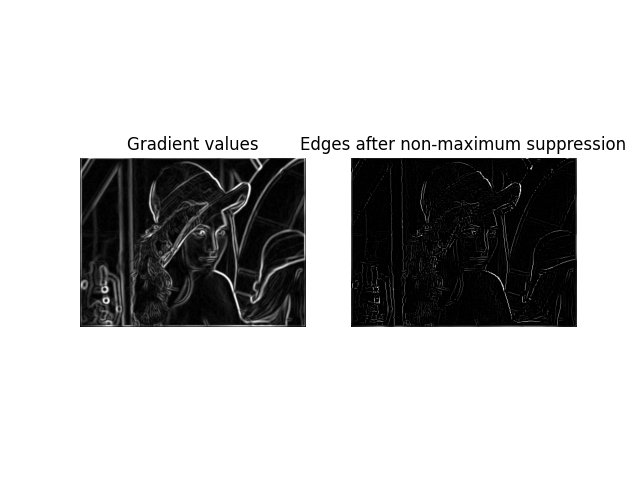
Các bước thực hiện non-maximum suppression:

* Tạo một ma trận được khởi tạo bằng 0 có cùng kích thước của ma trận cường độ gradient ban đầu;
* Xác định hướng của gradient dựa trên giá trị góc từ ma trận góc;
* Kiểm tra xem pixel ở cùng một hướng có cường độ cao hơn pixel hiện đang được xử lý hay không;
* Trả lại hình ảnh được xử lý bằng thuật toán non-maximum suppression.

Áp dụng thuật toán vào code Python:

|  |
| --- |
| def non\_max\_suppression(img, D):      M, N = img.shape      Z = np.zeros((M,N), dtype=np.int32)      angle = D \* 180. / np.pi      angle[angle < 0] += 180      q = 255      r = 255      for i in range(1,M-1):          for j in range(1,N-1):                 #angle 0                  if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):                      q = img[i, j+1]                      r = img[i, j-1]                  #angle 45                  elif (22.5 <= angle[i,j] < 67.5):                      q = img[i+1, j-1]                      r = img[i-1, j+1]                  #angle 90                  elif (67.5 <= angle[i,j] < 112.5):                      q = img[i+1, j]                      r = img[i-1, j]                  #angle 135                  elif (112.5 <= angle[i,j] < 157.5):                      q = img[i-1, j-1]                      r = img[i+1, j+1]                  if (img[i,j] >= q) and (img[i,j] >= r):                      Z[i,j] = img[i,j]                  else:                      Z[i,j] = 0      return Z |

Giải thích code:

* Hàm nhận vào một ảnh (img) và một ma trận góc độ (D) chứa các giá trị góc của gradient tương ứng với từng điểm ảnh.
* ***M, N = img.shape***: Lấy kích thước của ảnh.
* ***Z = np.zeros((M,N), dtype=np.int32)***: Tạo một ma trận zeros có kích thước giống với ảnh để lưu kết quả cuối cùng của non-maximum suppression.
* ***angle = D \* 180. / np.pi***: Chuyển đổi góc từ radian sang độ.
* ***angle[angle < 0] += 180***: Chuyển đổi các giá trị góc âm thành giá trị dương tương ứng để đảm bảo rằng góc nằm trong khoảng từ 0 đến 180 độ.
* ***q = 255, r = 255***: Khởi tạo giá trị q và r là 255.
* Lặp qua từng điểm ảnh trong vùng giữa **(*1 đến M-1 và 1 đến N-1*).**
* Kiểm tra góc của gradient tại điểm ảnh và thực hiện non-maximum suppression dựa trên các trường hợp của góc:
  + Nếu góc gần 0 độ hoặc gần 180 độ, so sánh giá trị pixel hiện tại với pixel bên phải và bên trái.
  + Nếu góc gần 45 độ, so sánh giá trị pixel hiện tại với pixel ở góc dưới bên trái và góc trên bên phải.
  + Nếu góc gần 90 độ, so sánh giá trị pixel hiện tại với pixel ở phía dưới và phía trên.
  + Nếu góc gần 135 độ, so sánh giá trị pixel hiện tại với pixel ở góc trên bên trái và góc dưới bên phải.
* Nếu giá trị pixel hiện tại lớn hơn hoặc bằng cả q và r, thì pixel tại vị trí đó được giữ lại (không bị chấp nhận non-maximum suppression). Ngược lại, pixel đó sẽ được đặt là 0.
* Hàm trả về ma trận kết quả sau khi thực hiện non-maximum suppression.

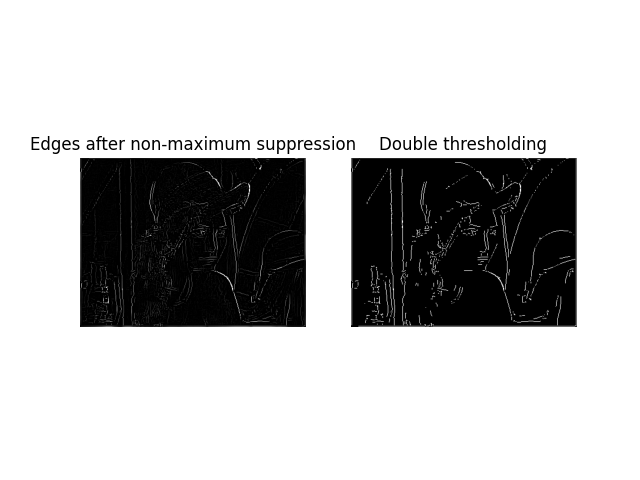
Hình 3: Ảnh sau khi non-maximum suppression

Kết quả thu được hình ảnh với các cạnh mỏng hơn trước. Nhưng vẫn thấy sự không đồng đều truong cường độ của các cạnh

1. **Ngưỡng kép (Double threshold)**

Các pixel cạnh còn lại sau bước non-maximum suppression vẫn được đánh dấu bằng cường độ từng pixel của chúng. Nhiều trong số này có thể là các cạnh thực của ảnh, nhưng một số có thể có thể do nhiễu hoặc sự thay đổi màu sắc chẳng hạn do bề mặt gồ ghề.

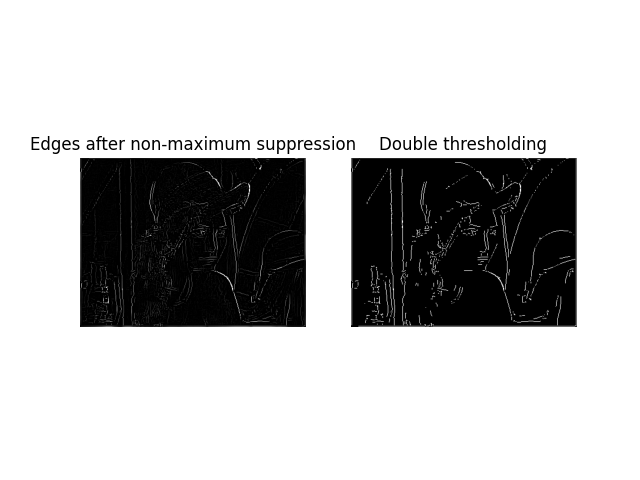
Cách đơn giản nhất để phân biệt giữa những điều này là sử dụng một ngưỡng, sao cho chỉ những cạnh mạnh hơn mới được giữ nguyên một giá trị nhất định. Thuật toán phát hiện cạnh Canny sử dụng ngưỡng kép. Các pixel cạnh mạnh hơn ngưỡng cao được đánh dấu là mạnh; các pixel cạnh yếu hơn ngưỡng thấp sẽ bị chặn và các pixel cạnh giữa hai ngưỡng được đánh dấu là yếu. Hiệu ứng trên ảnh thử nghiệm với ngưỡng 20 và 80.

****Áp dụng thuật toán vào code Python:

|  |
| --- |
| def threshold(img):      res = np.zeros\_like(img, dtype=np.int32)        highThreshold = 80      lowThreshold = 20      weak = np.int32(25)      strong = np.int32(255)        strong\_i, strong\_j = np.where(img > highThreshold)      weak\_i, weak\_j = np.where((img <= highThreshold) & (img >= lowThreshold))        res[strong\_i, strong\_j] = strong      res[weak\_i, weak\_j] = weak        return (res, weak, strong) |

Giải thích code:

* ***res = np.zeros\_like(img, dtype=np.int32)***: Tạo một ma trận zeros có kích thước giống với ảnh đầu vào để lưu kết quả của việc áp dụng ngưỡng.
* ***highThreshold = 80 và lowThreshold = 20***: Đặt ngưỡng cao và ngưỡng thấp cho việc phân loại pixel.
* ***weak = np.int32(25) và strong = np.int32(255)***: Đặt giá trị cho các pixel được coi là yếu và mạnh.
* ***strong\_i, strong\_j = np.where(img > highThreshold)***: Lấy các chỉ số của các pixel có giá trị lớn hơn ngưỡng cao.
* ***weak\_i, weak\_j = np.where((img <= highThreshold) & (img >= lowThreshold))***: Lấy các chỉ số của các pixel có giá trị nằm giữa ngưỡng thấp và ngưỡng cao.
* ***res[strong\_i, strong\_j] = strong***: Đặt giá trị mạnh (255) cho các pixel được coi là mạnh.
* ***res[weak\_i, weak\_j] = weak***: Đặt giá trị yếu (25) cho các pixel được coi là yếu.
* Hàm trả về tuple ***(res, weak, strong)*** chứa ma trận kết quả sau khi áp dụng ngưỡng, giá trị cho pixel yếu và giá trị cho pixel mạnh.

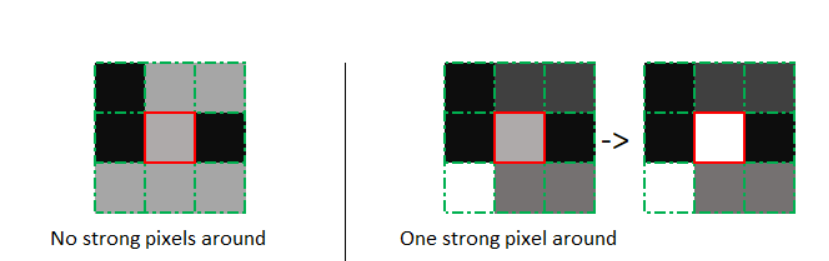
****

Hình 4: Ảnh sau khi sử dụng ngưỡng kép

Kết quả thu được hình ảnh chỉ có 2 giá trị cường độ pixel (mạnh và yếu)

1. **Theo dõi cạnh bằng độ trễ**

Bây giờ chúng ta đã xác định được cạnh mạnh và cạnh yếu là gì, chúng ta cần xác định cạnh yếu nào là cạnh thực. Để làm điều này, chúng tôi thực hiện thuật toán theo dõi cạnh. Các cạnh yếu được kết nối với các cạnh mạnh sẽ là các cạnh thực/thực. Các cạnh yếu không được kết nối với các cạnh mạnh sẽ bị loại bỏ.

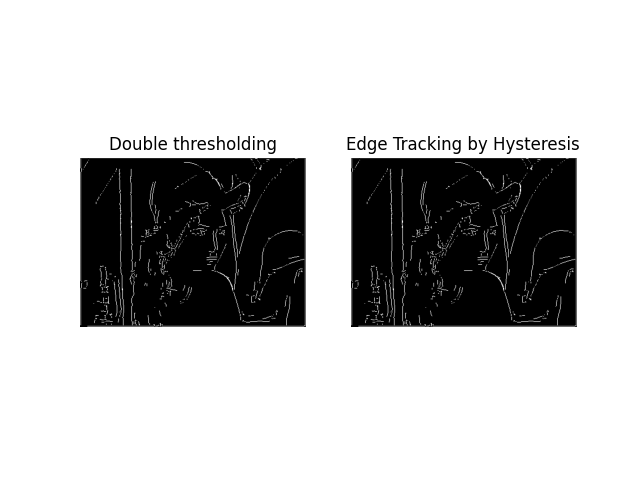


Áp dụng thuật toán vào code Python:

|  |
| --- |
| def hysteresis(img, weak, strong):        for i in range(1, len(img) - 1):          for j in range(1, len(img[0]) - 1):              if (img[i,j] == weak):                  try:                      if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong) or (img[i+1, j+1] == strong)                          or (img[i, j-1] == strong) or (img[i, j+1] == strong)                          or (img[i-1, j-1] == strong) or (img[i-1, j] == strong) or (img[i-1, j+1] == strong)):                          img[i, j] = strong                      else:                          img[i, j] = 0                  except IndexError as e:                      pass      return img |

Giải thích code:

* Lặp qua từng pixel trong vùng giữa ***(1 đến len(img)-1 và 1 đến len(img[0])-1)***.
* Kiểm tra nếu pixel tại vị trí (i, j) có giá trị là weak (pixel yếu).
* Trong trường hợp pixel là yếu, kiểm tra các pixel xung quanh để xem có pixel mạnh nào xung quanh không. Nếu có ít nhất một pixel mạnh trong các hướng xung quanh, đặt giá trị của pixel tại (i, j) thành giá trị strong.
* Nếu không có pixel mạnh xung quanh, đặt giá trị của pixel tại (i, j) thành 0.
* Đối với các trường hợp ngoại lệ (index error), sử dụng try-except để xử lý nếu có lỗi index.
* Hàm trả về ảnh sau khi đã thực hiện quá trình hysteresis.

****

Hình 5: Ảnh sau khi theo dõi cạnh bằng độ trễ

1. **Khuyết điểm**

Mặc dù **Canny Edge Detector** cung cấp một phương pháp tương đối đơn giản nhưng chính xác cho vấn đề phát hiện cạnh. Bên cạnh đó, thuật toán này vẫn còn một vài nhược điểm như sau:

* Bộ lọc Gaussian được áp dụng để làm mịn nhiễu nhưng nó cũng sẽ làm mịn cạnh, được coi là tính năng tần số cao. Điều này sẽ làm tăng khả năng thiếu các cạnh yếu và xuất hiện các cạnh bị cô lập trong kết quả.
* Trong thuật toán phát hiện cạnh Canny truyền thống, sẽ có hai giá trị ngưỡng toàn cục cố định để lọc ra các cạnh sai. Tuy nhiên, khi hình ảnh trở nên phức tạp, các vùng cục bộ khác nhau sẽ cần các giá trị ngưỡng rất khác nhau để tìm chính xác các cạnh thực. Ngoài ra, các giá trị ngưỡng chung được xác định thủ công thông qua các thử nghiệm theo phương pháp truyền thống, dẫn đến tính toán phức tạp khi cần xử lý một số lượng lớn các hình ảnh khác nhau.

1. **Kết luận**

Trong số các phương pháp phát hiện cạnh được phát triển cho đến nay, thuật toán phát hiện cạnh Canny là một trong những phương pháp được định nghĩa chặt chẽ nhất, cung cấp khả năng phát hiện tốt và đáng tin cậy. Nhờ tính tối ưu để đáp ứng ba tiêu chí cho phát hiện cạnh và sự đơn giản của quá trình triển khai, nó trở thành một trong những thuật toán phổ biến nhất cho phát hiện cạnh.