



CÁCH HỌC MOOC HIỆU QUẢ:

- ★ Các nhóm nên cùng nhau xem trước các video bài giảng của thầy theo từng nội dung thầy yêu cầu trong từng mini-project, xem kỹ các tài liệu học tập thầy cung cấp bên cạnh xem các video bài giảng, bao gồm nội dung lý thuyết trong sách, nội dung lý thuyết trong file WORD thầy gửi kèm theo khóa học này.
- ★ Các thành viên trong nhóm nên tra khảo bài nhau để tất cả thành viên trong nhóm đều hiểu rõ nội dung lý thuyết theo từng mini-project thầy yêu cầu.
- ★ Tất cả các ngày trong tuần, các nhóm chủ động hẹn thầy lên lab để được thầy training một cách gần gũi để các em nắm rõ lý thuyết hơn.
- ★ Sau khi cả nhóm đã nắm rõ lý thuyết rồi thì chủ động lên lab báo cáo face-2-face với thầy để lấy điểm quá trình cho từng cột mini-project của toàn khóa học.
- ★ Học MOOC thì thời gian tương tác giữa thầy và các nhóm sẽ nhiều hơn tại phòng LAB nếu các nhóm chủ động và mong muốn gặp thầy chứ không còn bị hạn chế theo một buổi học cứng trong tuần. Chưa hiểu lý thuyết chỗ nào thì cứ chủ động hẹn thầy lên LAB để thầy training cho rõ hơn.
- ★ Học MOOC cả thầy và các em có thể chủ động về mặt thời gian gặp nhau hơn thay vì mỗi tuần 1 buổi theo thời khóa biểu cứng như lớp thường.. Sinh viên sẽ không phải lo bị cấn lịch học giống như khi đăng ký học lớp thường.

MÔ TẢ KHÓA HỌC:

- Học phần trang bị cho sinh viên ngành Công nghệ kỹ thuật điều khiển và tự động hoá những kiến thức cơ bản về xử lý ảnh như khái niệm về điểm ảnh, ảnh màu RGB hay ảnh mức xám (**Grayscale Image**), ảnh nhị phân (**Binary Image**), Histogram của ảnh mức xám và ảnh màu RGB, không gian màu RGB hay các không gian màu khác như CMYK, HSI, HSV, YCrCb, XYZ, các thuật toán chuyển đổi từ ảnh màu RGB sang ảnh mức xám hay ảnh nhị phân, thuật toán tính Histogram cho ảnh mức xám và ảnh màu RGB, các thuật toán chuyển đổi từ không gian màu cơ bản RGB sang các không gian màu khác như CMYK, HSI, HSV, YCrCb, XYZ. Tiếp tiếp sẽ là những kiến thức về các thuật toán xử lý ảnh nền tảng và quan trọng như thuật toán làm mượt ảnh màu (**Color Image Smoothing**), thuật toán làm sắc nét ảnh màu (**Color Image Sharpening**), thuật toán phân đoạn ảnh (**Image Segmentation**), thuật toán nhận dạng đường biên (**Edge Detection**) cho ảnh mức xám và cho ảnh màu RGB,... Ngoài ra, các em có thể tìm hiểu thêm về các thuật toán khác như thuật toán các bộ lọc ảnh (**Filter**), các thuật toán trích xuất đặc trưng (**Feature Detection**), chuyển đổi Wavelet (**Wavelet Transform**),...
- Từ việc hiểu lý thuyết với từng công thức toán học của từng thuật toán, các em có thể tự chuyển đổi sang lập trình cho các thuật toán đó để có thể hiểu rõ hơn hoạt động của các thuật toán với các ngôn ngữ lập trình như C#.NET và Python.
- Dựa trên nền tảng kiến thức về lý thuyết xử lý ảnh như trên, các em có thể triển khai các ứng dụng xử lý ảnh như nhận dạng khuôn mặt, nhận dạng chữ viết, nhận dạng các đồ vật, nhận dạng Barcode, nhận dạng QR code, nhận dạng lane đường, xe cộ, biển báo giao thông, phân loại trái cây theo màu sắc và kích cỡ,... áp dụng các thư viện xử lý ảnh như OpenCV, EmguCV.Net, AForge.Net,... hoặc cũng có thể áp dụng các tool để ứng dụng trí tuệ nhân tạo (AI) cho xử lý ảnh như TensorFlow, YOLO,... chẳng hạn.

HÌNH THỨC TỔ CHỨC VÀ KẾ HOẠCH HỌC TẬP:

- Thầy sẽ chia nhỏ nội dung lý thuyết theo từng thuật toán xử lý ảnh, ngay sau mỗi video bài giảng giải thích lý thuyết thuật toán thầy sẽ cung cấp 2 video lập trình cho thuật toán đó bằng ngôn ngữ lập trình C#.NET và Python để các em có thể hiểu rõ hơn cách thức xử lý của từng thuật toán.
 - Các em chia nhóm tối đa 3 em/nhóm. Khi báo cáo thầy sẽ chọn ngẫu nhiên 1 thành viên trong nhóm để báo cáo và điểm số của nhóm sẽ phụ thuộc vào kết quả báo cáo của thành viên đó, do vậy các em cần phải có tinh thần team-work level max.
📌 Trong buổi học đầu tiên thầy sẽ cung cấp link Google Excel để các em đăng ký nhóm. Chọn đồng đội rất quan trọng khi học môn của thầy do chủ yếu là làm việc nhóm (team-work).
 - Các thành viên trong nhóm cần phải tích cực học tập cùng nhau, phân công nhiệm vụ để hoàn thiện các MINI-PROJECTs, trước khi face-to-face báo cáo với thầy thì các em nên training cho nhau để báo cáo với thầy về nội dung của mini-project nhằm tích lũy điểm số tốt nhất cho phần đánh giá điểm quá trình (lấy trung bình của 15 cột điểm báo cáo mini-projects).
 - Đến khoảng tuần thứ 10, các nhóm sẽ **tự đề xuất nội dung thực hiện cho FINAL-PROJECT** để thầy đánh giá và duyệt cho phép các em thực hiện khi đạt yêu cầu.
 - Kết thúc tuần 15 cũng là lúc các nhóm hoàn thành hết các báo cáo cho 15 bài mini-projects để tích điểm quá trình.
 - Các nhóm có 2 tuần tiếp theo 16-17 để hoàn thiện final-project.
 - Trong quá trình hoàn thiện final-project, nếu nhóm nào hoàn thiện sớm thì có thể liên hệ thầy để được báo cáo sớm. Tuy nhiên, deadline cuối cùng để hoàn thành báo cáo cho final-project là tuần thứ 18 nhé các em.

HÌNH THỨC KIỂM TRA - ĐÁNH GIÁ:

CÁCH TÍNH ĐIỂM

Điểm quá trình (50%): MINI-PROJECTs															Điểm cuối kỳ (50%)
MP #1	MP #2	MP #3	MP #4	MP #5	MP #6	MP #7	MP #8	MP #9	MP #10	MP #11	MP #12	MP #13	MP #14	MP #15	FINAL-PROJECT

Phương pháp tính điểm cuối kỳ cho FINAL-PROJECT:

- Điểm số cuối kỳ của mỗi cá nhân sẽ được tính dựa vào 3 kết quả, như sau:
 1. Điểm tích lũy quá trình.
 2. Kết quả báo cáo final-project của nhóm.
 3. Và phương pháp tính điểm cá nhân QuickMoney, như sau:
 - Ví dụ điểm báo cáo nhóm đạt được là 82,5.
 - Sinh viên A sẽ chấm điểm cho mình và cho hai thành viên còn lại dựa trên sự tự đánh giá công sức của mình và các thành viên khác để hoàn thành final project. Cái này đòi hỏi sự trung thực và chân thành của các thành viên trong nhóm.
 - Tương tự như vậy cho sinh viên B và C. Dĩ nhiên phần tự đánh giá của các cá nhân trong nhóm sẽ được che đi để không em nào biết phần đánh giá của các em khác.
 - Thầy sẽ dựa trên bảng tự chấm điểm này để quyết định điểm báo cáo cuối cùng cho từng sinh viên A, B và C.
 - Ngoài ra sẽ có thêm một cột do chính thầy đánh giá cho từng thành viên trong nhóm để có đánh giá điểm kết quả chính xác nhất cho từng thành viên trong nhóm.
 - Ví dụ minh họa cách tính điểm cuối kỳ cho từng cá nhân trong nhóm (thực tế thì các ô màu xanh đậm phần tự đánh giá của sinh viên sẽ được che đi):

THÔNG TIN LIÊN HỆ VỚI GIẢNG VIÊN:

1. Giảng viên: TS. Nguyễn Văn Thái
2. Bộ môn: Điều khiển tự động, khoa Điện-Điện tử
3. Phòng làm việc: 3DVisionLab, phòng 301, tầng 3 tòa nhà Viện Sư phạm Kỹ thuật Tp HCM
Địa chỉ: 484 Lê Văn Việt, P. Tăng Nhơn Phú A, Quận 9, Tp HCM
4. Số điện thoại di động: 0902 80 7576
5. Email: nguyenvanthy@hcmute.edu.vn
6. Skype: [nguyenvanthy110](#)
7. Facebook: <https://www.facebook.com/nguyen.v.thai.10>
8. Zalo: 0902 80 7576

 Các em có thể kết bạn FB với thầy, hoặc LIKE page của LAB để theo dõi các bài viết mới của thầy và LAB:

Page của 3DVisionLAB:

<https://www.facebook.com/profile.php?id=100055390519606>

Page của IoTVision LAB:

<https://www.facebook.com/IoTVisionHCMUTE>

 Các em kết bạn Zalo với thầy để thầy add vào group Zalo của lớp. Group FB thầy dùng post những bài viết mang tính phân luồng để các em dễ theo dõi, còn group Zalo dùng để tương tác nhanh trong quá trình học tập, bao gồm trao đổi các vấn đề trong học tập, đặt câu hỏi, trả lời các câu hỏi, share video hay hình ảnh.

 **Lưu ý:** Khi kết bạn ZALO các em nhớ để thông tin là Sinh viên lớp XLA để thầy biết mà add vào cho đúng group môn học nhé, do thầy có dạy các môn khác như Robot, SCADA, IoT.

LINK TẢI SÁCH HỌC CHÍNH:

Rafael C. Gonzalez, Richard E. Woods - Digital Image Processing (2008, Prentice Hall):

<https://drive.google.com/file/d/1Z6RyDBam35TeqiqoC33ekFYfRXLvwRtm/view?usp=sharing>

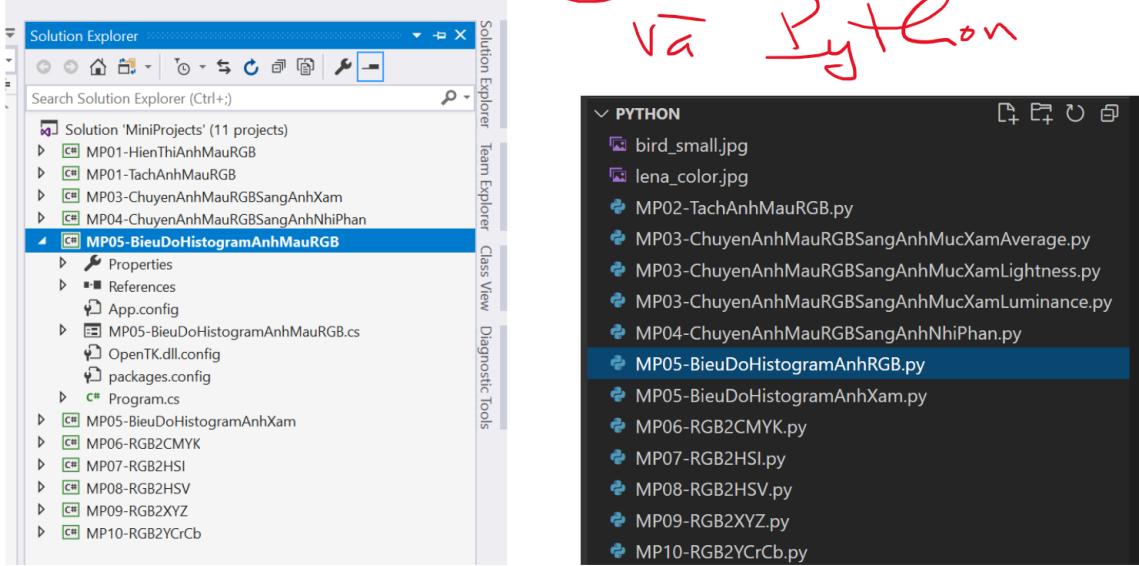
MỤC LỤC

VIDEO GIỚI THIỆU MÔN HỌC

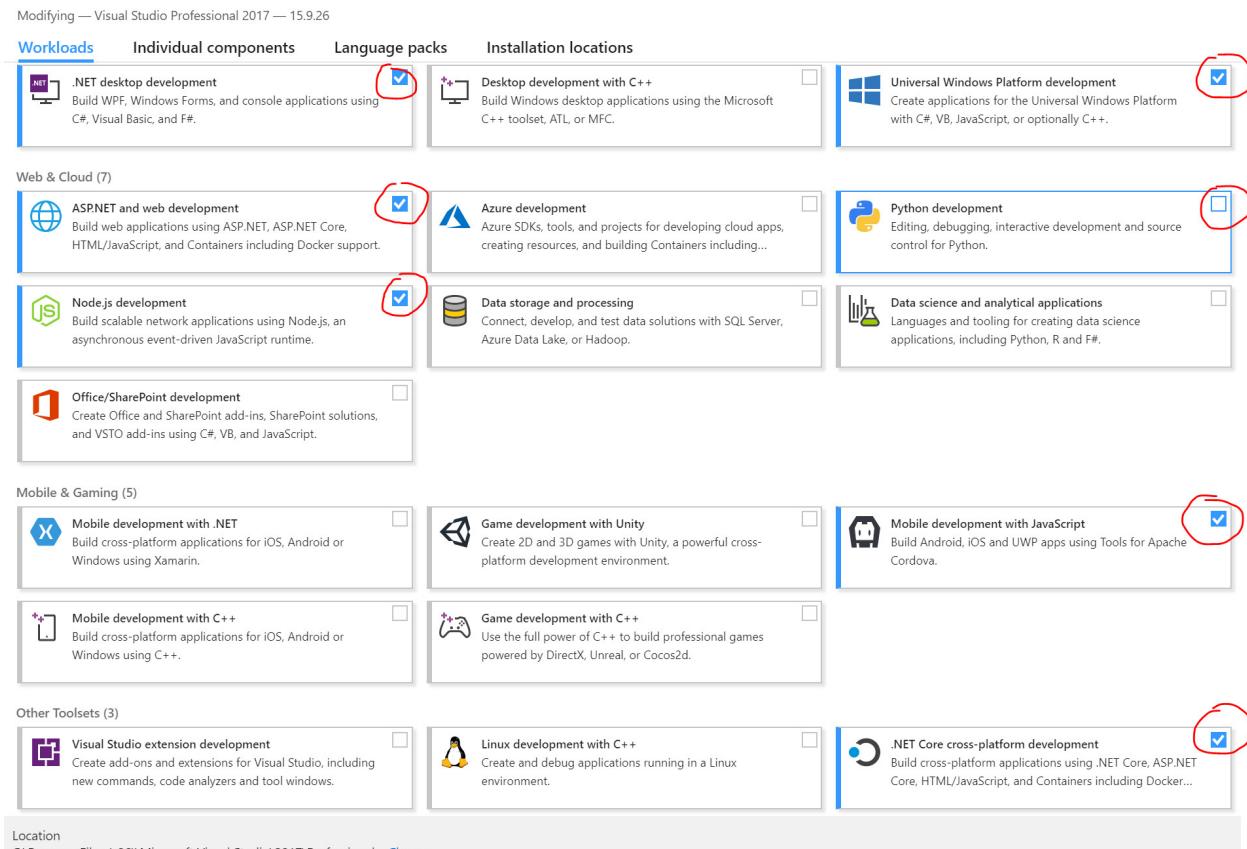
1. HƯỚNG DẪN NHÚNG THƯ VIỆN EmguCV VÀO C#.NET
2. LOAD ẢNH TỪ FILE ẢNH VÀ HIỂN THỊ DÙNG EmguCV
3. KHÔNG GIAN MÀU RED - GREEN - BLUE (RGB Color Space)
 - ♥ MINI-PROJECT 1
 - ♥ MINI-PROJECT 2
6. ẢNH MỨC XÁM (Grayscale Image) - CHUYỂN ĐỔI ẢNH MÀU RGB SANG ẢNH MỨC XÁM
 - 4.1. Lập trình chuyển đổi ảnh màu RGB sang Grayscale dùng phương pháp Average
 - 4.2. Lập trình chuyển đổi ảnh màu RGB sang Grayscale dùng phương pháp Lightness
 - 4.3. Lập trình chuyển đổi ảnh màu RGB sang Grayscale dùng phương pháp Luminance
 - ♥ MINI-PROJECT 3
7. ẢNH NHỊ PHÂN (BINARY IMAGE) - CHUYỂN ĐỔI ẢNH MÀU RGB SANG ẢNH NHỊ PHÂN
 - ♥ MINI-PROJECT 4
8. BIỂU ĐỒ HISTOGRAM CỦA ẢNH KỸ THUẬT SỐ
 - 6.1. Lập trình tính và vẽ biểu đồ Histogram cho ảnh mức xám bằng ngôn ngữ C#.NET và Python
 - 6.2. Lập trình tính và vẽ biểu đồ Histogram cho 3 kênh màu R-G-B bằng ngôn ngữ C#.NET và Python
 - ♥ MINI-PROJECT 5
9. CHUYỂN ĐỔI KHÔNG GIAN MÀU (Color Space)
 - 7.1. Chuyển đổi không gian màu RGB sang không gian màu CMYK
 - ♥ MINI-PROJECT 6
 - 7.2. Chuyển đổi không gian màu RGB sang không gian màu HSI
 - ♥ MINI-PROJECT 7
 - 7.3. Chuyển đổi không gian màu RGB sang không gian màu HSV
 - ♥ MINI-PROJECT 8
 - 7.5. Chuyển đổi không gian màu RGB sang không gian màu XYZ
 - ♥ MINI-PROJECT 9
 - 7.4. Chuyển đổi không gian màu RGB sang không gian màu YCrCb
 - ♥ MINI-PROJECT 10
10. THUẬT TOÁN LÀM MƯỢT ẢNH MÀU RGB (Color Image Smoothing)
 - ♥ MINI-PROJECT 11
11. THUẬT TOÁN LÀM SẮC NÉT ẢNH MÀU RGB (Color Image Sharpening)
 - ♥ MINI-PROJECT 12
12. THUẬT TOÁN PHÂN ĐOẠN ẢNH (Segmentation)
 - ♥ MINI-PROJECT 13
13. THUẬT TOÁN NHẬN DẠNG ĐƯỜNG BIÊN (Edge Detection)
 - 11.1. Nhận dạng đường biên dùng phương pháp Sobel cho ảnh mức xám (Edge Detection using Sobel for grayscale image)
 - ♥ MINI-PROJECT 14
 - 11.2. Nhận dạng đường biên dùng phương pháp Sobel cho ảnh màu RGB (Edge Detection using Sobel for RGB image)
 - ♥ MINI-PROJECT 15
14. YÊU CẦU ĐỐI VỚI FINAL-PROJECT

❤️🔥☕️🌹 Các em nên tổ chức các file trong C#.NET và Python như thày gợi ý dưới đây để dễ quản lý các bài mini-projects của mình trong suốt quá trình học môn XLA của thày nhen các em.

Tổ chức file trong C#.NET
và Python



 Các em có thể tải và cài đặt phiên bản Visual Studio cao hơn, như 2022 mới nhất. Còn nếu cài VS2017 thì khi cài thì mình có thể chọn các mục này để cài đặt nha các em:



☕️🌾🔥 Sau khi cài xong VS2017 các em update lên version sau cùng của VS2017

Visual Studio Installer

Products

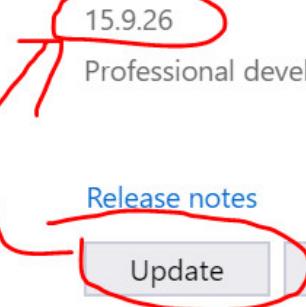
Installed

 Visual Studio Professional 2017
15.9.26

Professional developer tools and services for small teams

[Release notes](#)

[Update](#) [Launch](#) More ▾



Available

 Visual Studio Enterprise 2017
15.9.27

Microsoft DevOps solution for productivity and coordination across teams of any size

[License terms](#) | [Release notes](#)

[Install](#)

More ▾

 Link tải Visual Studio 2017 (Đừng dùng bản 2019 nha các em). Các em có thể tự tải VS phiên bản cao hơn như 2021 hay 2022 từ nhiều nguồn khác trên internet chứ không nhất thiết dùng VS2017.

 Chú ý: trong link có key cho VS2017 luôn nha các em.

<https://drive.google.com/drive/folders/0B2FMaRmsx2cHdE5QLUhkV1dFYmc?resourcekey=0-n8zveitescZWctlGs7aP1w&usp=sharing>

 Ngoài ra, các em cần phải tự tải Visual Code của Microsoft để biên soạn code cho Python, Visual Code thì Microsoft cho tải miễn phí nên không cần phải key.

 Để lập trình được cho Python thì các em cần phải cài đặt Python, xem hướng dẫn cài đặt Python tại trang chính.

⭐️ Hình dùng cho các bài tập xử lý ảnh: cô gái Lena, một cô gái đã trở thành huyền thoại của dân lập trình xử lý ảnh.

Link tải ảnh gốc Lena (.jpg):

https://www.cosy.sbg.ac.at/~pmeerw/Watermarking/Lena.html?fbclid=IwAR0Bin4GSyj3ZICz9kBJoIQQoAGrY4J8_xX7GdOSkm6IMEaQ0HeOlkl8IXM

Link tải ảnh gốc Lena (.png):

<https://drive.google.com/file/d/1BH0NYZdwmlgoKzTPKH8FsA3trLAGvcjP/view?usp=sharing>

🌹 Thêm một hình gốc nữa phục vụ cho việc test các thuật toán của môn XLA của thầy:

Hình small bird để các e test thuật toán của mình và đối chiếu với kết quả của thầy ha:

https://drive.google.com/file/d/1nBn7JbSwG2diKZPTJbNv7aLil_Lsnlte/view?usp=sharing

🔥 MINI-PROJECT 1:

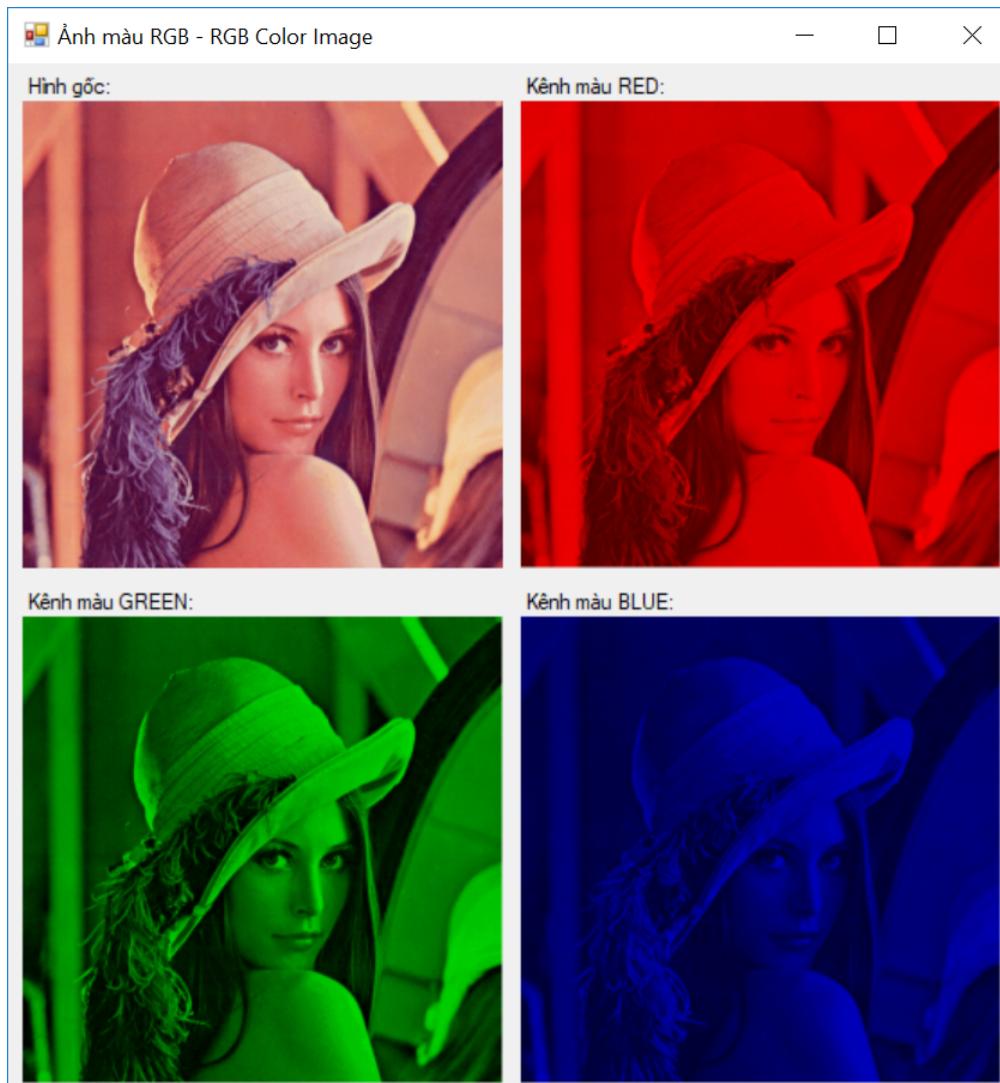
- ⌚ Nhúng thư viện EmguCV vào Win Form Project trong Visual C#.NET 2017
- ⌚ Load hình cô gái Lena lên một PictureBox.
- ⌚ Tách hình ảnh màu cô gái Lena thành 3 lớp (layer) RED - GREEN - BLUE theo hình kết quả thầy gửi trên FB Group.

💌 Thầy sẽ mở link để các em upload video thuyết trình lập trình của mình, sau đó các em copy link video của mình và paste vào ô tên mình ở file excel online để thầy click vào và mở video để chấm điểm cho các em.

Các em submit video và đặt tên theo quy ước: STT-Họ tên SV. STT các em lấy theo số thứ tự trong danh sách file excel online, lưu ý những em có STT từ 1-9 thì phải thêm số 0 phía trước, thầy ví dụ: 05-Nguyễn Trung Dũng

💌 Trong Giao diện nhớ thêm LABEL ghi tên và MSSV của mình nha, bài này là cá nhân, không phải nhóm nghen các em.

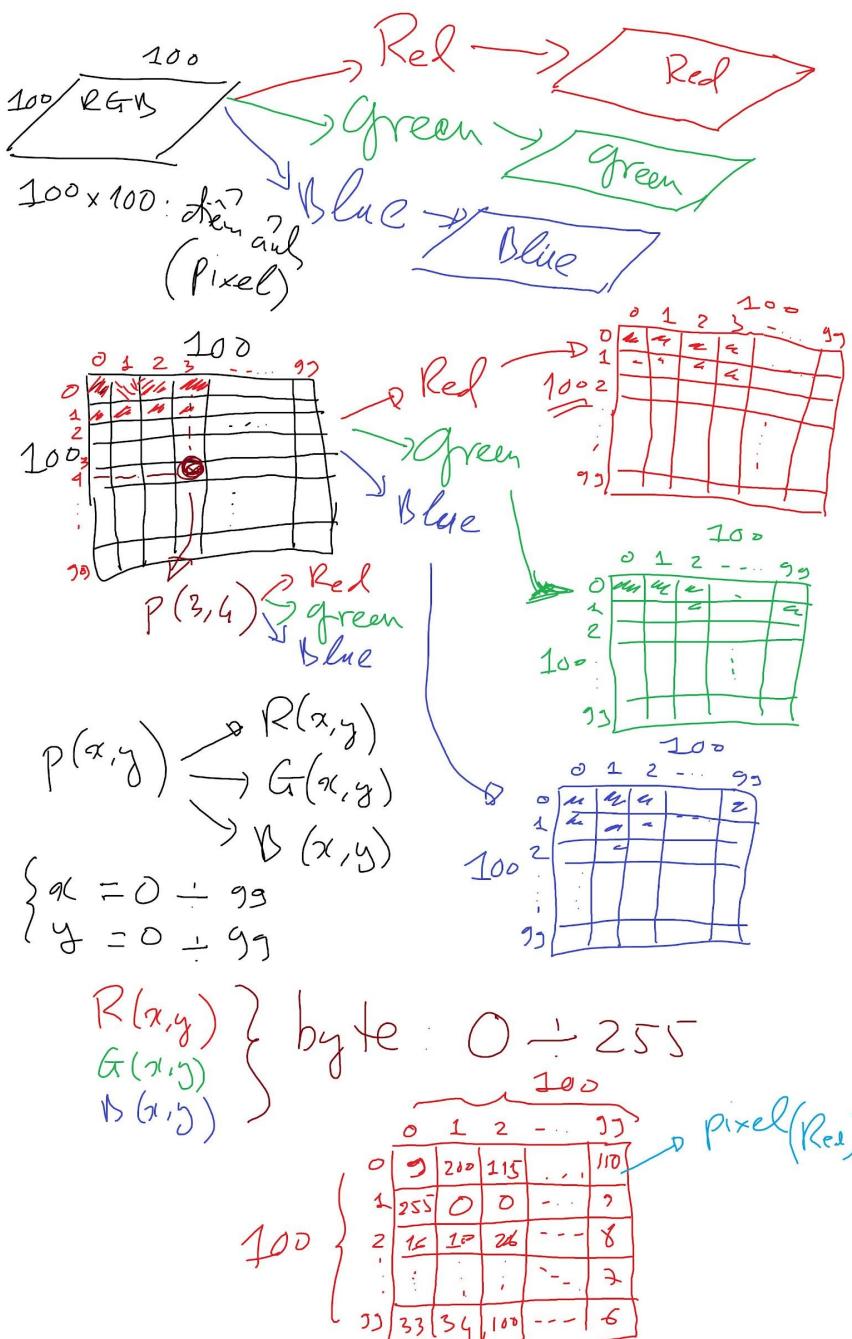
Kết quả C#.NET:



❤ Các videos bài giảng của thầy:

1. UTEX#01 - Nhúng thư viện xử lý ảnh EmguCV vào Visual Studio C# NET
<https://youtu.be/nqkAw6etBOw>
2. UTEX#02 Load ảnh từ file và hiển thị dùng EmguCV
<https://youtu.be/7Ls4s0vFb00>
3. UTEX#03 - Không gian màu RGB, tách và hiển thị từng kênh màu R-G-B
<https://youtu.be/z4N1WchsjP4>
4. UTEX#04 - Lập trình tách màu R G B bằng ngôn ngữ C#.NET
<https://youtu.be/4wdG7RnYmC0>

❤ Điểm ảnh (pixel) màu RGB trong một ảnh màu:

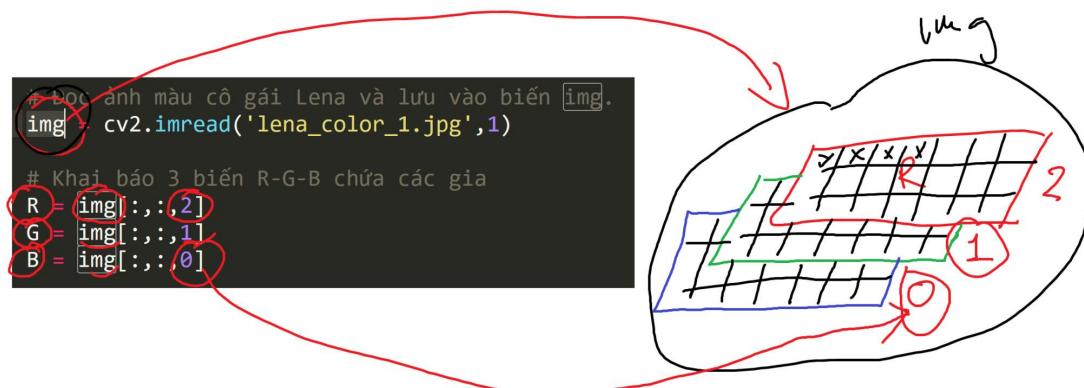


MINI-PROJECT 2:

⌚ Lập trình bằng Python tách hình màu RGB cô gái huyền thoại Lena ra 3 kênh màu RED - GREEN - BLUE

- ⌚ Chụp hình màn hình code và insert vào ô Hình code
- ⌚ Chụp hình box hình RED và insert vào ô Hình RED
- ⌚ Chụp hình box hình GREEN và insert vào ô Hình GREEN
- ⌚ Chụp hình box hình BLUE và insert vào ô Hình BLUE

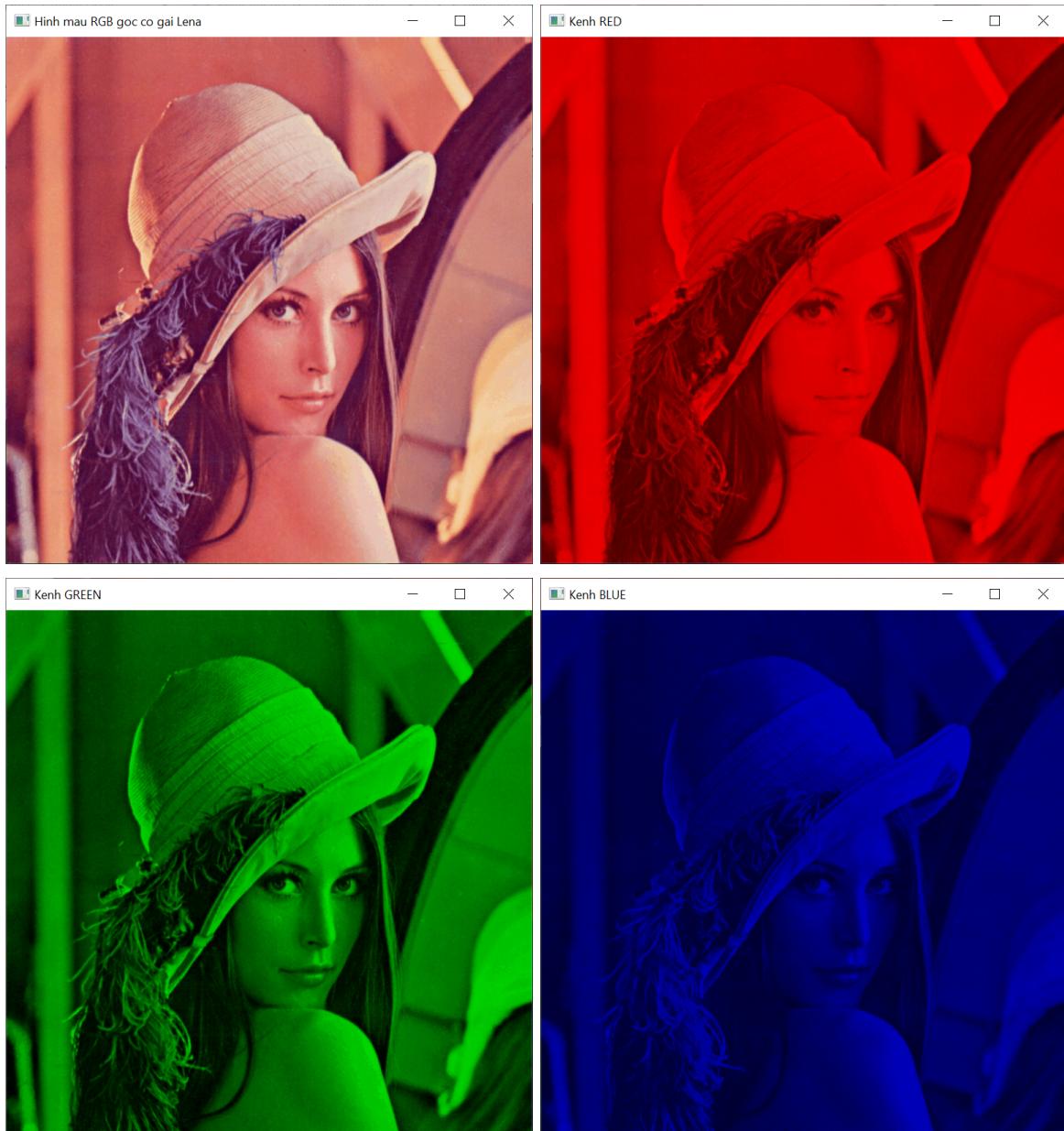
❤️ Chú ý: Trong code nhớ có đoạn ghi chú TÊN & MSSV, các box hình đều có TÊN & MSSV để thầy chấm nha các em.



```

6  img = cv2.imread('bird_small.jpg', cv2.IMREAD_COLOR)
7
8  # Lấy kích thước của ảnh
9  height, width, channel = img.shape
10
11 # Khai báo 3 biến để chứa hình 3 kênh R-G-B
12 red = np.zeros((height, width, 3), np.uint8)    # số 3 là ba kênh R-G-B, mỗi kênh 8bit
13 green = np.zeros((height, width, 3), np.uint8)
14 blue = np.zeros((height, width, 3), np.uint8)
15
16 # Ban đầu set zero cho tất cả điểm ảnh có trong cả 3 kênh trong mỗi hình
17 red[:] = [0, 0, 0]
18 green[:] = [0, 0, 0]
19 blue[:] = [0, 0, 0]
20
21 # Mỗi hình là một ma trận 2 chiều nên sẽ dùng 2 vòng for
22 # để đọc hết các điểm ảnh (pixel) có trong hình
23 for x in range(width):
24     for y in range(height):
25         # Lấy giá trị điểm ảnh tại vị trí (x, y)
26         R = img[y, x, 2]          Các Câu Câu
27         G = img[y, x, 1]          đó c
28         B = img[y, x, 0]          Kết Vô Chiêu
29
30         # Thiết lập màu cho các kênh
31         red[y, x, 2] = R
32         green[y, x, 1] = G
33         blue[y, x, 0] = B          Đọc Ảnh Bằng OpenCV
34
35 # Hiển thị hình dùng thư viện OpenCV
36 cv2.imshow('Hình màu RGB gốc cô gái Lena', img)
37 cv2.imshow('Kênh RED', red)
38 cv2.imshow('Kênh GREEN', green)
39 cv2.imshow('Kênh BLUE', blue)
Câu Python
```

Kết quả Python:



❤️ Videos bài giảng của thầy:

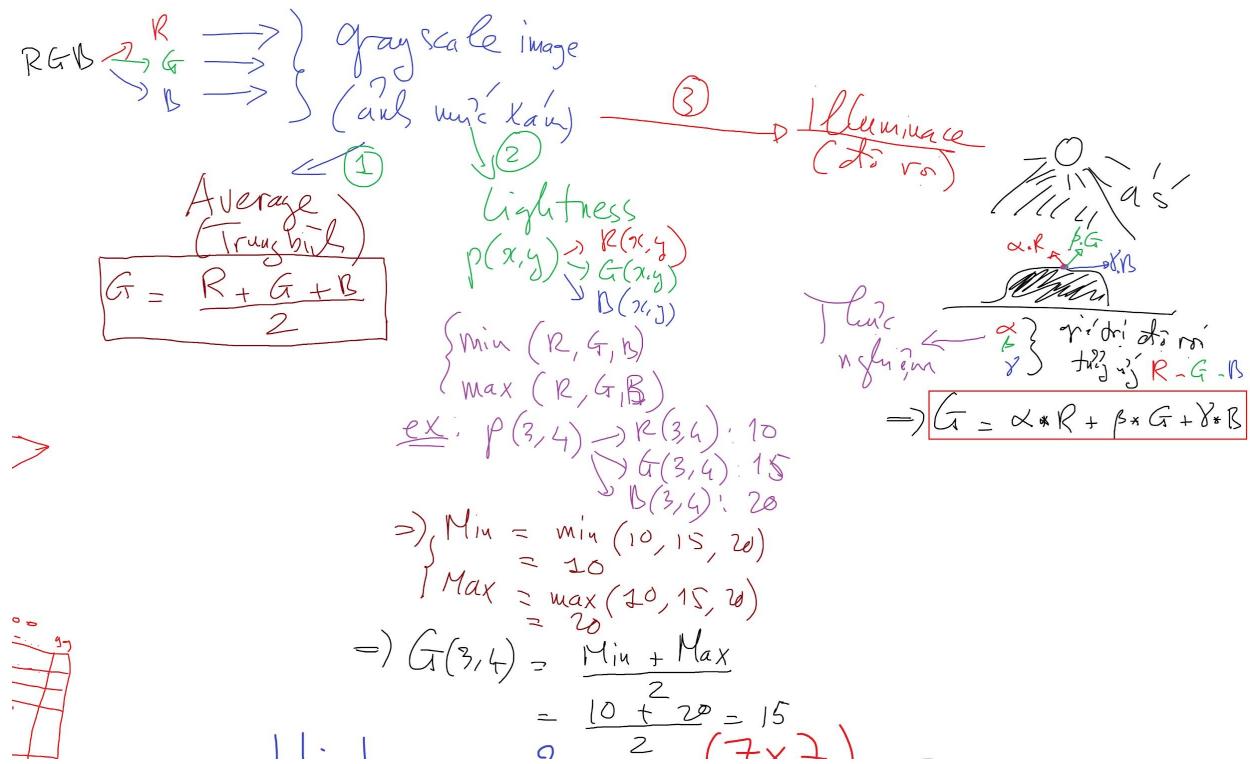
1. UTEX#05 - Lập trình tách màu R G B bằng ngôn ngữ Python
<https://youtu.be/DApZEHS9TzM>

MINI-PROJECT 3:

🕒 Hiểu lý thuyết chuyển đổi hình ảnh màu RGB sang ảnh mức xám Grayscale (xem video bài giảng của thầy)

- 🕒 Lý thuyết chuyển đổi RGB sang Grayscale bằng phương pháp Average
- 🕒 Lý thuyết chuyển đổi RGB sang Grayscale bằng phương pháp Lightness
- 🕒 Lý thuyết chuyển đổi RGB sang Grayscale bằng phương pháp Luminance
- 🕒 Lập trình chuyển đổi RGB sang Grayscale bằng phương pháp Average bằng C#.NET
- 🕒 Lập trình chuyển đổi RGB sang Grayscale bằng phương pháp Lightness bằng C#.NET
- 🕒 Lập trình chuyển đổi RGB sang Grayscale bằng phương pháp Luminance bằng C#.NET
- 🕒 Lập trình chuyển đổi RGB sang Grayscale bằng phương pháp Average bằng Python
- 🕒 Lập trình chuyển đổi RGB sang Grayscale bằng phương pháp Lightness bằng Python
- 🕒 Lập trình chuyển đổi RGB sang Grayscale bằng phương pháp Luminance bằng Python

❤️ Thầy gồm 3 công thức chuyển đổi ảnh màu RGB sang ảnh mức xám (grayscale) như hình dưới cho các em dễ hiểu bài hơn nha:



❤️ Các videos bài giảng lý thuyết và hướng dẫn lập trình chuyển đổi ảnh màu RGB sang ảnh mức xám bằng ngôn ngữ lập trình C#.NET và Python:

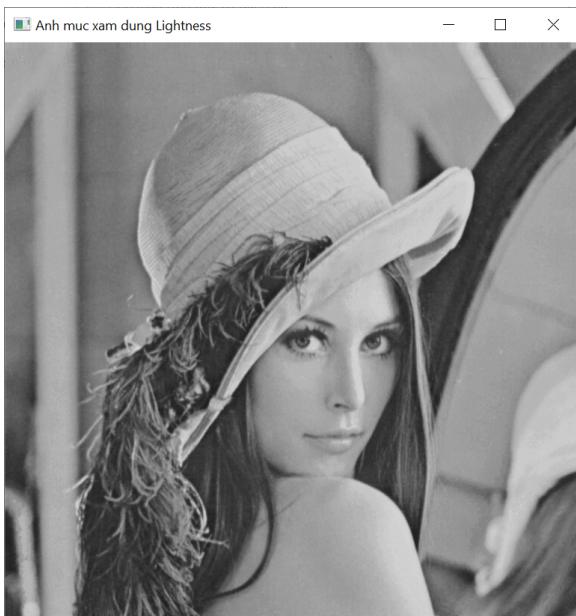
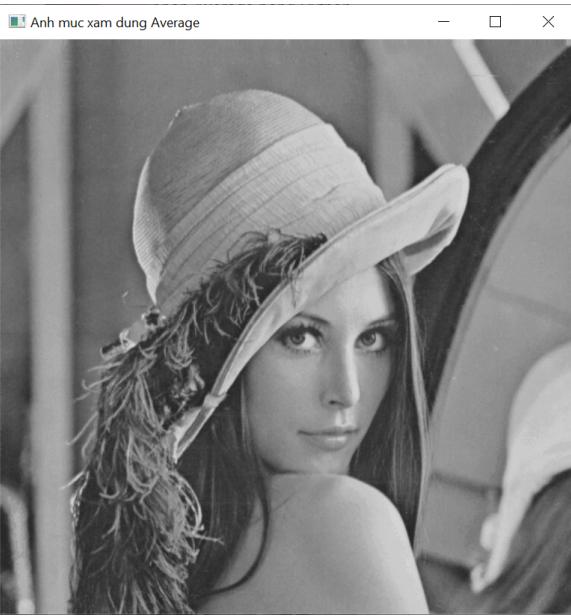
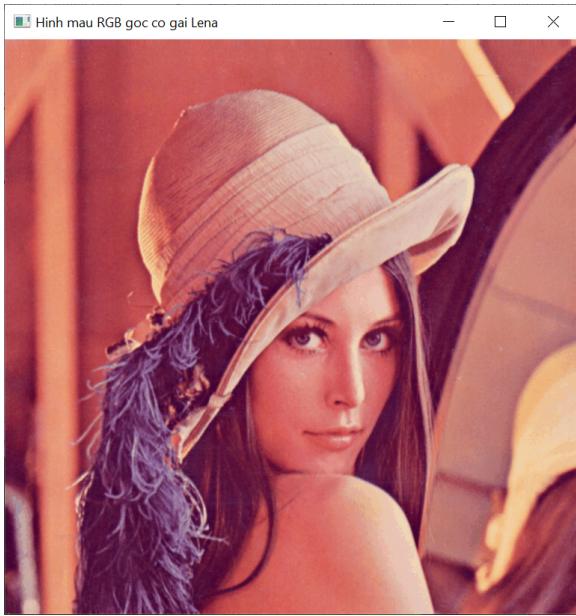
1. UTEX#06 - Thuật toán chuyển đổi ảnh màu RGB sang ảnh mức xám (Grayscale Image)
<https://youtu.be/yeeUnZa73-c>
2. Xử lý ảnh #5 - Chuyển ảnh màu RGB sang ảnh mức xám Grayscale dùng phương pháp Average - Ngôn ngữ C#.NET
https://youtu.be/-R_IYCIRH1g
3. Xử lý ảnh #4 - Chuyển ảnh màu RGB sang ảnh mức xám Grayscale dùng phương pháp Lightness - Ngôn ngữ C#.NET
<https://youtu.be/WRs8Go8G7os>

4. Xử lý ảnh #6 - Chuyển ảnh màu RGB sang ảnh mức xám Grayscale dùng phương pháp Luminance - Ngôn ngữ C#.NET
<https://youtu.be/9AChfGqsMxA>
5. UTEX#07 - Lập trình bằng Python chuyển đổi ảnh màu RGB sang ảnh Grayscale bằng phương pháp Average
<https://youtu.be/ck8XBiHhc60>
6. UTEX#08 - Lập trình bằng Python chuyển đổi ảnh màu RGB sang ảnh Grayscale bằng phương pháp Lightness
<https://youtu.be/jkBCHnZSlqM>
7. UTEX#09 - Lập trình bằng Python chuyển đổi ảnh màu RGB sang ảnh Grayscale bằng phương pháp Luminance
<https://youtu.be/z-e0baurGTo>

Kết quả C#.NET:



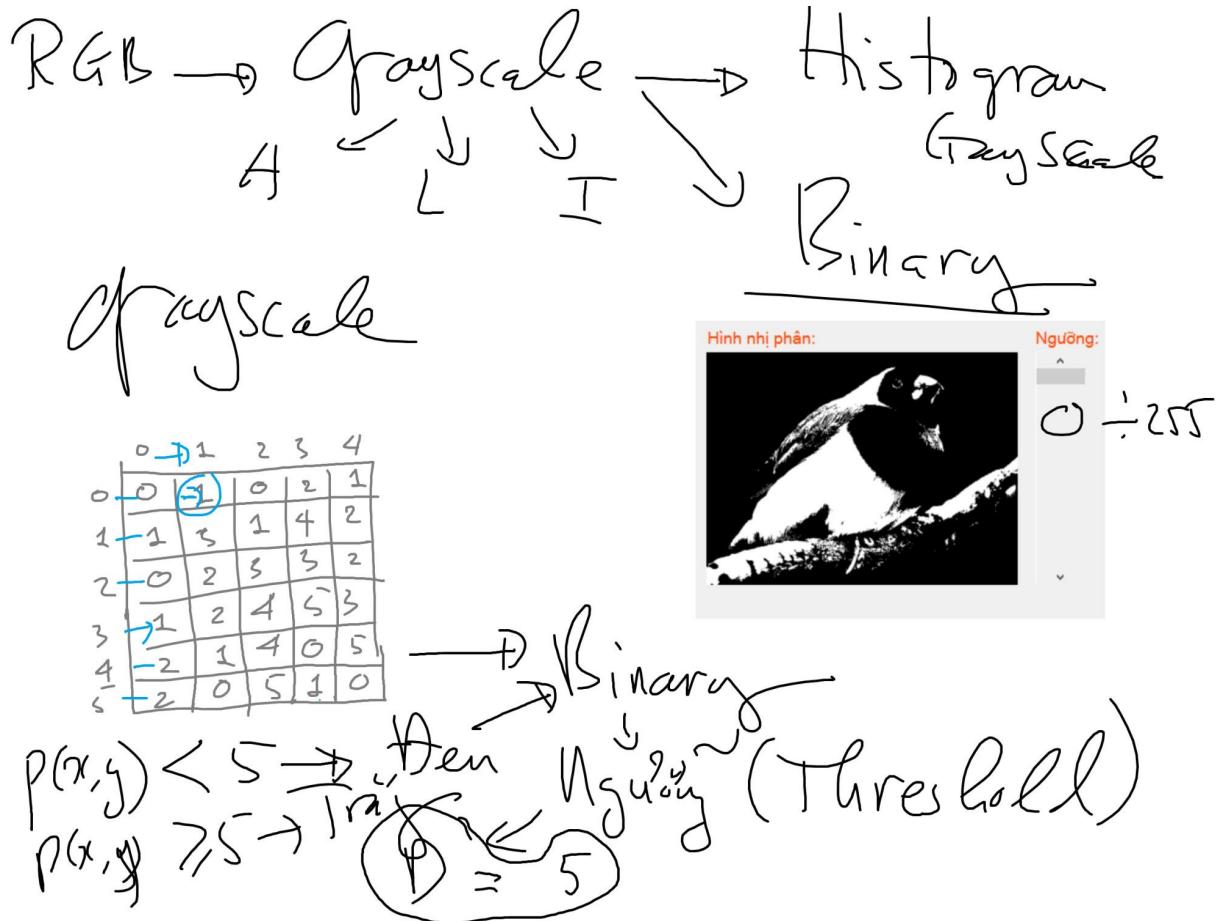
Kết quả Python:



🔥 MINI-PROJECT 4:

- ⌚ Hiểu lý thuyết chuyển đổi hình ảnh màu RGB sang ảnh nhị phân Binary (xem video bài giảng của thầy)
- ⌚ Lập trình chuyển đổi RGB sang ảnh nhị phân Binary bằng C#.NET
- ⌚ Lập trình chuyển đổi RGB sang ảnh nhị phân Binary bằng Python

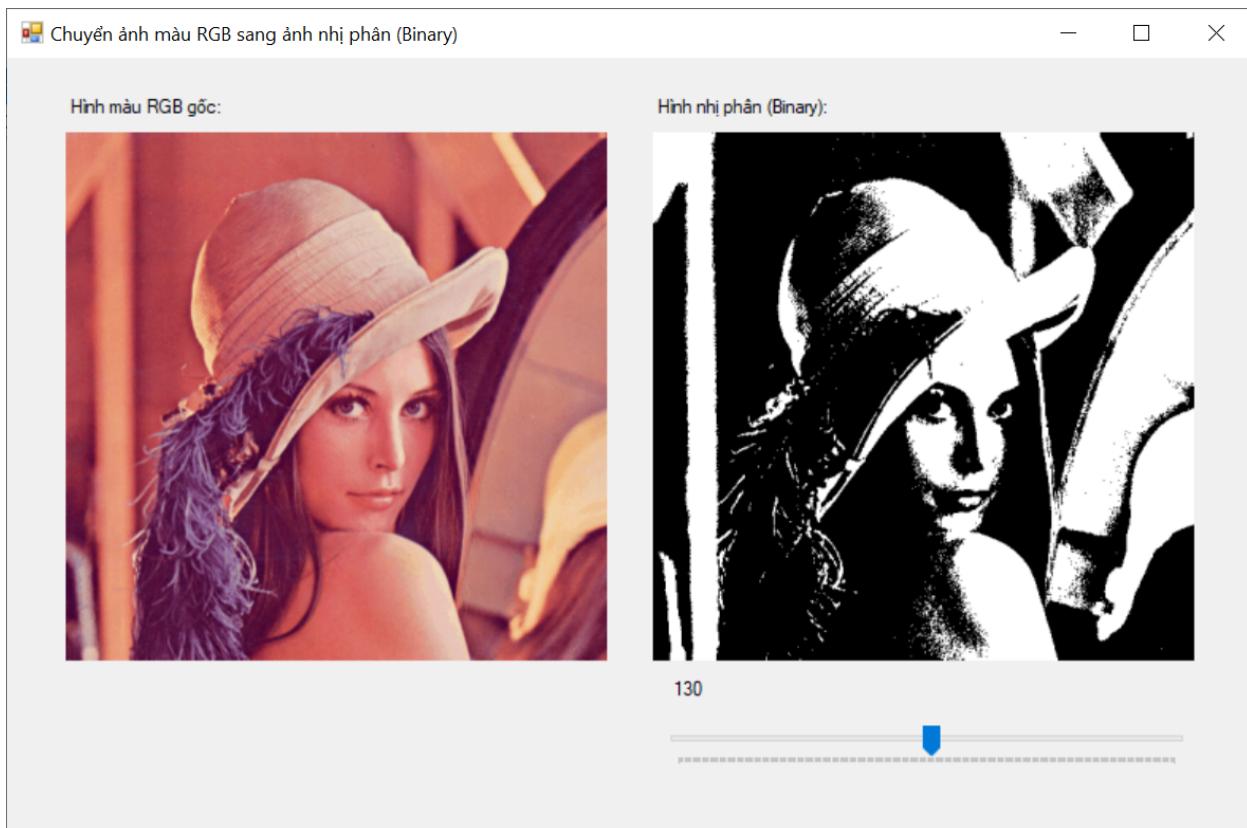
❤️ Thầy viết lại cách thức chuyển đổi ảnh màu RGB sang ảnh nhị phân để các em dễ hiểu hơn:



❤️ Videos bài giảng của thầy:

1. Xử lý ảnh #7 - Chuyển ảnh màu RGB sang ảnh nhị phân Binary
<https://youtu.be/-t6zzWYtaBU>
2. UTEX#10 - Lập trình bằng Python chuyển đổi ảnh màu RGB sang ảnh nhị phân Binary
<https://youtu.be/gfyOnjBFApY>

Kết quả C#.NET



Kết quả Python:



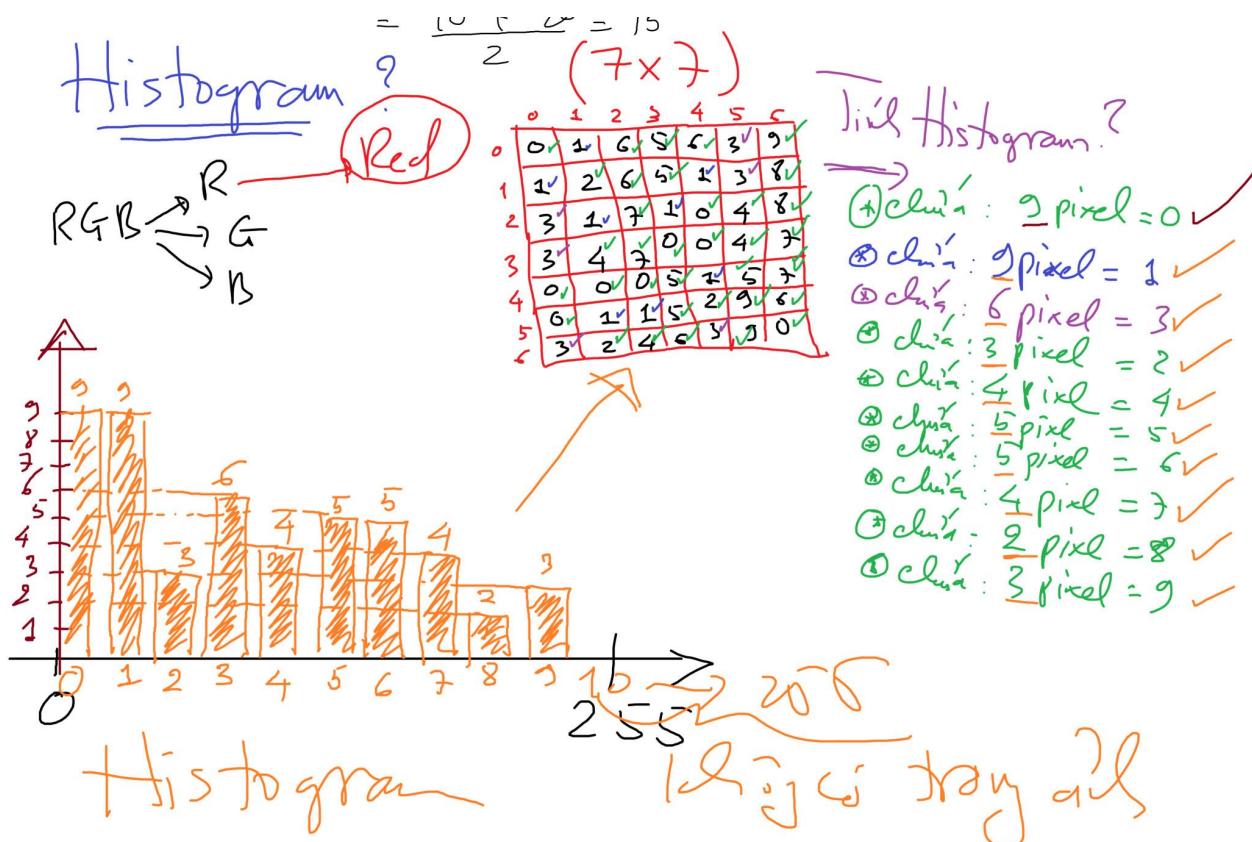
MINI-PROJECT 5:

- ⌚ Hiểu lý thuyết tính biểu đồ histogram (xem video bài giảng của thầy)
- ⌚ Lập trình vẽ biểu đồ histogram cho ảnh mức xám bằng ngôn ngữ C#.NET
- ⌚ Lập trình vẽ biểu đồ histogram cho ảnh mức xám bằng ngôn ngữ Python
- ⌚ Lập trình vẽ biểu đồ histogram cho ảnh màu RGB bằng ngôn ngữ C#.NET
- ⌚ Lập trình vẽ biểu đồ histogram cho ảnh màu RGB bằng ngôn ngữ Python
- ⌚ Lưu ý: Sử dụng hình bird_small.jpg cho ảnh đầu vào

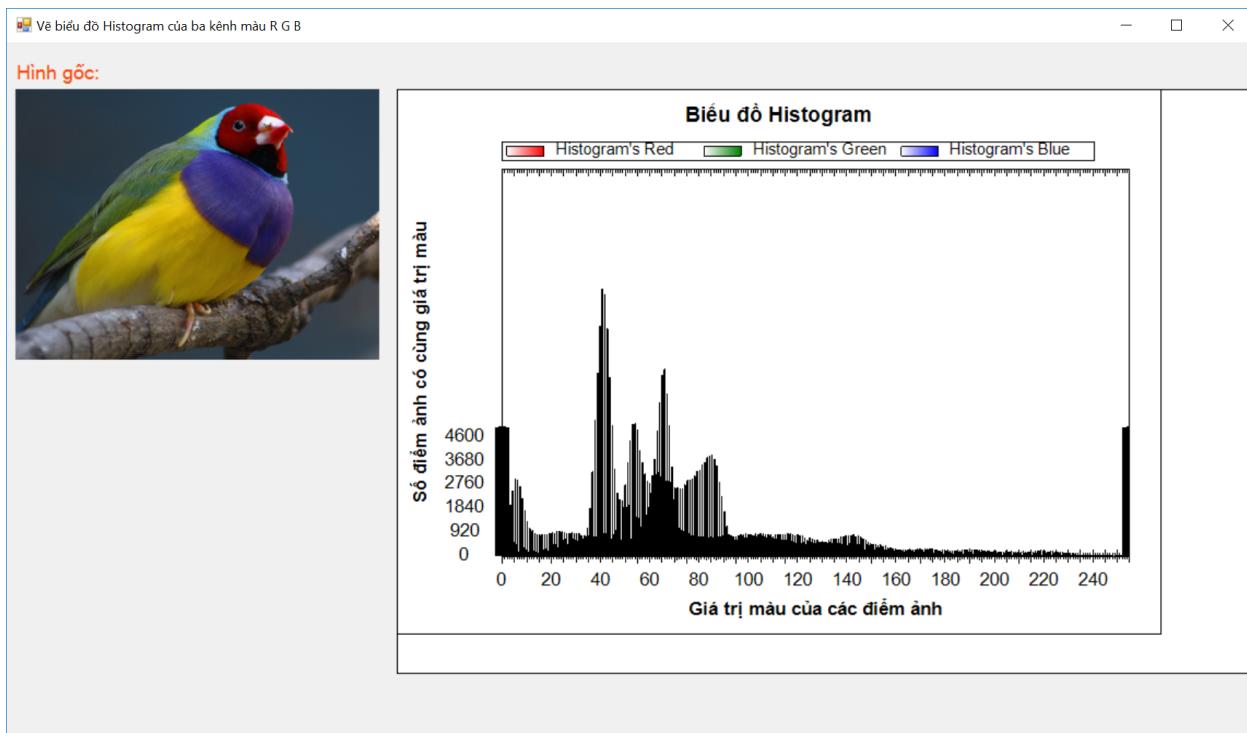
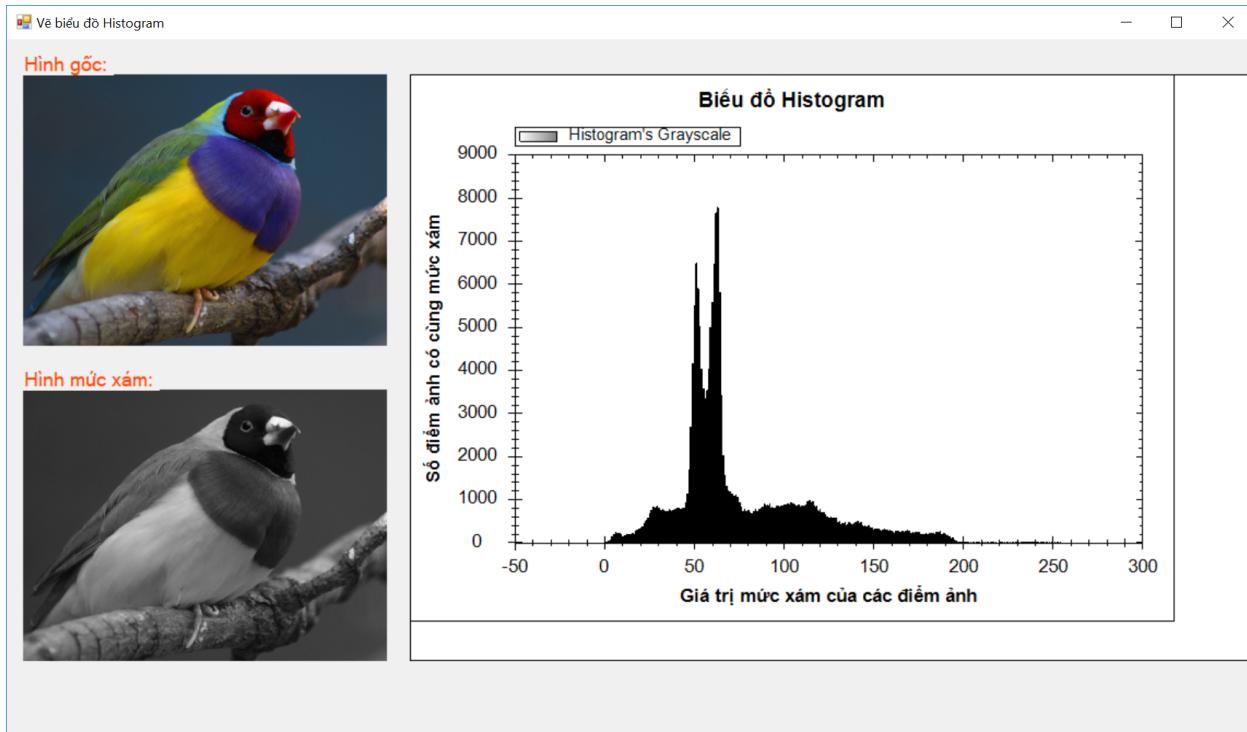
❤️ Videos bài giảng của thầy:

1. UTEX#11 - Thuật toán tính Histogram của ảnh
https://youtu.be/eoFS3K_6Cog
2. Xử lý ảnh #8 - Tính và vẽ biểu đồ Histogram của ảnh mức xám bằng C#.NET
<https://youtu.be/pSBeQ-luuDw>
3. Xử lý ảnh #9 - Tính và vẽ biểu đồ Histogram của các kênh màu R-G-B của ảnh màu RGB - Ngôn ngữ C#.NET
<https://youtu.be/xl6ascjwcNw>
4. UTEX#12 - Lập trình bằng Python tính và vẽ biểu đồ Histogram cho ảnh mức xám (Grayscale Image)
<https://youtu.be/KwN4zc4dJxQ>

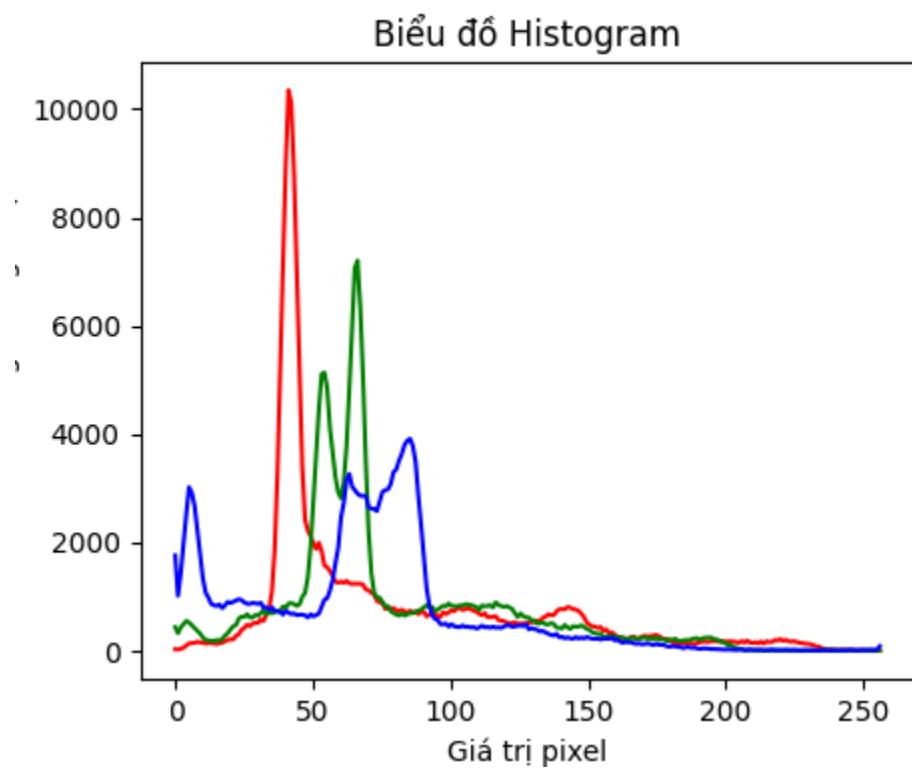
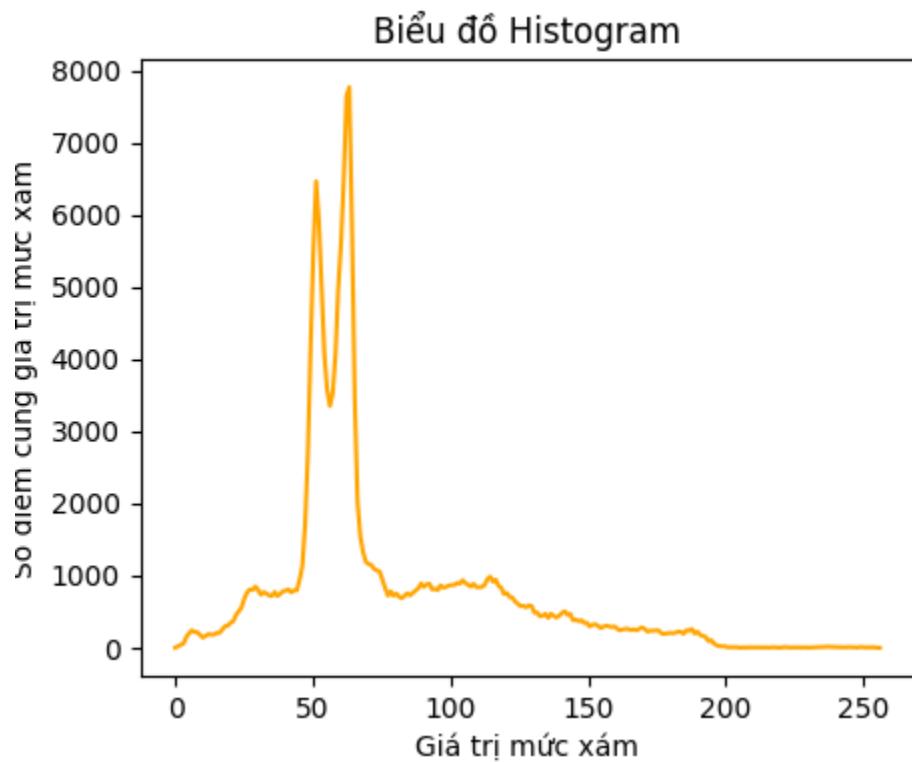
❤️ Một ví dụ minh họa tính biểu đồ histogram cho một kênh để các em dễ hiểu lý thuyết hơn nhé:



Kết quả C#.NET:



Kết quả Python:



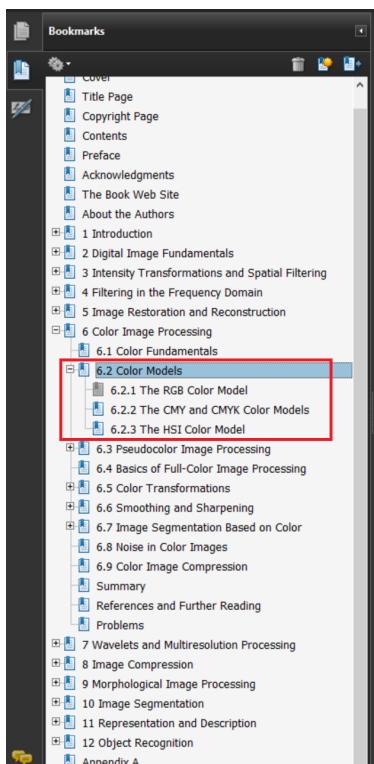
MINI-PROJECT 6:

- ⌚ Hiểu lý thuyết chuyển đổi không gian màu RGB sang không gian màu CMYK
- ⌚ Lập trình chuyển đổi không gian màu RGB sang không gian màu CMYK bằng C#.NET
- ⌚ Lập trình chuyển đổi không gian màu RGB sang không gian màu CMYK bằng Python

🌹 Videos bài giảng của thầy:

1. Xử lý ảnh #10 - Chuyển đổi không gian màu RGB sang CMYK - Ngôn ngữ C#.NET
<https://youtu.be/mASKnQQT24k>

🌹🌹 Các em đọc tài liệu trong cuốn sách thầy đã gửi cho các em trước đó tại các index này nhé, nhớ đọc cho kỹ lý thuyết vì thầy sẽ hỏi nhiều về lý thuyết xem các em có hiểu không nha:



printing is a combination of additive and subtractive color mixing, a process that is much more difficult to control than that of displaying colors on a monitor, which is based on the addition of three highly controllable light primaries.

6.2 Color Models

The purpose of a color model (also called *color space* or *color system*) is to facilitate the specification of colors in some standard, generally accepted way. In essence, a color model is a specification of a coordinate system and a subspace within that system where each color is represented by a single point.

Most color models in use today are oriented either toward hardware (such as for color monitors and printers) or toward applications where color manipulation is a goal (such as in the creation of color graphics for animation). In

Các em xem lý thuyết về các không gian màu tại mục 6.2 trang 401 trong sách nhé.

402 Chapter 6 ■ Color Image Processing

terms of digital image processing, the hardware-oriented models most commonly used in practice are the RGB (red, green, blue) mode and a broad class of color video cameras; the CMY (cyan, magenta, yellow) and CMYK (cyan, magenta, yellow, black) models for

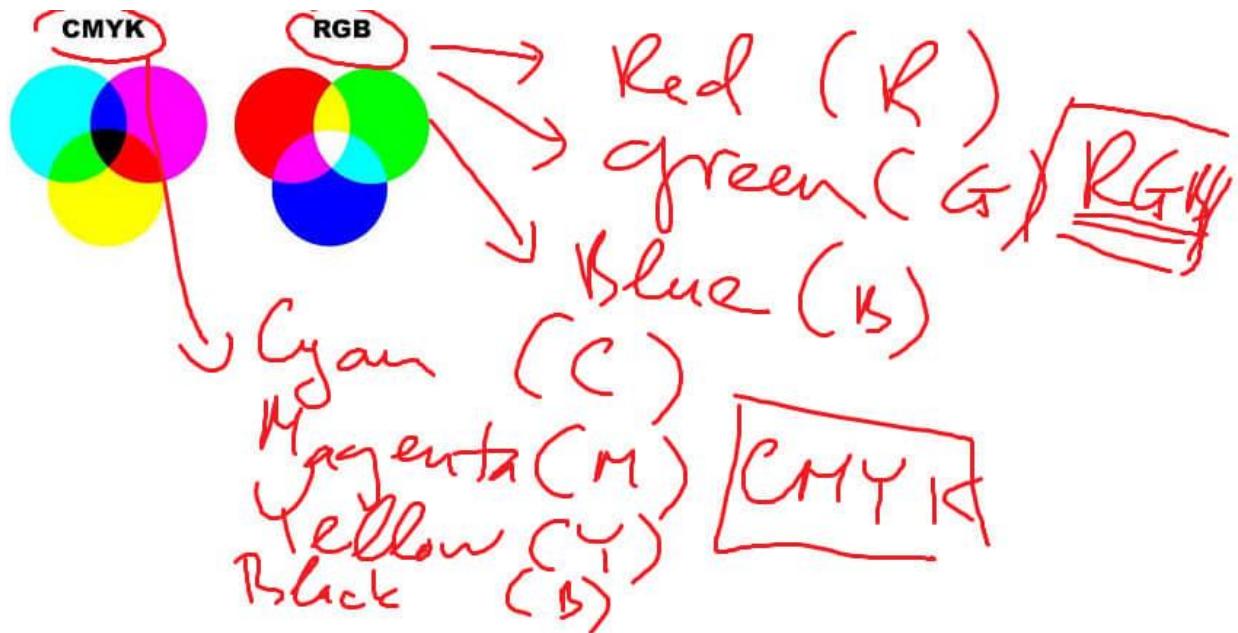
♥ Hai không gian màu cơ bản nhất đó là RGB tương ứng là Red-Green-Blue

Đối nghịch lại RGB là không gian màu CMYK, tương ứng là Cyan-Magenta-Yellow-Black

- Red: màu đỏ
- Green: màu xanh lá cây
- Blue: màu xanh nước biển
- Cyan: màu xanh dương (xanh da trời)
- Magenta: màu đỏ tươi
- Yellow: màu vàng
- Kênh K là màu Black, màu đen (đây có thể được coi là kênh màu mức xám của ảnh CMYK)

✿✿✿ Các em lưu ý việc tính chuyển đổi hệ màu RGB sang CMYK và ngược lại đơn giản là sự pha trộn màu của các kênh tương ứng, các em không cần phải dùng công thức tính gì cho nặng chương trình tính toán cũng như làm suy giảm giá trị màu sau mỗi lần chuyển đổi:

- Màu Cyan (xanh dương) là kết hợp giữa Green và Blue, vậy nên mình set kênh Red = 0
- Màu Magenta (tím) là kết hợp giữa Red và Blue, nên set kênh Green = 0
- Màu Yellow (vàng) là kết hợp giữa Red và Green, nên set kênh Blue = 0



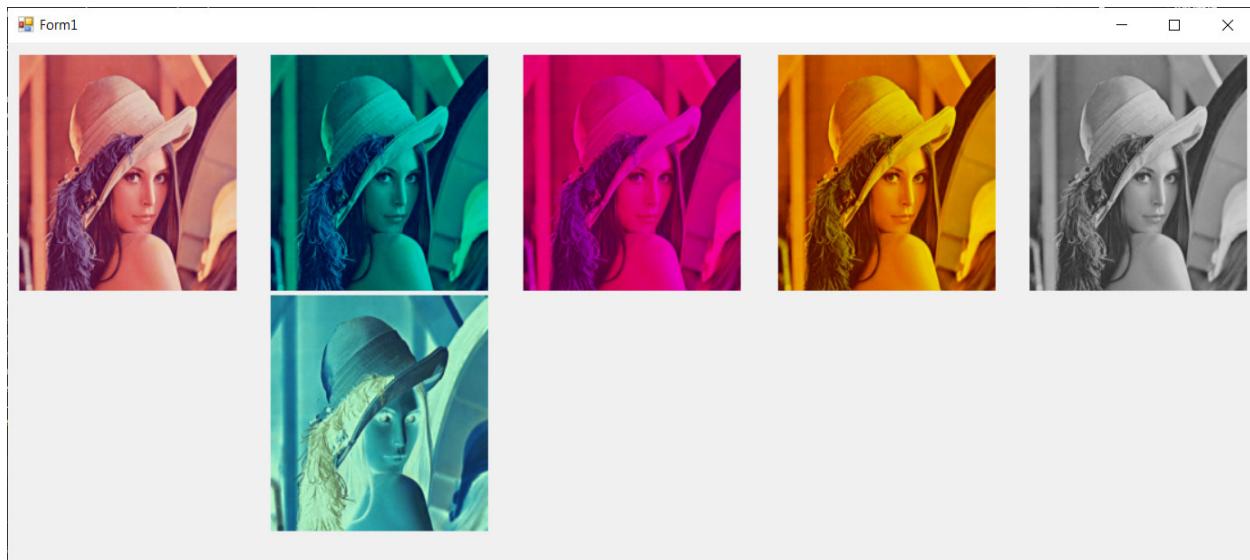
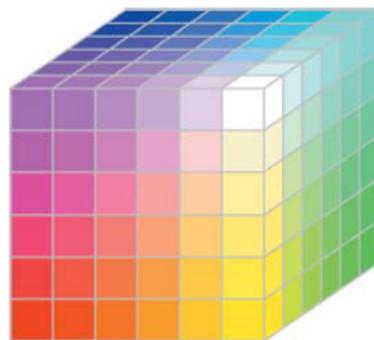
6.2.2 The CMY and CMYK Color Models

As indicated in Section 6.1, cyan, magenta, and yellow are the secondary colors of light or, alternatively, the primary colors of pigments. For example, when a surface coated with cyan pigment is illuminated with white light, no red light is reflected from the surface. That is, cyan subtracts red light from reflected white light, which itself is composed of equal amounts of red, green, and blue light.

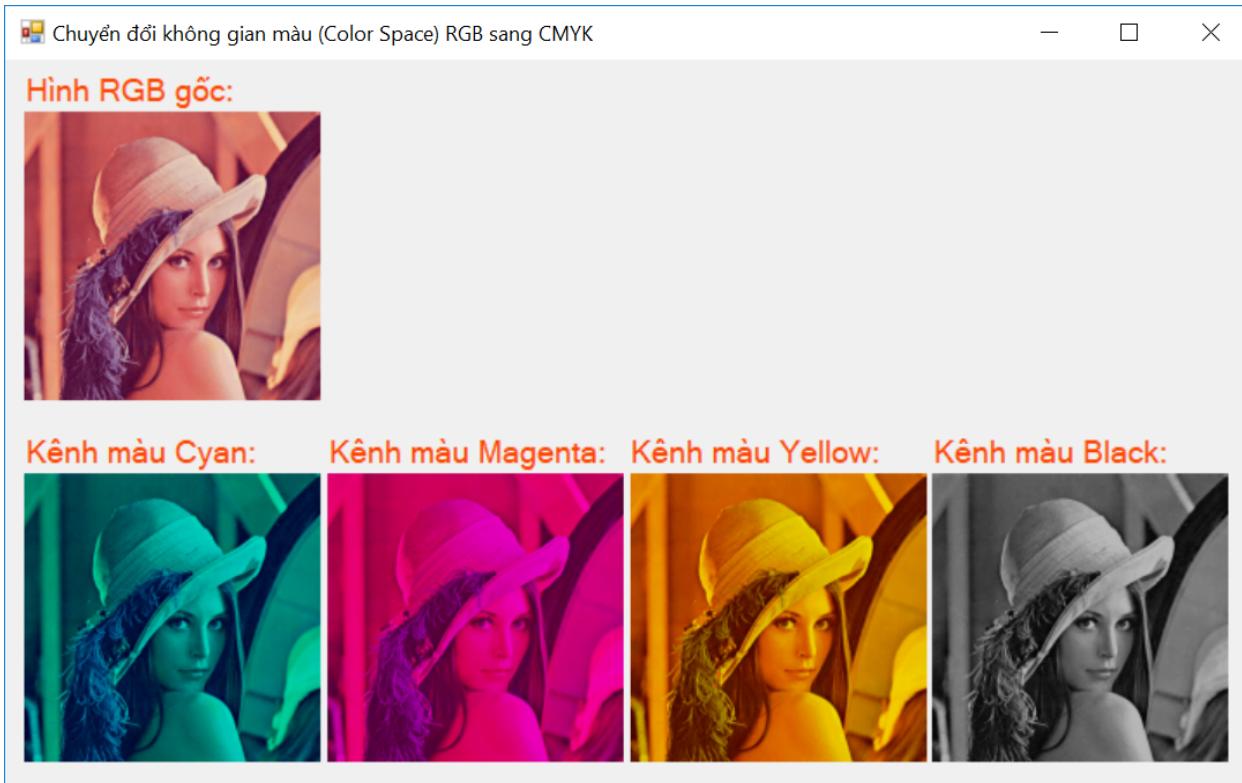
Most devices that deposit colored pigments on paper, such as color printers and copiers, require CMY data input or perform an RGB to CMY conversion internally. This conversion is performed using the simple operation

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (6.2-1)$$

where, again, the assumption is that all color values have been normalized to the range [0, 1]. Equation (6.2-1) demonstrates that light reflected from a



Kết quả C#.NET:



Kết quả Python:



MINI-PROJECT 7:

- ⌚ Hiểu lý thuyết chuyển đổi không gian màu RGB sang không gian màu HSI
- ⌚ Lập trình chuyển đổi không gian màu RGB sang không gian màu HSI bằng C#.NET
- ⌚ Lập trình chuyển đổi không gian màu RGB sang không gian màu HSI bằng Python

 Videos bài giảng của thầy:

1. UTEX#13 - Thuật toán chuyển đổi không gian màu RGB sang HSI
<https://youtu.be/QQsCtUft0IE>
2. Xử lý ảnh #11 - Chuyển đổi không gian màu RGB sang HSI
https://youtu.be/Nw_KQf3xSjE

 Thầy review lại phần chuyển đổi không gian màu cho các em nắm:

 TẠI SAO PHẢI CHYỂN ĐỔI KHÔNG GIAN MÀU RGB SANG CÁC KHÔNG GIAN MÀU KHÁC NHƯ HSI, HSV, CMYK, YCrCb, XYZ

Không gian màu RGB với mắt người thì rất dễ dàng để cảm nhận được đối tượng đó là màu gì, xanh hay đỏ hay tím hay vàng. Tuy nhiên, với máy tính thì nó sẽ không biết đó là màu xanh hay đỏ hay tím hay vàng, mà phải làm sao đó để biểu diễn màu sắc đó dưới một con số thì máy tính sẽ xử lý dễ hơn. Do vậy cần phải có các mô hình màu khác nhằm giúp máy tính làm được chuyện này.

 Một chú ý là các em thấy dù chuyển đổi sang bất kỳ không gian màu nào, ví dụ: HSI, HSV, CMYK, YCrCb hay XYZ thì trong các không gian màu đó luôn chứa một kênh giá trị mức xám (gray level). Ví dụ:

- HSI có kênh mức xám là I (Intensity) được tính theo công thức Average:
 $I = (R + G + B)/3$
- HSV có kênh mức xám là V (Value) được tính theo công thức MAX:
 $V = \text{Max}(R, G, B)$
- CMYK có kênh mức xám là K (Black) được tính theo công thức MIN:
 $K = \text{Min}(R, G, B)$
- YCrCb có kênh mức xám là Y được tính theo công thức Luminance:
 $Y = a*R + b*G + c*B$
- XYZ thậm chí chứa 3 kênh mức xám là X-Y-Z, mỗi kênh được tính dựa theo công thức Luminance, chỉ khác các hệ số alpha, beta, gama cho mỗi kênh là khác nhau:
 $X = a1*R + b1*G + c1*B$
 $Y = a2*R + b2*G + c2*B$
 $Z = a3*R + b3*G + c3*B$

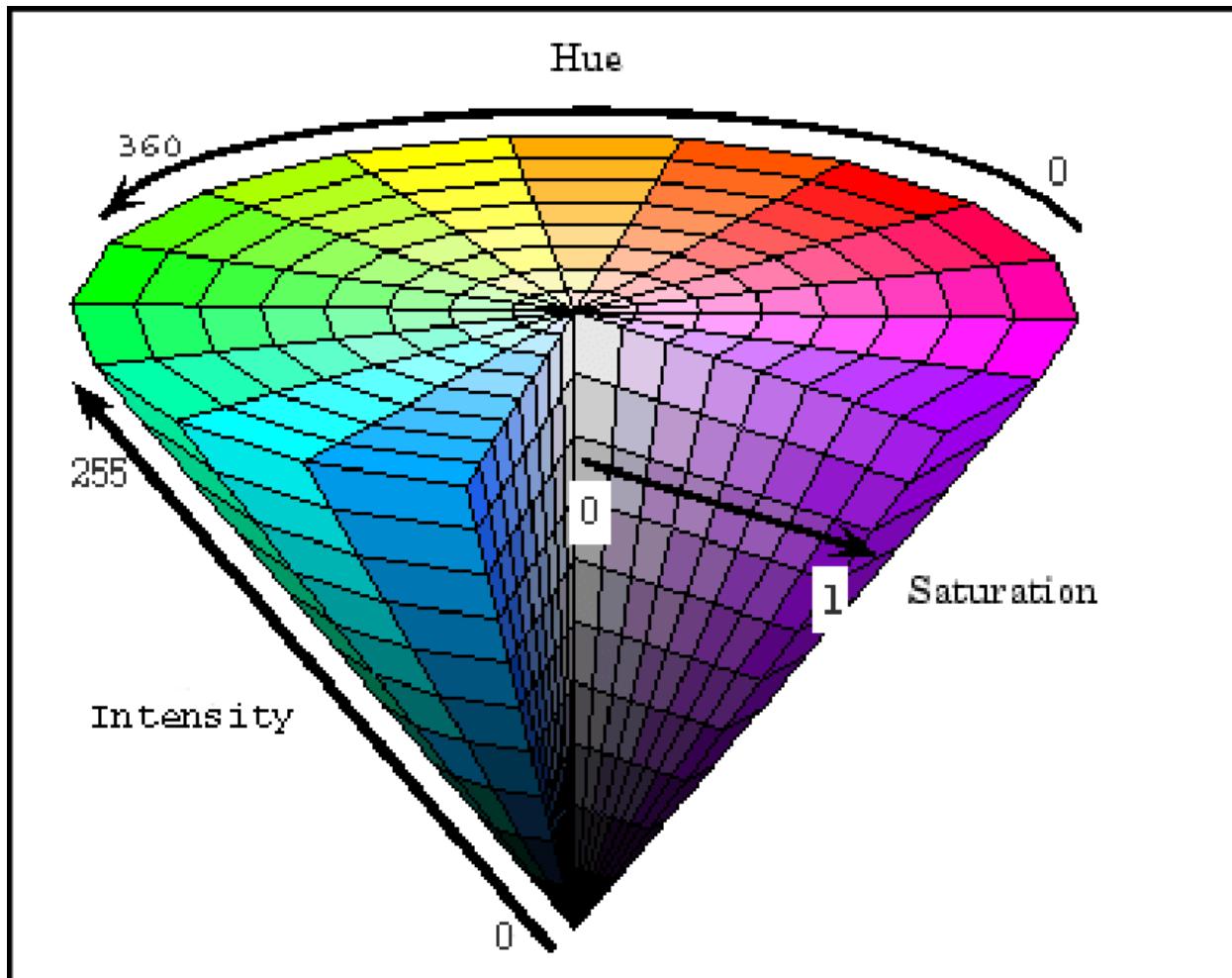
 Như vậy việc xử lý ảnh thực chất là mình phải xử lý trên kênh giá trị mức xám.

 MÔ HÌNH MÀU HSI (Như trong hình):

- **H (Hue)** là màu sắc. Lúc này màu sắc được biểu diễn bởi giá trị góc theta của một đường tròn lượng giác từ 0 đến 360 độ. Trong đường tròn này người ta chia ra từng múi, mỗi múi là một màu, ví dụ như trong hình thì tại múi đầu tiên ($\theta = 0$) thì con người nhìn vào sẽ biết ngay đó là màu tím, hoặc tại múi có $\theta = 180$ độ thì con người nhìn vào sẽ biết đó là màu green. Tuy nhiên, đối với máy tính thì nó chỉ quan tâm đến

con số theta = 0 hay theta = 180 để nó xử lý tính toán trên con số 0 hoặc 180 đó mà thôi, máy tính không biết đó tương ứng là màu tím hay màu green. Mọi xử lý của máy tính là trên con số, và như vậy việc tính toán sẽ trở nên dễ dàng hơn.

- **S (Saturation) là độ bão hòa màu.** Ví dụ, khi con người nhìn vào một đối tượng, mắt người sẽ nhanh chóng nhận biết đối tượng đó là màu gì, ví dụ là màu đỏ, đó là thông tin về HUE (màu sắc). Nếu cũng đối tượng màu đỏ đó được sơn với màu đậm hơn thì mắt người sẽ nhận biết đó là màu đỏ đậm, hoặc ngược lại nếu được sơn với màu nhạt thì mắt người sẽ nhận biết đó là màu đỏ nhạt. Đó chính là ý niệm về SATURATION. Cụ thể, trong hình mô hình màu HSI ở dưới, các em thấy S = [0, 1], ví dụ ngay tại mũi màu tím, nếu S = 1 thì con người hiểu đó là màu tím (bình thường), nếu S = 0.1 theo chiều hướng vào tâm thì con người hiểu đó là màu tím nhạt, cũng với S = 0.1 theo chiều hướng về đỉnh chóp hình nón thì con người hiểu là màu tím đậm (thậm chí tím đen đậm luôn). Tuy nhiên, với máy tính thì nó chỉ hiểu đó là con số 0.1 và nó chỉ xử lý trên con số 0.1 mà không quan tâm phải biết đó là màu tím, hay tím đậm, hay tím nhạt.
- **I (Intensity) là độ sáng tối (brightness), hay còn gọi là giá trị mức xám (gray level).** Đây chính là kênh gray level mà thầy đã đề cập ở trên, nó là kênh quan trọng vì mọi xử lý ảnh là xử lý trên gray level. Nó có giá trị nằm trong khoảng [0, 255]



Computations from RGB to HSI and back are carried out on a per-pixel basis. We omitted the dependence on (x, y) of the conversion equations for notational clarity.

Converting colors from RGB to HSI

Given an image in RGB color format, the H component of each RGB pixel is obtained using the equation

$$H = \begin{cases} \theta & \text{if } B \leq G \\ 360 - \theta & \text{if } B > G \end{cases} \quad (6.2-2)$$

Hue: màu sắc \rightarrow Độ số độ

Khi lập trình làm việc trong Python
để trả về radian \Rightarrow Dưới đây là cách convert
with[†]

$$\theta = \cos^{-1} \left\{ \frac{\frac{1}{2}[(R - G) + (R - B)]}{[(R - G)^2 + (R - B)(G - B)]^{1/2}} \right\}$$

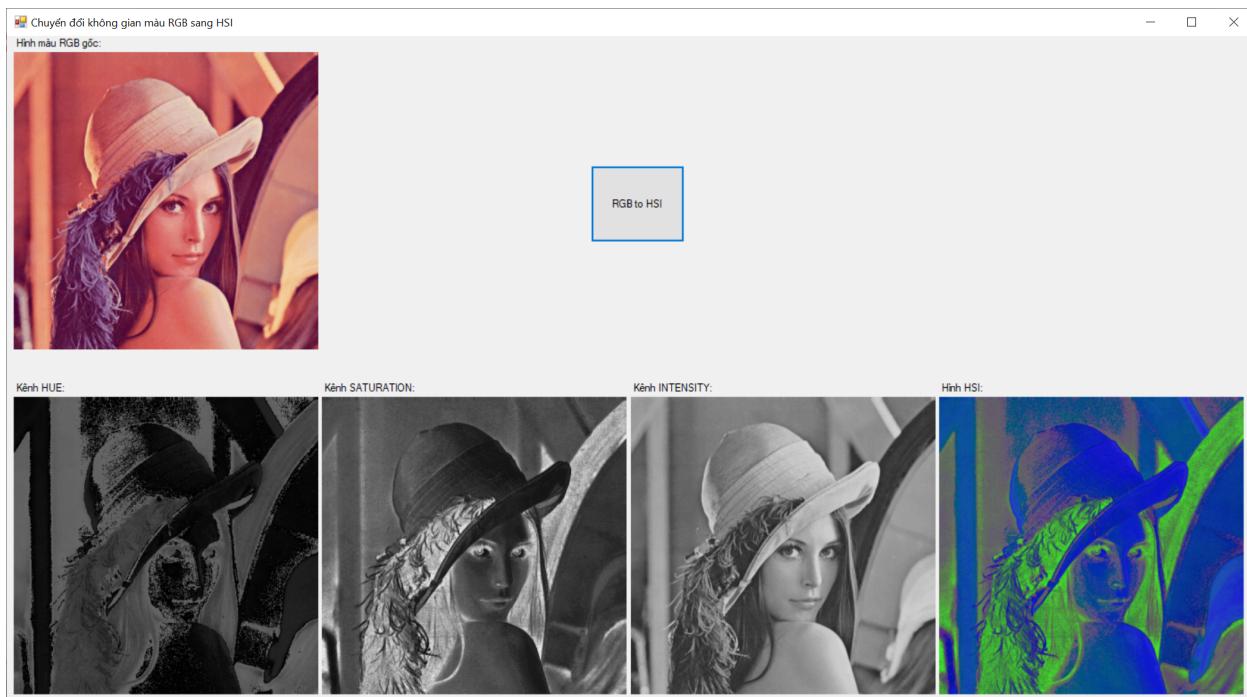
The saturation component is given by $S = [0, 1] \Rightarrow$ Cân phán normalize

$$S = 1 - \frac{3}{(R + G + B)} [\min(R, G, B)] \quad (6.2-3) \quad S = [0, 255]$$

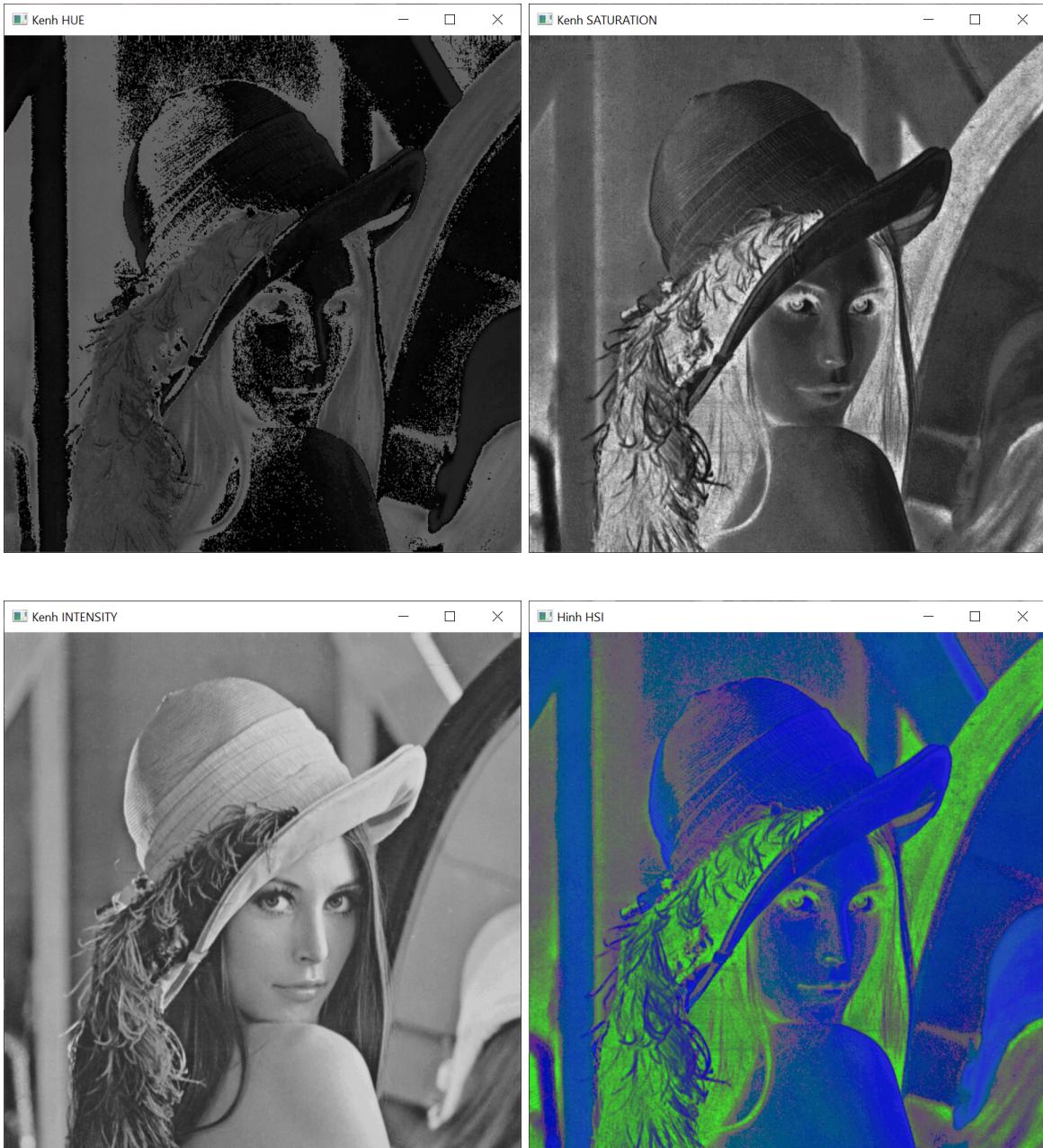
Finally, the intensity component is given by

$$I = \frac{1}{3}(R + G + B) \quad (6.2-4)$$

Kết quả C#.NET



Kết quả Python:



MINI-PROJECT 8:

- ⌚ Hiểu lý thuyết chuyển đổi không gian màu RGB sang không gian màu HSV
- ⌚ Lập trình chuyển đổi không gian màu RGB sang không gian màu HSV bằng C#.NET
- ⌚ Lập trình chuyển đổi không gian màu RGB sang không gian màu HSV bằng Python

Converting colors from RGB to ~~HSL~~ HSV

Computations from RGB to HSI and back are carried out on a per-pixel basis. We omitted the dependence on (x, y) of the conversion equations for notational clarity.

Given an image in RGB color format, the H component of each RGB pixel is obtained using the equation

$$H = \begin{cases} \theta & \text{if } B \leq G \\ 360 - \theta & \text{if } B > G \end{cases} \quad (6.2-2)$$

6.2 ■ Color Models 411

with[†]

$$\theta = \cos^{-1} \left\{ \frac{\frac{1}{2}[(R - G) + (R - B)]}{[(R - G)^2 + (R - B)(G - B)]^{1/2}} \right\}$$

The saturation component is given by

$$S = 1 - \frac{3}{(R + G + B)} [\min(R, G, B)] \quad (6.2-3)$$

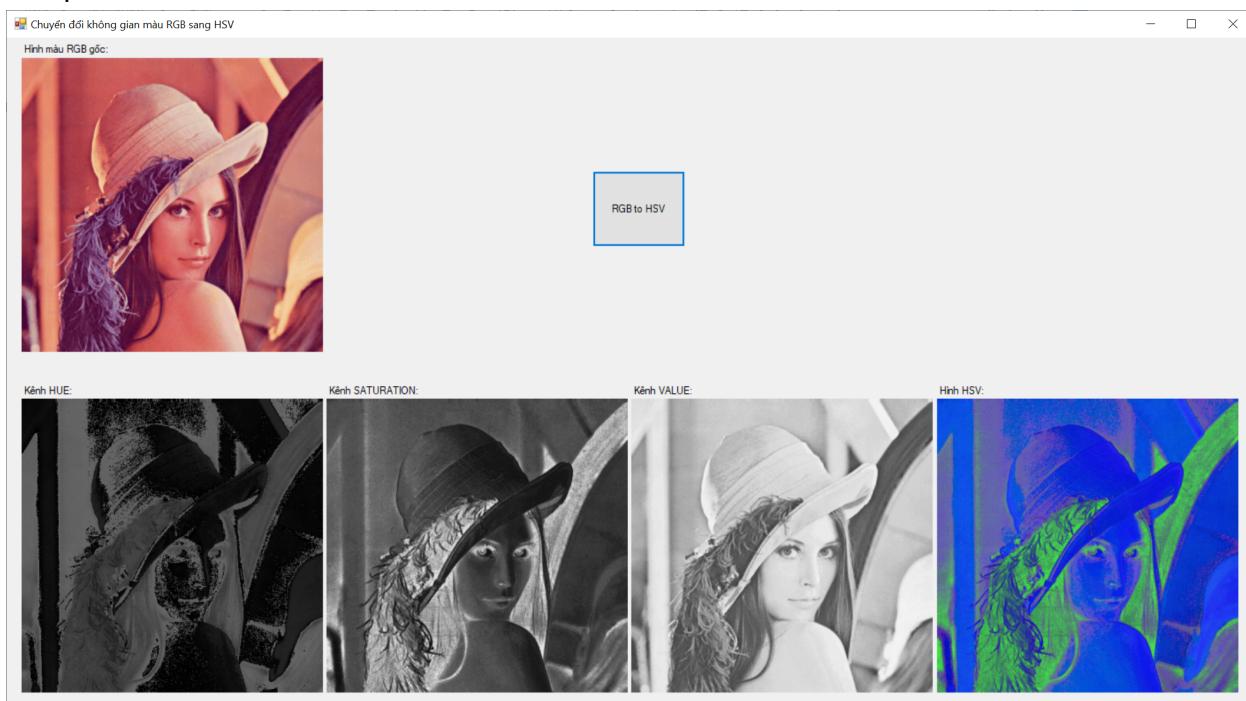
Finally, the ~~intensity~~ component is given by

$$I = \frac{1}{3}(R + G + B) \quad (6.2-4)$$

V: Value

$$V = \max(R, G, B)$$

Kết quả C#.NET:



Kết quả Python:



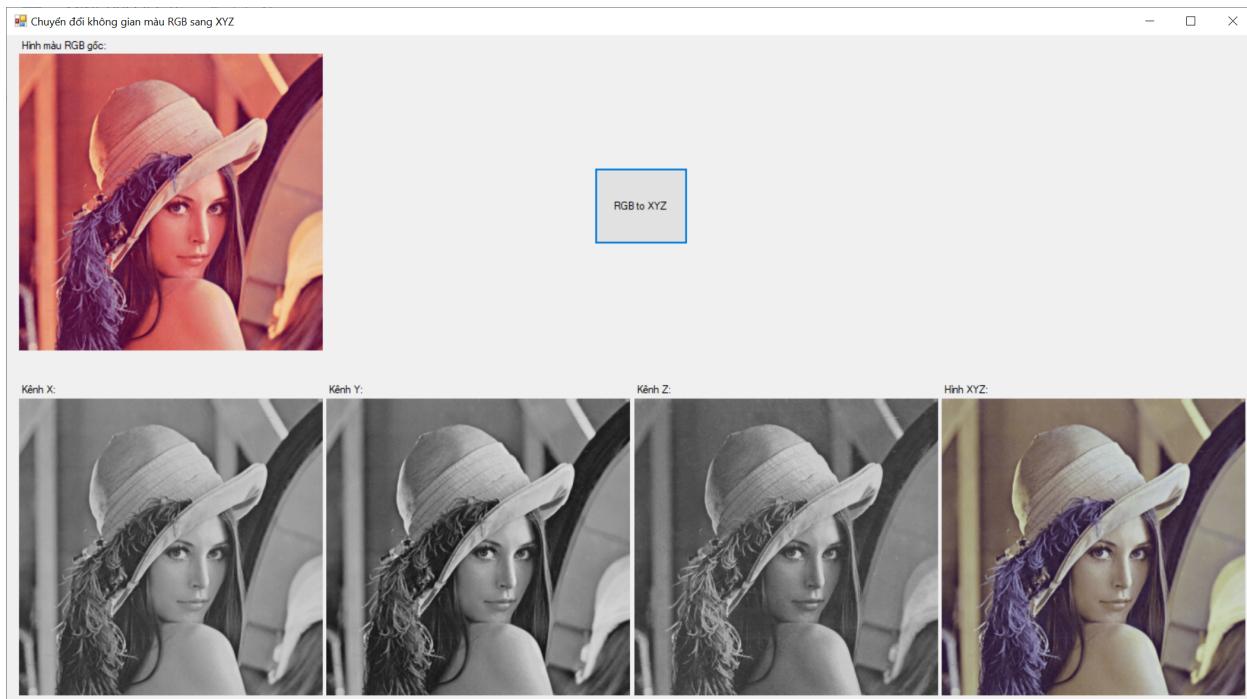
🔥 MINI-PROJECT 9:

- ⌚ Hiểu lý thuyết chuyển đổi không gian màu RGB sang không gian màu XYZ
- ⌚ Lập trình chuyển đổi không gian màu RGB sang không gian màu XYZ bằng C#.NET
- ⌚ Lập trình chuyển đổi không gian màu RGB sang không gian màu XYZ bằng Python

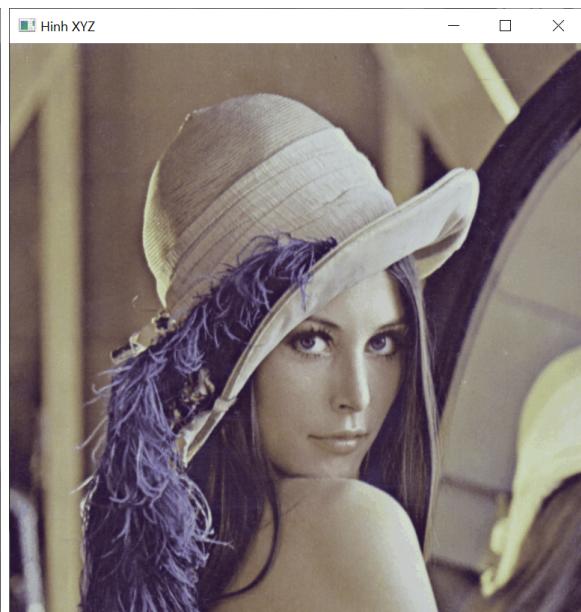
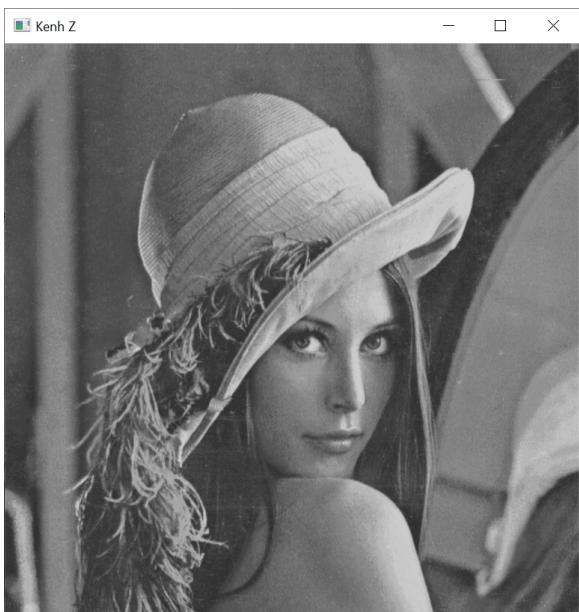
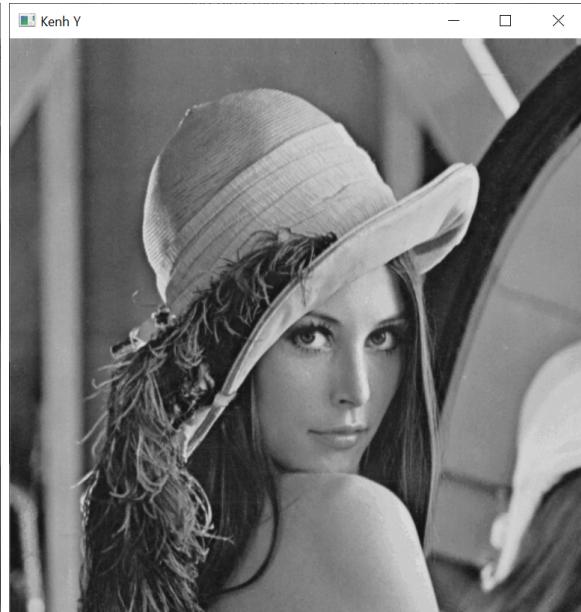
❤ Ma trận chuyển đổi không gian màu RGB sang XYZ:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Kết quả C#.NET:



Kết quả Python:



🔥 MINI-PROJECT 10:

- ⌚ Hiểu lý thuyết chuyển đổi không gian màu RGB sang không gian màu YCrCb
- ⌚ Lập trình chuyển đổi không gian màu RGB sang không gian màu YCrCb bằng C#.NET
- ⌚ Lập trình chuyển đổi không gian màu RGB sang không gian màu YCrCb bằng Python

❤️ Chuyển đổi không gian màu RGB sang YCrCb:

One important task in image processing applications is the color space conversion. Real-time images and videos are stored in RGB color space, because it is based on the sensitivity of color detection cells in the human visual system. In digital image processing the YCbCr color space is often used in order to take advantage of the lower resolution capability of the human visual system for color with respect to luminosity. Thus, RGB to YCbCr conversion is widely used in image and video processing [1].

Given a digital pixel represented in RGB format, 8 bits per sample, where 0 and 255 represents the black and white color, respectively, the YCbCr components can be obtained according to equations (1) to (3):

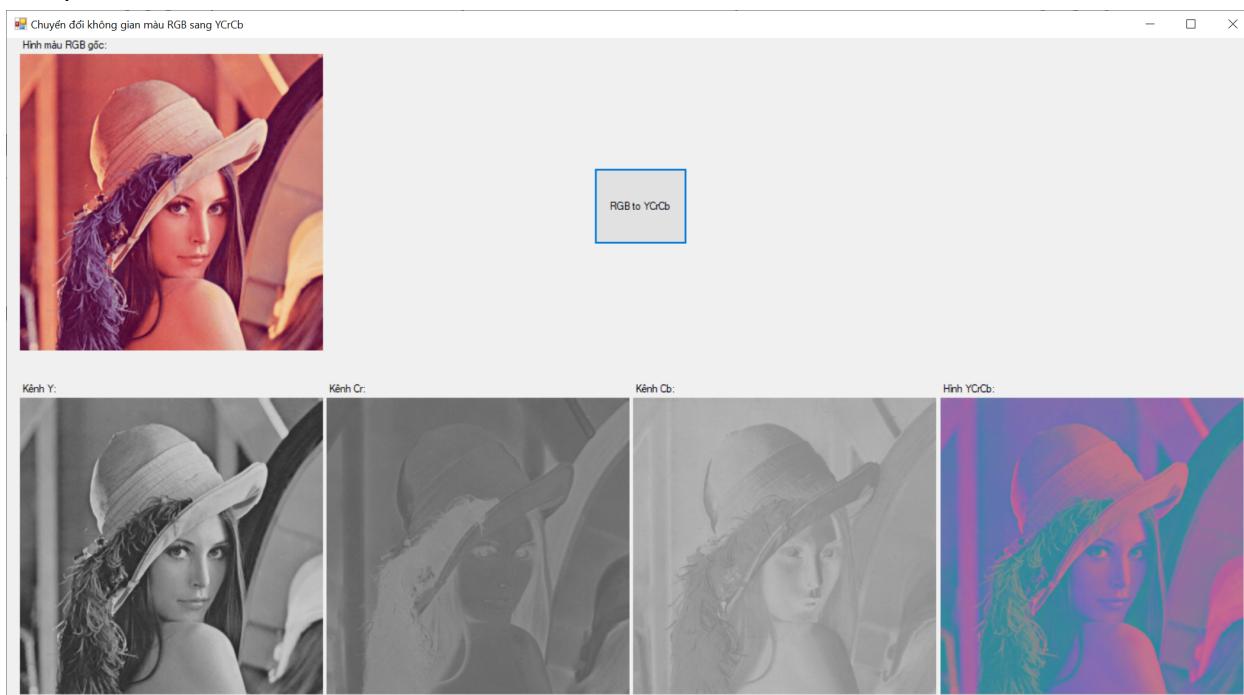
$$\left\{ \begin{array}{l} Y = 16 + \frac{65.738R}{256} + \frac{129.057G}{256} + \frac{25.064B}{256} \\ Cb = 128 - \frac{37.945R}{256} - \frac{74.494G}{256} + \frac{112.439B}{256} \\ Cr = 128 + \frac{112.439R}{256} - \frac{94.154G}{256} - \frac{18.285B}{256} \end{array} \right. \quad (1)$$

$$(2)$$

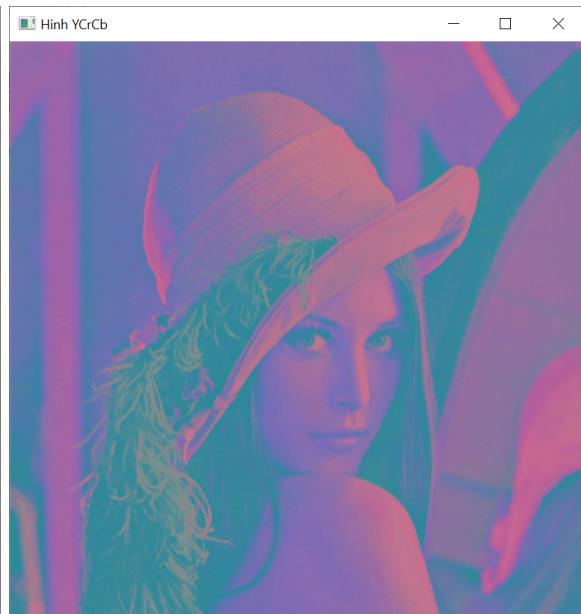
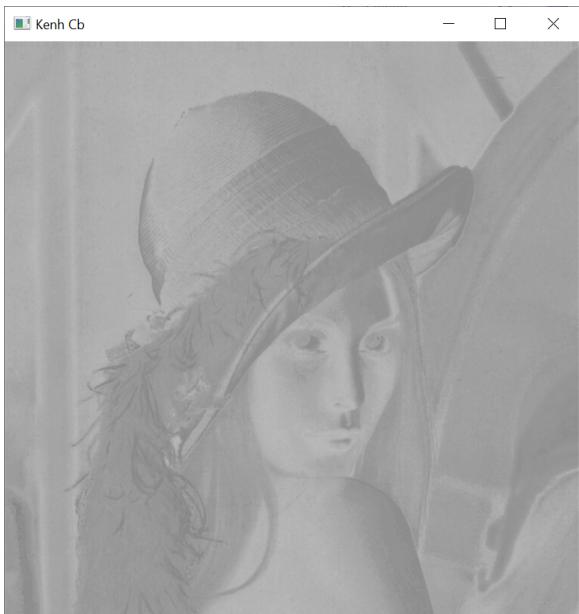
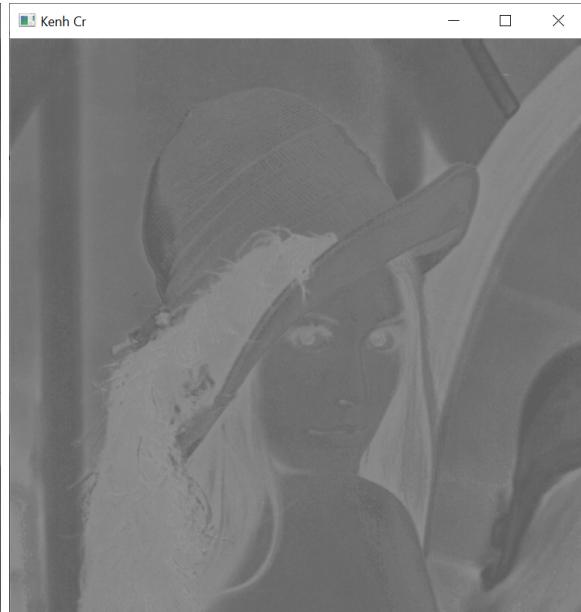
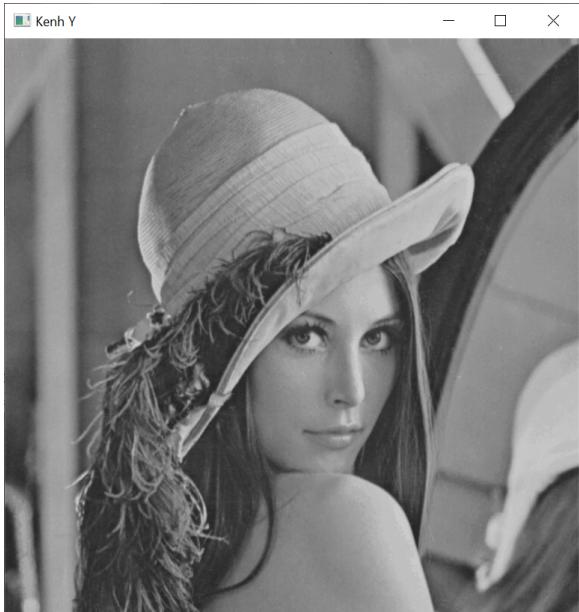
$$(3)$$

Equations (1) to (3): RGB to YCbCr conversion, source: [1]

Kết quả C#.NET:



Kết quả Python:



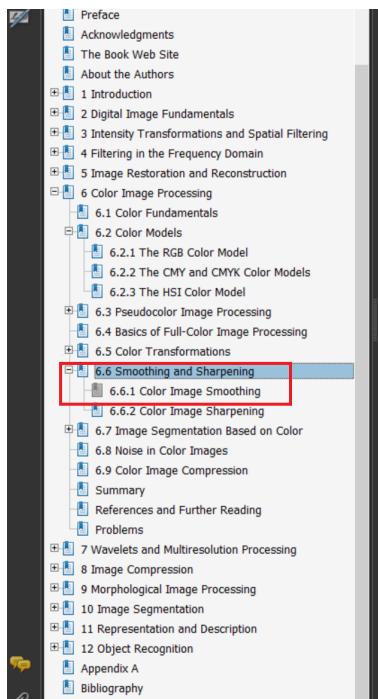
MINI-PROJECT 11:

ribbon Color Image Smoothing - Làm mượt ảnh màu ribbon

ribbon👉👉 Tạo các mặt nạ (mask) với các kích thước khác nhau: (3x3), (5x5), (7x7) và (9x9) để làm mượt (smoothing) hình Lena 512x512 24bit màu.

- 🎯 Hiểu lý thuyết làm mượt ảnh màu (color image smoothing)
- 🎯 Lập trình làm mượt ảnh màu với các mặt nạ yêu cầu ở trên bằng C#.NET
- 🎯 Lập trình làm mượt ảnh màu với các mặt nạ yêu cầu ở trên bằng Python

🌹VN Các em đọc tài liệu trong cuốn sách thầy đã gửi cho các em trước đó tại các index này nhé, nhớ đọc cho kỹ lý thuyết vì thầy sẽ hỏi nhiều về lý thuyết xem các em có hiểu không nha:



6.6 Smoothing and Sharpening

The next step beyond transforming each pixel of a color image without regard to its neighbors (as in the previous section) is to modify its value based on the characteristics of the surrounding pixels. In this section, the basics of this type of neighborhood processing are illustrated within the context of color image smoothing and sharpening.

6.6.1 Color Image Smoothing

With reference to Fig. 6.29(a) and the discussion in Sections 3.4 and 3.5, grayscale image smoothing can be viewed as a spatial filtering operation in which the coefficients of the filtering mask have the same value. As the mask is slid across the image to be smoothed, each pixel is replaced by the average of the pixels in the neighborhood defined by the mask. As can be seen in Fig. 6.29(b), this concept is easily extended to the processing of full-color images. The principal difference is that instead of scalar intensity values we must deal with component vectors of the form given in Eq. (6.4-2).

Let S_{xy} denote the set of coordinates defining a neighborhood centered at (x, y) in an RGB color image. The average of the RGB component vectors in this neighborhood is

$$\bar{\mathbf{c}}(x, y) = \frac{1}{K} \sum_{(s,t) \in S_{xy}} \mathbf{c}(s, t) \quad (6.6-1)$$

❤ Videos bài giảng của thầy:

1. UTEX#14 - Thuật toán làm mượt ảnh màu (Color Image Smoothing)
<https://youtu.be/PVUYINylmiY>
2. Xử lý ảnh #12 - Lập trình thuật toán làm mượt ảnh màu RGB (Color Image Smoothing) -
 Ngôn ngữ C#.NET
<https://youtu.be/iezBTLrvq8U>

❤ Công thức 6.6-1 là công thức tổng quát để làm mượt (hay còn gọi là làm mờ - blur) một điểm ảnh.

❤ Công thức 6.6-2 là công thức làm mượt điểm ảnh màu RGB trên cả 3 kênh màu R-G-B

Let S_{xy} denote the set of coordinates defining a neighborhood centered at (x, y) in an RGB color image. The average of the RGB component vectors in this neighborhood is

$$\bar{\mathbf{c}}(x, y) = \frac{1}{K} \sum_{(s, t) \in S_{xy}} \mathbf{c}(s, t)$$

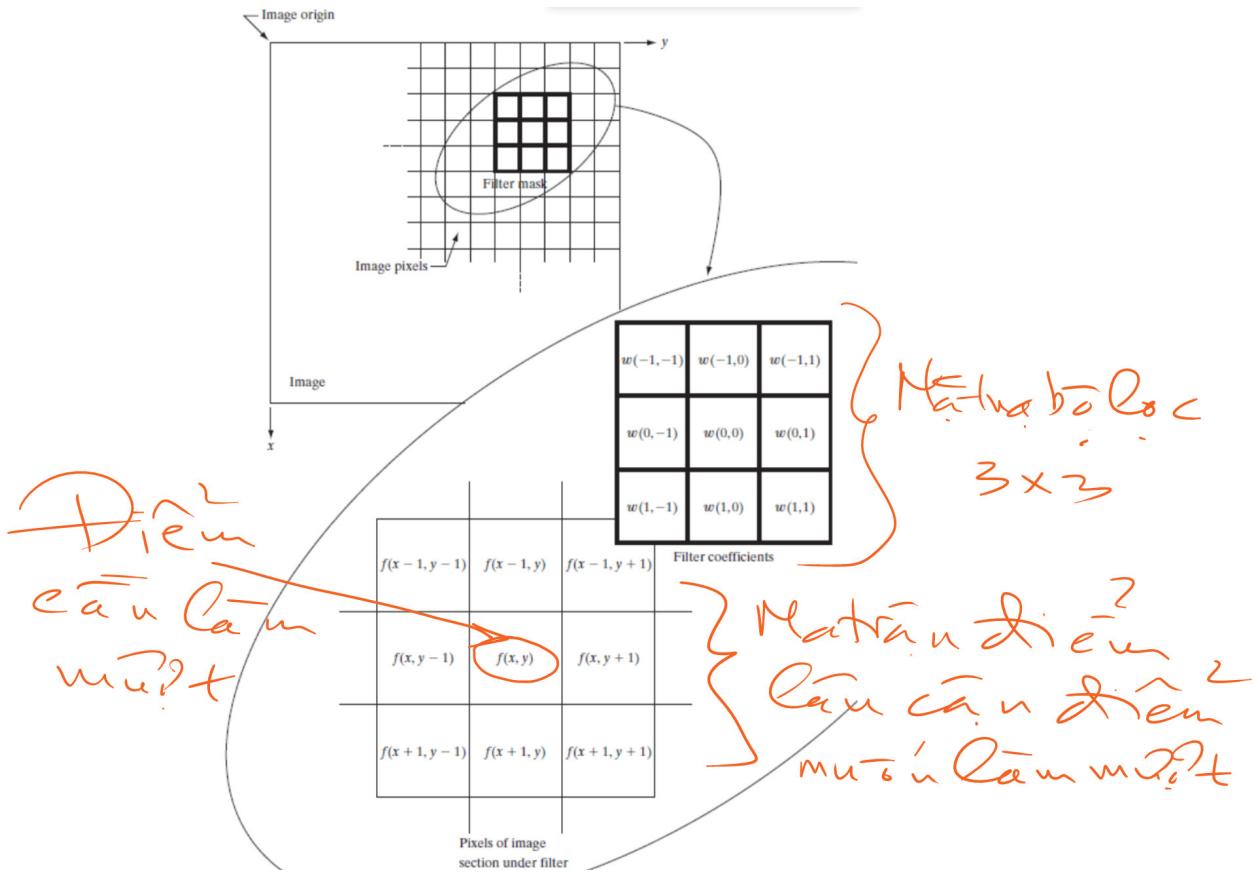
It follows from Eq. (6.4-2) and the properties of vector addition that

$$\mathbf{c}(x, y) = \begin{bmatrix} \frac{1}{K} \sum_{(s, t) \in S_{xy}} R(s, t) \\ \frac{1}{K} \sum_{(s, t) \in S_{xy}} G(s, t) \\ \frac{1}{K} \sum_{(s, t) \in S_{xy}} B(s, t) \end{bmatrix} \quad (6.6-2)$$

$\bar{\mathbf{c}}(x, y) : R G B$
sau smoothing

❤ Tham khảo tài liệu Digital Image Processing 3rd Edition trang 461, công thức 6.6-1

🎀 THẦY GIẢI THÍCH LÝ THUYẾT LÀM MƯỢT ẢNH MÀU RGB THEO HÌNH CHỤP BẰNG BÀI GIẢNG CỦA THẦY TRÊN LỚP ĐỂ CÁC EM NẮM.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
0	13	21	25	34	45	6	7	15	17	78	98	78	45	76	112	45	134	234	255	7	
1	2	4	0	23	31	32	54	76	87	98	12	43	45	76	98	45	123	36	252	231	
2	90	23	34	45	67	56	0	0	87	0	54	2	45	65	78	98	4	56	78	23	
3	12	43	54	67	23	45	34	54	65	76	87	98	90	12	32	43	24	54	65	76	
4	23	43	45	65	76	65	58	76	54	98	123	143	231	150	254	222	21	32	45	65	
5	43	54	65	76	23	54	56	43	54	56	76	43	5	76	9	12	43	65	7		
6	25	13	32	23	34	43	45	54	58	70	78	98	18	17	8	4	3	5	7	0	
7	36	2	3	5	7	8	9	1	2	3	4	6	2	4	43	65	77	78	143	155	
8	75	23	43	23	43	54	98	65	67	12	5	234	54	54	23	54	75	43	34	43	
9	81	54	65	23	4	6	8	6	8	108	34	65	23	86	2	43	2	65	4	12	43
10	89	34	54	65	43	34	34	2	3	5	7	4	54	34	43	34	87	98	0	21	
11	25	54	65	76	34	76	24	6	34	6	34	65	32	65	2	27	37	84	5	63	
12	123	34	564	65	34	43	54	34	43	43	54	34	54	44	34	76	67	8	92	5	
13	34	65	78	12	4	5	9	14	5	34	6	3	4	5	4	65	76	76	76		
14	23	54	76	89	65	21	34	65	76	87	98	14	15	17	65	85	97	200	237	98	
15	43	54	65	76	2	43	54	65	7	8	9	2	7	65	78	37	98	2	34	65	
16	65	0	2	0	1	6	8	9	0	34	56	111	21	145	24	235	22	21	87	43	
17	234	54	65	34	232	43	54	7	76	45	34	65	87	4	32	5	565	23	4	5	
18	6	32	54	65	6	34	65	43	54	85	76	3	54	76	23	54	65	76	87	2	
19	43	43	54	65	76	86	87	23	54	65	75	87	34	54	6	34	34	34	34	34	

Size : $3 \times 3 = 9$

$$20 \times 20 \text{ pixels} : 0 \div 255 \text{ byte}$$

smoothing

$$\underbrace{c(x, y)}_{\text{pixel gốc}} \rightarrow \underbrace{\bar{c}(x, y)}_{\text{pixel}}$$

$$\bar{c}(x, y) = \frac{1}{K} \sum_{(s, t) \in S_{xy}} c(s, t) \quad \text{(6.6-1)}$$

Sum smoothing

$$c(8, 7) \rightarrow \bar{c}(8, 7) ?$$

$$\bar{c} = \left(\frac{65 + 87 + 12 + 8 + 58 + 34 + 2 + 65 + 7}{9} \right) = \boxed{X}$$

$$\bar{c}(8, 7) = \frac{\bar{c}}{9}$$

- Theo công thức 6.6-1 trang 461 thì giá trị đã làm mượt (smoothing) của điểm ảnh RGB tại vị trí (x, y) được ký hiệu là $\bar{c}(x, y)$ có cái gạch ngang trên đầu chữ c . Ví dụ trong hình chụp bảng của thầy thì điểm ảnh đang xét trong hình RGB có vị trí là $(8, 9)$.
- Nếu mình dùng mặt nạ size 3×3 thì $K = 3 \times 3 = 9$

- xy là tất cả các điểm ảnh RGB trong một mặt nạ 3×3 với tâm mặt nạ là điểm ảnh RGB tại (x, y) , theo ví dụ trong hình là điểm $(8, 9)$. Như vậy tất cả điểm ảnh RGB thuộc mặt nạ này là:

$$\begin{array}{lll} c(7, 8) & c(8, 8) & c(9, 8) \\ c(7, 9) & c(8, 9) & c(9, 9) \\ c(7, 10) & c(8, 10) & c(9, 10) \end{array}$$

* Như vậy, điểm ảnh đã làm mượt (smoothed pixel) tại vị trí $(8, 9)$ sẽ được tính như sau:

$$c_{\text{gạch}}(8, 9) = (\text{Tổng các điểm ảnh trong mặt nạ}) / 9$$

Nhịn 3x3 theo cách
lên mực định
+ lèn

$$\frac{1}{9} \times \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Ví dụ cho điểm cần làm mượt tại vị trí $(2, 2)$:

Điểm
vốn
lên
mực

	0	1	2	3	4	5
0	13	21	25	34	45	6
1	2	4	0	23	31	32
2	90	23	34	45	67	58
3	12	43	54	67	23	45
4	23	43	45	65	76	65
5	43	54	65	76	23	55

Làm mượt theo phương pháp Average:

① Làm mượt + đón giao?

$$\bar{c}(2,2) = \frac{1}{9} \times \left(\begin{array}{ccc} 4 & 0 & 23 \\ 23 & 34 & 45 \\ 43 & 54 & 67 \end{array} \right) \quad \text{(X)}$$

$$= \frac{293}{9} = 32.55 \approx \boxed{32}$$

Điểm sau là làm
mượt + đón giao?

⚠ Một ví dụ tính điểm ảnh sau làm mượt với mặt nạ 5x5:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	13	21	25	34	45	6	7	15	17	78	98	76	45	76	112	45	134	255	7
1	2	4	0	23	31	32	54	78	87	98	12	43	45	78	98	45	123	36	252
2	90	23	34	45	67	56	0	0	67	0	54	2	45	65	78	98	4	56	78
3	12	43	54	67	23	45	34	54	65	76	87	98	90	12	32	43	24	54	65
4	23	43	45	05	76	85	56	76	54	98	123	143	231	150	254	222	21	32	45
5	43	54	65	76	23	54	56	43	43	54	56	78	43	5	78	9	12	43	65
6	25	13	32	23	34	43	45	54	56	76	78	98	18	17	8	4	3	5	7
7	36	2	3	5	7	8	9	1	2	3	4	6	2	4	43	65	77	78	143
8	75	23	43	23	43	54	98	65	87	12	5	234	54	54	23	54	75	43	34
9	81	54	65	23	4	6	8	8	98	34	65	23	86	2	43	2	65	4	12
10	89	34	54	65	43	34	34	2	65	7	4	54	34	43	32	76	87	98	0
11	25	54	65	76	34	76	24	6	34	6	34	65	32	65	2	27	37	84	5
12	123	34	564	65	34	43	54	34	43	43	54	34	54	44	34	76	87	3	92
13	34	65	78	12	4	5	9	14	5	34	6	3	4	4	65	34	5	4	65
14	23	54	76	89	65	21	34	65	76	87	98	14	15	17	65	85	97	200	237
15	43	54	65	78	2	43	54	65	7	8	9	2	7	65	78	87	98	23	34
16	65	0	2	0	1	6	8	9	0	34	56	111	21	145	234	235	22	21	87
17	234	54	65	34	232	43	54	7	76	45	34	65	87	4	32	5	565	23	4
18	6	32	54	65	6	34	65	43	54	65	76	3	54	76	23	54	65	76	87
19	43	43	54	65	76	86	87	23	54	65	75	87	34	54	6	34	34	34	34

Mặt nạ (mask) (5×5)

$$c(4,4) = 76$$

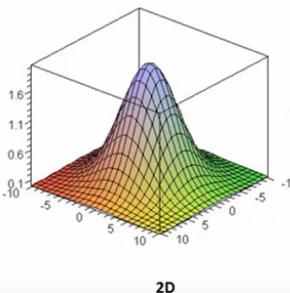
$$\Rightarrow \bar{c}(4,4) = \overline{\sum (5 \times 5)}$$

$$= \overline{34 + 45 + 67 + 56 + 0 + 54 + 67 + 23 + 45 + 34 + 45 + 65 + 76 + 55 + 56 + 65 + 76 + 23 + 54 + 56 + 32 + 23 + 34 + 43 + 45 + 56 + 65 + 76 + 87 + 98 + 0 + 21} / 25$$

$$\Rightarrow \boxed{\bar{c}(4,4)}$$

Thiệt toán làm mờ ảnh Gaussian Blur

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



In some applications, we have a continuous function of two variables, and the objective is to obtain a spatial filter mask based on that function. For example, a Gaussian function of two variables has the basic form

$$h(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where σ is the standard deviation and, as usual, we assume that coordinates x and y are integers. To generate, say, a 3×3 filter mask from this function, we

Hàm Gaussian

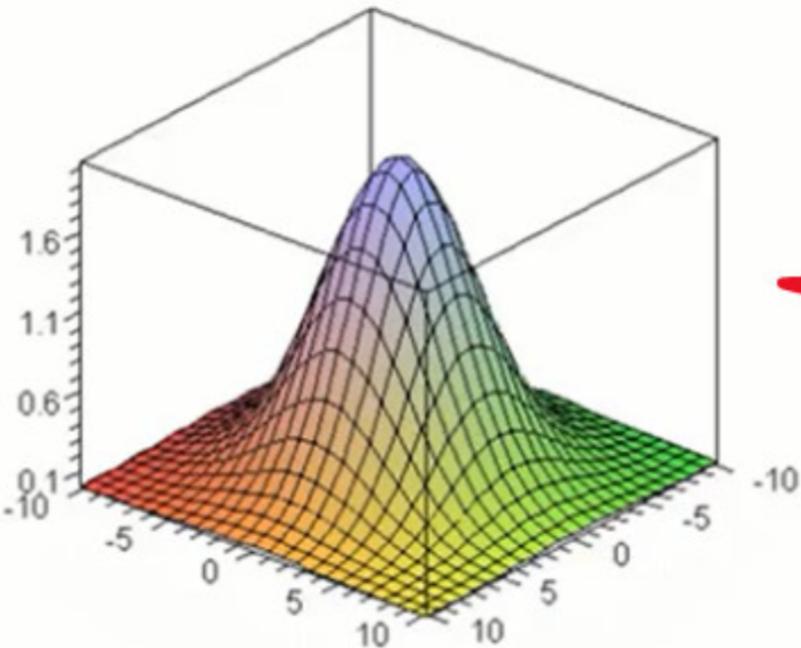
2 chiều

152 Chapter 3 ■ Intensity Transformations and Spatial Filtering

sample it about its center. Thus, $w_1 = h(-1, -1), w_2 = h(-1, 0), \dots, w_9 = h(1, 1)$. An $m \times n$ filter mask is generated in a similar manner. Recall that a 2-D Gaussian function has a bell shape, and that the standard deviation controls the “tightness” of the bell.

Generating a *nonlinear* filter requires that we specify the size of a neighborhood and the operation(s) to be performed on the image pixels contained in the neighborhood. For example, recalling that the max operation is nonlinear (see Section 2.6.2), a 5×5 max filter centered at an arbitrary point (x, y) of an image obtains the maximum intensity value of the 25 pixels and assigns that value to location (x, y) in the processed image. Nonlinear filters are quite

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



2D

Xem trọng
sắc thấy
đi?

Giống hinh cái chúng
vừa xem

In some applications, we have a continuous function of two variables, and the objective is to obtain a spatial filter mask based on that function. For example, a Gaussian function of two variables has the basic form

$$h(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where σ is the standard deviation and, as usual, we assume that coordinates x and y are integers. To generate, say, a 3×3 filter mask from this function, we

152 Chapter 3 ■ Intensity Transformations and Spatial Filtering

sample it about its center. Thus, $w_1 = h(-1, -1), w_2 = h(-1, 0), \dots, w_9 = h(1, 1)$. An $m \times n$ filter mask is generated in a similar manner. Recall that a 2-D Gaussian function has a bell shape, and that the standard deviation controls the “tightness” of the bell.

In some applications, we have a continuous function of two variables, and the objective is to obtain a spatial filter mask based on that function. For example, a Gaussian function of two variables has the basic form

$$h(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where σ is the standard deviation and, as usual, we assume that coordinates x and y are integers. To generate, say, a 3×3 filter mask from this function, we

$$G = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

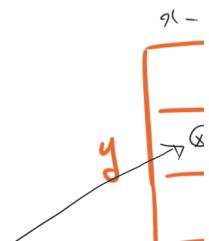
Hess'

152 Chapter 3 ■ Intensity Transformations and Spatial Filtering

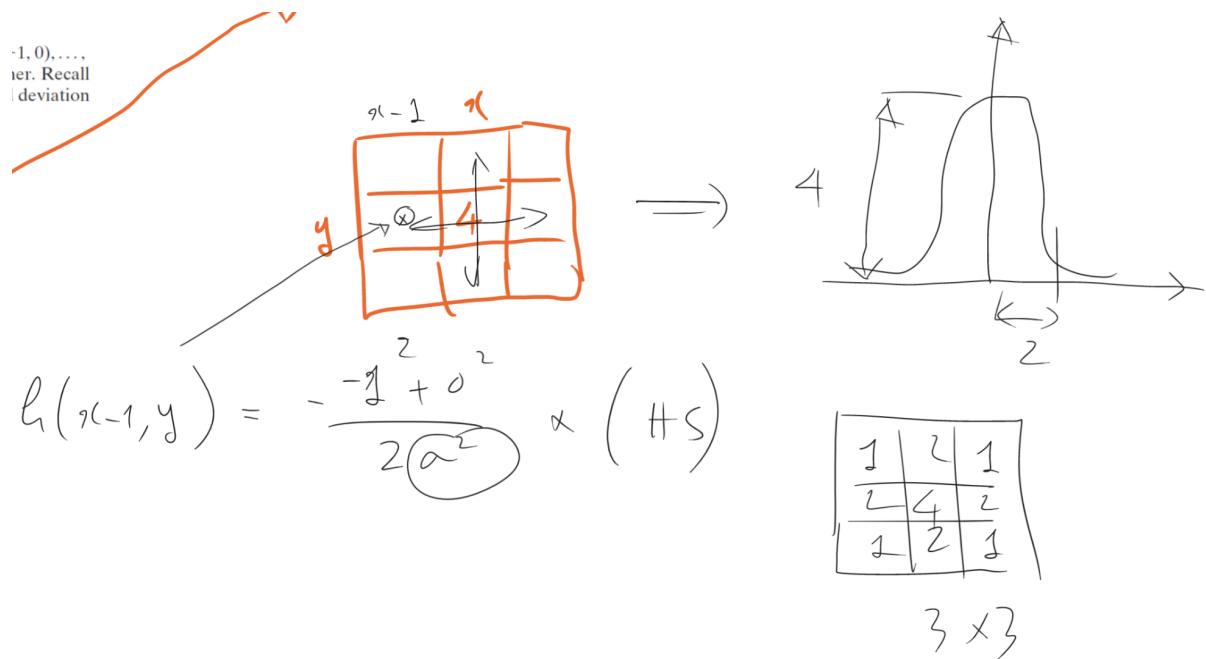
sample it about its center. Thus, $w_1 = h(-1, -1), w_2 = h(-1, 0), \dots, w_9 = h(1, 1)$. An $m \times n$ filter mask is generated in a similar manner. Recall that a 2-D Gaussian function has a bell shape, and that the standard deviation controls the “tightness” of the bell.

$$\begin{aligned} h(0, 0) &= e^{-\frac{0^2+0^2}{2\sigma^2}} \\ &= e^0 = 1 \end{aligned}$$

$$\frac{1}{2\pi\sigma^2} = 4$$



-1, 0, ...,
ier. Recall
deviation



3x3



2	4	2
4	8	4
2	4	2

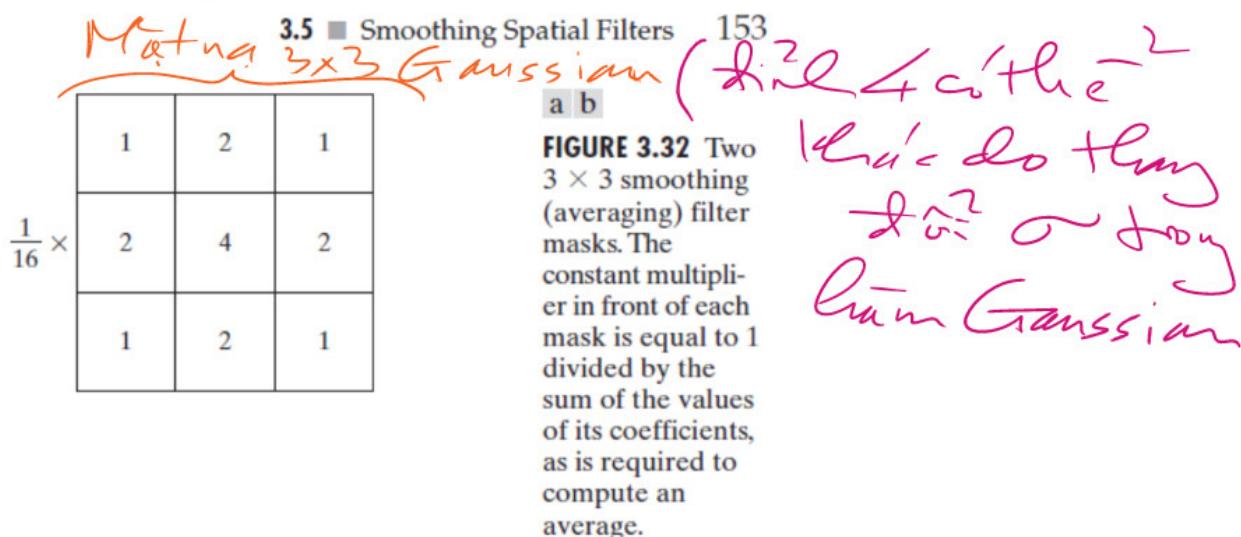
1	x	2	x	1
x	x	4	x	x
2	4	8	4	2
x	x	4	-	-
1	2	1	-	-

(5x5)

② Làm mờ với 2 mátx Gaussian 3x3

$$\begin{aligned} \bar{C}(2,2) &= \frac{1}{16} \times \left(\begin{array}{|ccc|} \hline 4 & 0 & 23 \\ 23 & 34 & 45 \\ 43 & 54 & 67 \\ \hline \end{array} \right) \times \left(\begin{array}{|ccc|} \hline 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \\ \hline \end{array} \right) \\ &= \frac{1}{16} \times (4 \times 1 + 0 \times 2 + 23 \times 1 + \\ &\quad 23 \times 2 + 34 \times 4 + 45 \times 2 + \\ &\quad 43 \times 1 + 54 \times 2 + 67 \times 1) \\ &= \frac{512}{16} = 32,31 \approx 32 \end{aligned}$$

*Điểm số anh
Làm mờ với 2 Gaussian*

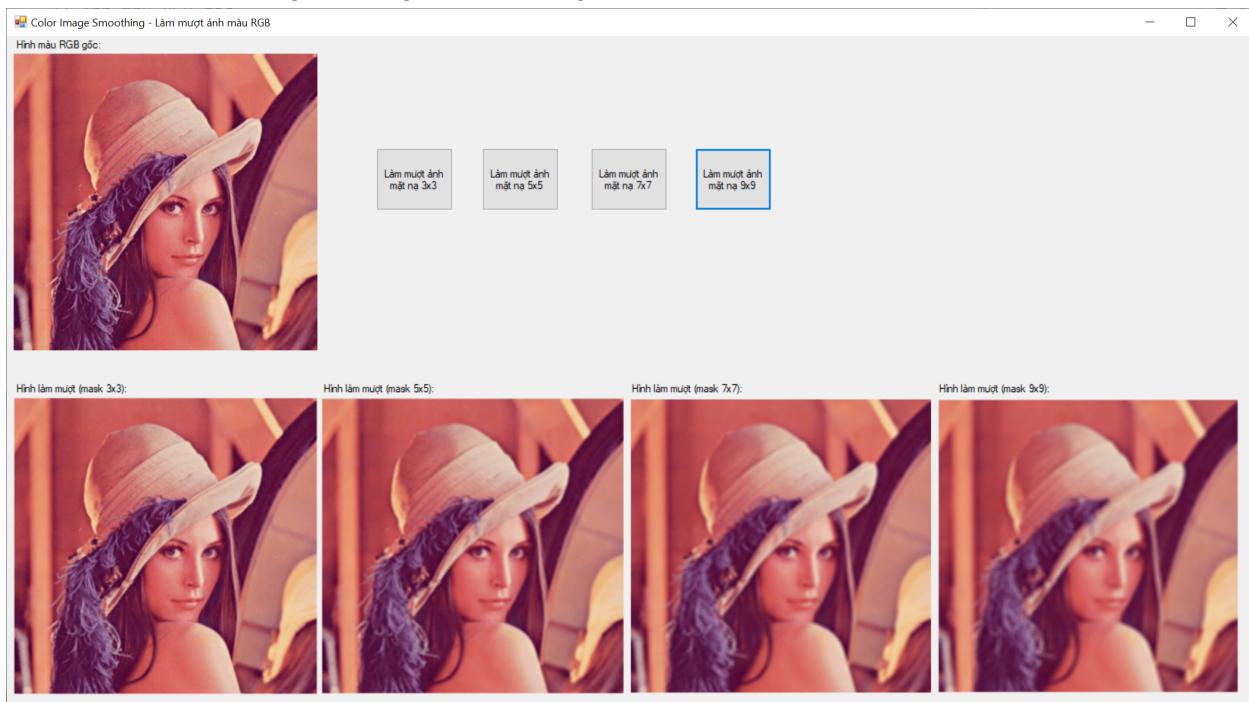


Ví dụ code với mặt nạ Gaussian 5x5 như sau:

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

$\frac{1}{273}$

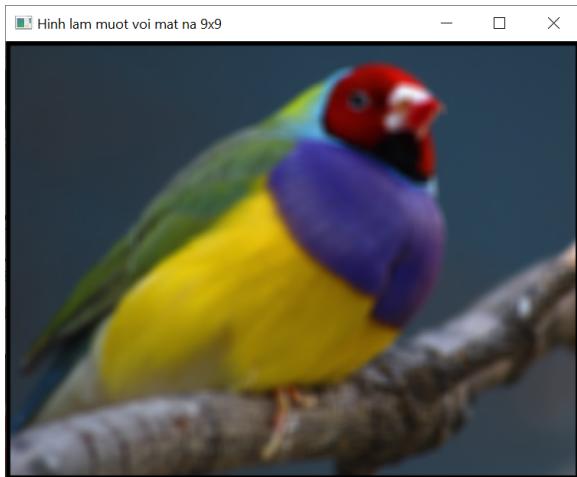
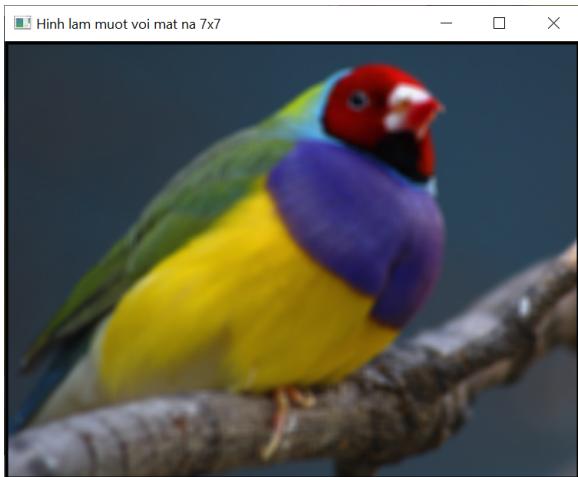
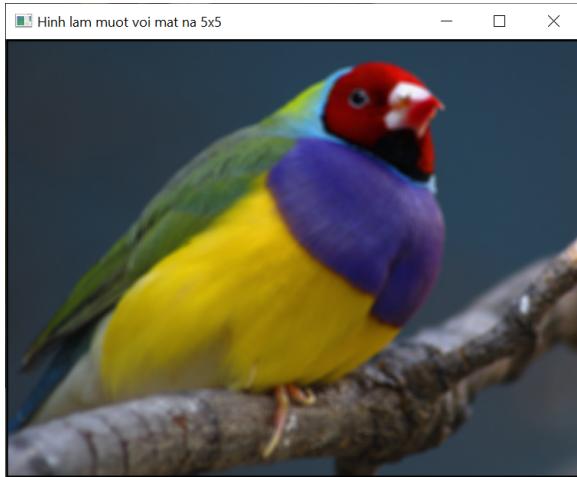
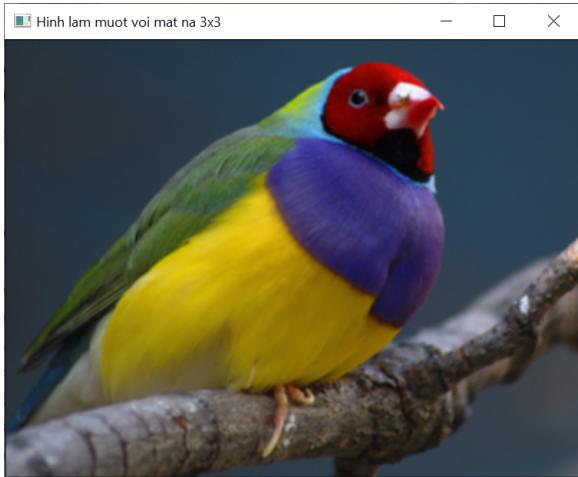
Kết quả C#.NET (Dùng phương pháp Average):



Kết quả Python (Dùng phương pháp Average):



Kết quả Python với hình bird_small.jpg:



MINI-PROJECT 12:

🎀 Color Image Sharpening - Làm nét ảnh màu🎀

🎀👉 Tham khảo tài liệu Digital Image Processing 3rd Edition trang 464, mục 6.6.2. Color Image Sharpening, công thức 6.6-3

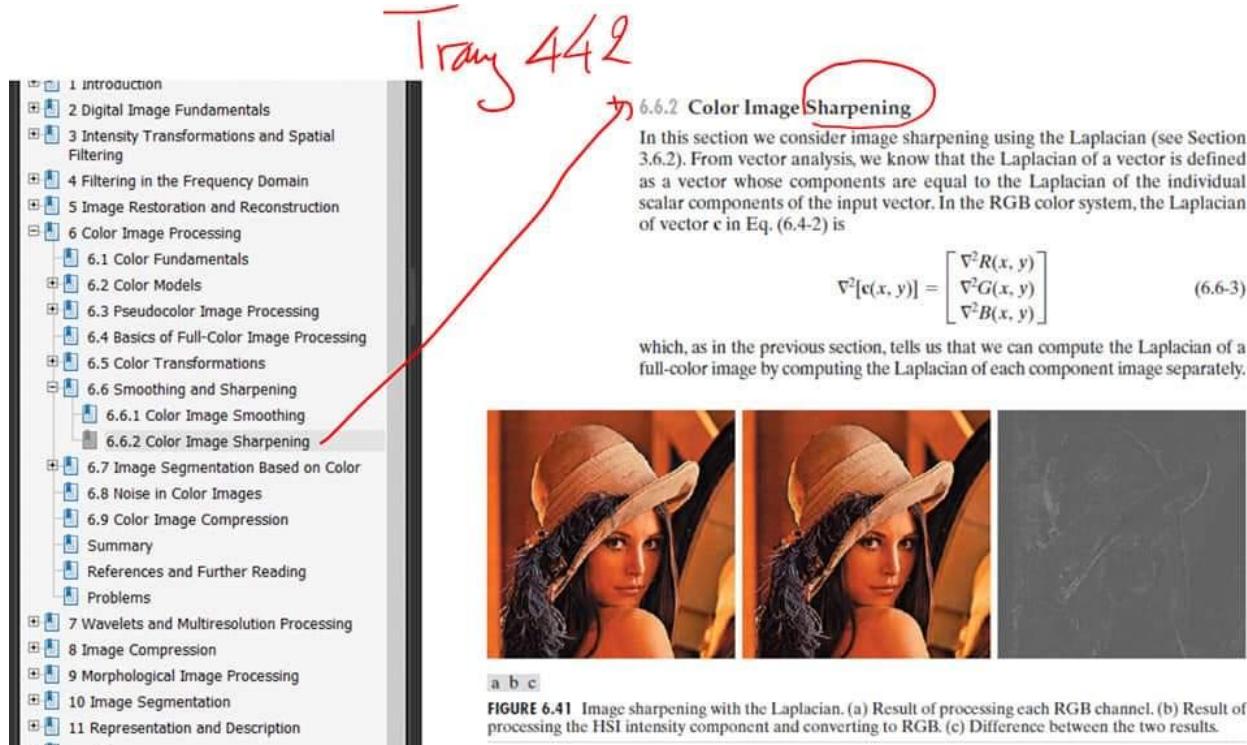
- 🎯 Hiểu lý thuyết làm nét ảnh màu (color image sharpening)
- 🎯 Lập trình làm nét ảnh màu bằng C#.NET
- 🎯 Lập trình làm nét ảnh màu bằng Python

❤️ Videos bài giảng của thầy:

1. UTEX#15 - Thuật toán làm nét ảnh màu (Color Image Sharpening)
<https://youtu.be/NJZJ8Vkfql>

❤️ Thầy tổng hợp các hình chụp bài giảng của thầy để các e xem lại:

Trang 442



The screenshot shows the table of contents of the Digital Image Processing 3rd Edition book. A red arrow points from the left margin to the '6.6.2 Color Image Sharpening' section. The page number 'Trang 442' is handwritten in red at the top right of the book image.

6.6.2 Color Image Sharpening

In this section we consider image sharpening using the Laplacian (see Section 3.6.2). From vector analysis, we know that the Laplacian of a vector is defined as a vector whose components are equal to the Laplacian of the individual scalar components of the input vector. In the RGB color system, the Laplacian of vector \mathbf{c} in Eq. (6.4-2) is

$$\nabla^2[\mathbf{c}(x, y)] = \begin{bmatrix} \nabla^2R(x, y) \\ \nabla^2G(x, y) \\ \nabla^2B(x, y) \end{bmatrix} \quad (6.6-3)$$

which, as in the previous section, tells us that we can compute the Laplacian of a full-color image by computing the Laplacian of each component image separately.



FIGURE 6.41 Image sharpening with the Laplacian. (a) Result of processing each RGB channel. (b) Result of processing the HSI intensity component and converting to RGB. (c) Difference between the two results.

6.6.2 Color Image Sharpening

In this section we consider image sharpening using the Laplacian (see Section 3.6.2). From vector analysis, we know that the Laplacian of a vector is defined as a vector whose components are equal to the Laplacian of the individual scalar components of the input vector. In the RGB color system, the Laplacian of vector c in Eq. (6.4-2) is

$$\nabla^2[c(x, y)] = \begin{bmatrix} \nabla^2R(x, y) \\ \nabla^2G(x, y) \\ \nabla^2B(x, y) \end{bmatrix} \rightarrow \begin{array}{l} R \\ G \\ B \end{array} \quad (6.6-3)$$

which, as in the previous section, tells us that we can compute the Laplacian of a full-color image by computing the Laplacian of each component image separately.

3.5 Smoothing Spatial Filters
3.6 Sharpening Spatial Filters
3.6.1 Foundation
3.6.2 Using the Second Derivative for Image Sharpening—The Laplacian
3.6.3 Unsharp Masking and Highboost Filtering
3.6.4 Using First-Order Derivatives for (Nonlinear) Image Sharpening—The Gradient
3.7 Combining Spatial Enhancement Methods
3.8 Using Fuzzy Techniques for Intensity Transformations and Spatial Filtering
Summary
References and Further Reading
Problems
4 Filtering in the Frequency Domain
5 Image Restoration and Reconstruction
6 Color Image Processing
6.1 Color Fundamentals
6.2 Color Models
6.3 Pseudocolor Image Processing
6.4 Basics of Full-Color Image Processing
6.5 Color Transformations
6.6 Smoothing and Sharpening
6.6.1 Color Image Smoothing
6.6.2 Color Image Sharpening
6.7 Image Segmentation Based on Color

3.6.2 Using the Second Derivative for Image Sharpening—The Laplacian

In this section we consider the implementation of 2-D, second-order derivatives and their use for image sharpening. We return to this derivative in Chapter 10, where we use it extensively for image segmentation. The approach basically consists of defining a discrete formulation of the second-order derivative and then constructing a filter mask based on that formulation. We are interested in *isotropic* filters, whose response is independent of the direction of the discontinuities in the image to which the filter is applied. In other words, isotropic filters are *rotation invariant*, in the sense that rotating the image and then applying the filter gives the same result as applying the filter to the image first and then rotating the result.

It can be shown (Rosenfeld and Kak [1982]) that the simplest isotropic derivative operator is the Laplacian, which, for a function (image) $f(x, y)$ of two variables, is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (3.6-3)$$

Because derivatives of any order are linear operations, the Laplacian is a linear operator. To express this equation in discrete form, we use the definition in Eq. (3.6-2), keeping in mind that we have to carry a second variable. In the x -direction, we have

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \quad (3.6-4)$$

and, similarly, in the y -direction we have

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \quad (3.6-5)$$

Laplacian: *đạo hàm bậc 2*

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (3.6-3)$$

Because derivatives of any order are linear operations, the Laplacian is a linear operator. To express this equation in discrete form, we use the definition in Eq. (3.6-2), keeping in mind that we have to carry a second variable. In the x -direction, we have

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \quad (3.6-4)$$

and, similarly, in the y -direction we have

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \quad (3.6-5)$$

3.6 ■ Sharpening Spatial Filters 161

Therefore, it follows from the preceding three equations that the discrete Laplacian of two variables is

$$\begin{aligned} \nabla^2 f(x, y) &= f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) \\ &\quad - 4f(x, y) \end{aligned} \quad (3.6-6)$$

As noted in the previous paragraph, it is important to keep in mind which definition of the Laplacian is used. If the definition used has a negative center coefficient, then we subtract, rather than add, the Laplacian image to obtain a sharpened result. Thus, the basic way in which we use the Laplacian for image sharpening is

$$g(x, y) = f(x, y) + c[\nabla^2 f(x, y)] \quad |c = -1 \quad (3.6-7)$$

where $f(x, y)$ and $g(x, y)$ are the input and sharpened images, respectively. The constant is $c = -1$ if the Laplacian filters in Fig. 3.37(a) or (b) are used, and $c = 1$ if either of the other two filters is used.

Điểm ảnh sau khi làm sắc nét
Phép biến đổi Laplacian của $f(x, y)$

It can be shown (Rosenfeld and Kak [1982]) that the simplest isotropic derivative operator is the Laplacian, which, for a function (image) $f(x, y)$ of two variables, is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Hàm f có 2 biến x, y

(3.6-3)

Because derivatives of any order are linear operations, the Laplacian is a linear operator. To express this equation in discrete form, we use the definition in Eq. (3.6-2), keeping in mind that we have to carry a second variable. In the x -direction, we have

~~Differential~~

Véc 2 the

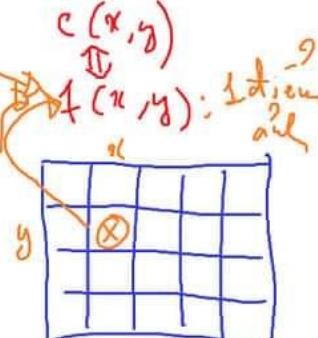
$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \quad (3.6-4)$$

and, similarly, in the y -direction we have

~~Differential~~

Véc 2

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \quad (3.6-5)$$



~~Differential~~

Véc 2 của x theo y

$$\begin{aligned} \nabla^2 f(x, y) &= \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \\ &= f(x+1, y) + f(x-1, y) - 2f(x, y) + \\ &\quad f(x, y+1) + f(x, y-1) - 2f(x, y) \\ \nabla^2 f(x, y) &= f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \end{aligned}$$

Laplacian của hàm $f(x, y) \Rightarrow$ Cúp chia lỗ kết quả
để để tính giá trị
sắc nét của điểm $f(x, y)$

It can be shown (Rosenfeld and Kak [1982]) that the simplest isotropic derivative operator is the **Laplacian**, which, for a function (image) $f(x, y)$ of two variables, is defined as

$$\text{Đạo hàm riêng} \quad \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (3.6-3)$$

Hàm f là 2 biến x, y

Because derivatives of any order are linear operations, the Laplacian is a linear operator. To express this equation in discrete form, we use the definition in Eq. (3.6-2), keeping in mind that we have to carry a second variable. In the x -direction, we have

$$\text{Đạo hàm riêng} \quad \frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \quad (3.6-4)$$

(theo x)

and, similarly, in the y -direction we have

$$\text{Đạo hàm riêng} \quad \frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \quad (3.6-5)$$

(theo y)

$$3.6-3 \Rightarrow \boxed{\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}} \Rightarrow f(x+1, y) + f(x-1, y) - 2f(x, y) + f(x, y+1) + f(x, y-1) - 2f(x, y) \\ \Rightarrow \boxed{\Delta f(x, y)} = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

Laplacian

15	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
16	65	9	2	0	1	6	8	9	0	34	50	111	21	145	234	235	22	21	37	43								
17	234	54	85	34	232	43	54	7	76	45	34	85	87	4	32	6	565	23	4	5								
18	6	32	54	65	6	34	58	43	54	55	76	3	54	76	23	54	65	76	87	2								
19	43	43	54	65	76	39	97	23	54	65	76	87	34	54	65	34	4	34	34	34	34							

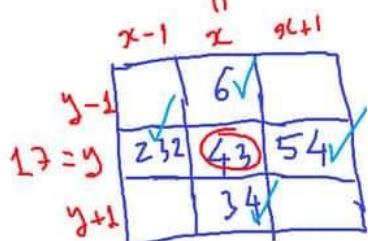
$$f(x,y) = f(5,17) \xrightarrow{\text{Laplacian}} \nabla^2 f(5,17)$$

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

↓
Laplacian

$$\Rightarrow \nabla^2 f(5,17) = 54 + 232 + 34 + 6 - 4 \times 43 = \underline{\underline{154}}$$

Laplacian



Điều chỉnh sao khi dùng sharpening

$$g(x, y) = f(x, y) + c [\nabla^2 f(x, y)] \quad (3.6-7)$$

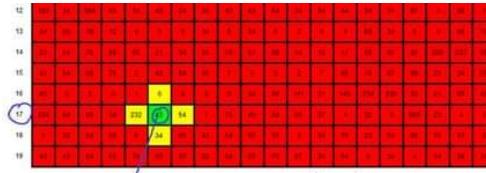
Laplacian của $f(x,y)$
Điều chỉnh cù Sharpening.

where $f(x, y)$ and $g(x, y)$ are the input and sharpened images, respectively.
The constant is $c = -1$ if the Laplacian filters in Fig. 3.37(a) or (b) are used,
and $c = 1$ if either of the other two filters is used.

$$\underline{\underline{154}} \Rightarrow g(5,17) = f(5,17) + c \times 154$$

Laplacian @vdu: $c = -1 \Rightarrow g(5,17) = 43 + (-1) \times 154 = \underline{\underline{-111}}$

Gia trị điều chỉnh
cù $(5,17)$ sau sharpening



$$f(5, 17) = f(5, 17) \xrightarrow{\text{Laplacian}} \nabla^2 f(5, 17)$$

$$\nabla^2 f(5, 17) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

Laplacian

$$\Rightarrow \nabla^2 f(5, 17) = 54 + 232 + 34 + 6 - 4 \times 43 = 154 \quad \begin{matrix} 5 \\ \text{Laplacian} \end{matrix}$$

x-1	2	y+1
17=5	232	43
y-1	34	54

Điểm 2, sao khử nhiễu sharpening

$$g(x, y) = f(x, y) + c[\nabla^2 f(x, y)] \quad (3.6-7)$$

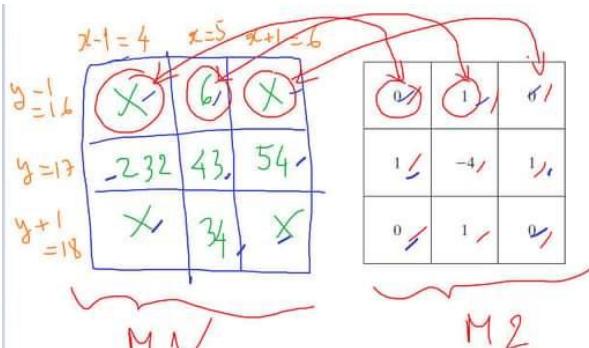
Điểm 2 là cùn sharpening

where $f(x, y)$ and $g(x, y)$ are the input and sharpened images, respectively.
The constant is $c = 1$ if the Laplacian filters in Fig. 3.37(a) or (b) are used,
and $c = 0$ if either of the other two filters is used.

$$g(5, 17) = f(5, 17) + c \times 154$$

$$\text{vì } c = -1 \Rightarrow g(5, 17) = 43 + (-1) \times 154 = -111$$

Ghi rõ điều kiện
để $(5, 17)$ sao sharpening



$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \quad B = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix}$$

$$\nabla^2 f(5, 17) = 6 + 232 \times 1 + 43(-4) + 54 \times 1 + 34 \times 1 = 154$$

Laplacian $f(x, y)$

\Rightarrow Nhân tử là chấp 2 mảng

$$A \otimes B = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3 + a_4 \cdot b_4 + a_5 \cdot b_5 + a_6 \cdot b_6 + a_7 \cdot b_7 + a_8 \cdot b_8 + a_9 \cdot b_9$$

$$\Rightarrow \nabla^2 f(5, 17) = 54 + 232 + 34 + 6 - 4 \times 43 = 154 \quad \begin{matrix} 5 \\ \text{Laplacian} \end{matrix}$$

x-1	2	y+1
17=5	232	43
y-1	34	54

1. Giả sử kênh màu R của một ảnh màu RGB kích thước 20x20

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	18	21	28	34	40	5	7	12	27	70	39	78	48	75	152	14	138	204	255	17
1	9	4	0	23	31	32	54	29	87	88	12	43	45	26	98	46	123	26	282	231
2	90	23	34	47	87	50	6	6	57	9	54	3	49	65	78	99	6	66	78	23
3	12	43	94	67	23	49	34	54	60	78	87	98	90	12	32	43	24	54	95	78
4	23	43	48	60	76	66	36	26	54	98	126	140	231	150	254	222	23	32	46	68
5	43	54	55	78	23	54	59	43	43	54	58	79	42	5	78	9	12	43	95	7
6	28	19	38	23	34	47	46	54	60	78	78	98	18	17	8	4	9	16	7	9
7	38	22	3	5	7	6	9	1	3	3	4	9	2	8	43	65	77	28	143	155
8	75	23	43	33	43	54	60	85	87	12	5	234	54	58	23	54	78	43	34	43
9	81	64	66	20	4	6	8	8	86	54	65	23	48	2	43	2	65	4	12	43
10	88	34	56	90	47	34	34	2	50	7	4	54	34	43	25	75	47	96	9	23
11	29	56	65	78	34	78	28	6	34	8	34	85	32	98	2	27	37	84	9	93
12	123	34	884	80	34	40	54	34	43	43	34	34	54	48	34	78	87	3	92	5
13	34	95	78	12	4	5	2	14	5	54	6	3	4	4	95	34	5	4	98	78
14	28	54	78	89	85	21	38	65	78	87	80	14	16	17	65	85	87	200	237	98
15	43	54	65	78	2	40	54	65	7	9	9	2	7	65	78	87	23	34	65	
16	98	0	2	0	7	6	18	9	0	34	55	111	21	140	234	235	22	28	37	43
17	234	54	86	34	232	43	54	7	26	45	34	65	87	4	92	6	593	23	4	5
18	4	32	54	60	6	34	50	42	54	66	78	3	54	78	23	54	85	78	27	2
19	19	12	54	69	76	89	87	23	54	66	78	87	34	54	6	34	4	34	34	14

$$f(x,y) = f(5,17) \xrightarrow{\text{Laplacian}} \nabla^2 f(5,17)$$

$x-1$	x	$x+1$	
$y-1$	1	6	9
$y=17$	232	43	54
$y+1$	6	34	65

Làm sao net $f(5,17)$

$$g(5,17) = f(5,17) + C \times \left[\nabla^2 f(5,17) \right]$$

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

Công thức tính Laplacian

$$\begin{aligned} \nabla^2 f(5,17) &= f(6,17) + f(4,17) + f(5,18) + f(5,16) - 4f(5,17) \\ &= 54 + 232 + 34 + 6 - 4 \times 43 = 154 \end{aligned}$$

$$\Rightarrow g(5,17) = 43 + (-1) \times 154 = -111$$

$$C = -1$$

giá trị điểm
San kín làm
Sắc nét

Mask: mặt nạ $\rightarrow 6'$
4 masks

0	1	0
1	-4	
0	1	0

0	-1	0
-1	4	-1
0	-1	0

15	43	50	65	70	72	42	54	65	7	8	9	12
15	35	9	2	7	1	6	8	2	0	14	10	11
17	232	54	65	34	232	33	54	7	78	40	24	66
19	67	32	64	60	0	34	65	42	74	65	70	71
19	43	42	54	65	78	95	97	23	34	95	75	81

Nhân tích chập 2 ma trận

$$\begin{aligned}
 f(x-1,y) &= f(5,1) \\
 &= 1 * 0 + 6 * (-1) + 8 * 0 + \\
 &= 232 * (-1) + 43 * 4 + 54 * (-1) + \\
 &= 6 * 0 + 34 * (-1) + 65 * 6 \\
 &= 154
 \end{aligned}$$

$$f(x-1,y+1) = f(5,2) = 4 * f(1,2)$$

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \quad B = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix}$$

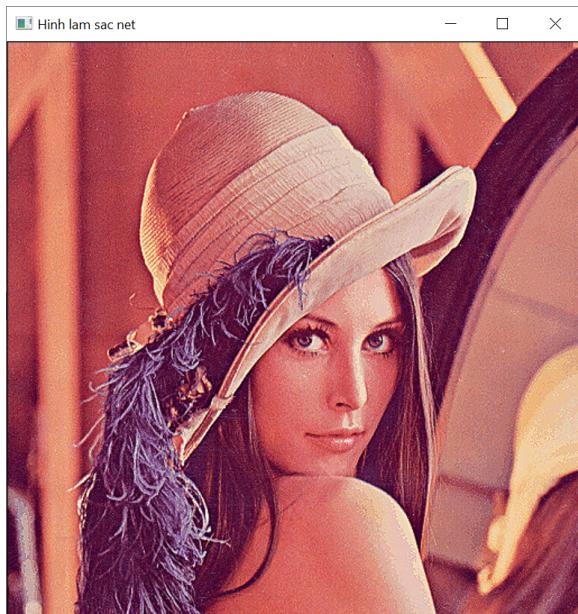
Nhân tích chập

$$\begin{aligned}
 A \otimes B &= a_1 * b_1 + a_2 * b_2 + a_3 * b_3 + \\
 &\quad a_4 * b_4 + a_5 * b_5 + a_6 * b_6 + \\
 &\quad a_7 * b_7 + a_8 * b_8 + a_9 * b_9 \\
 &\Rightarrow \text{Lết qua} \\
 &\text{là 1 con số}
 \end{aligned}$$

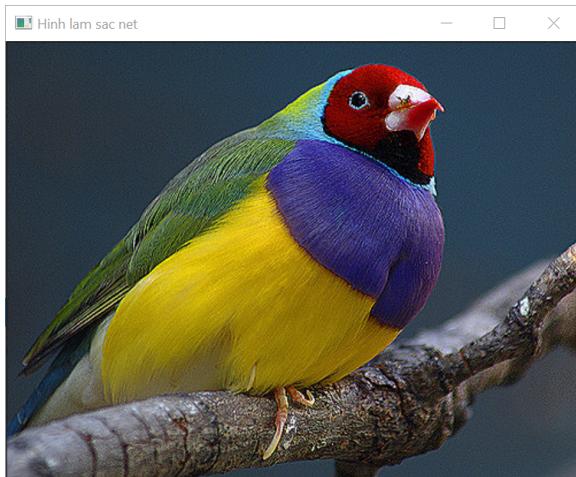
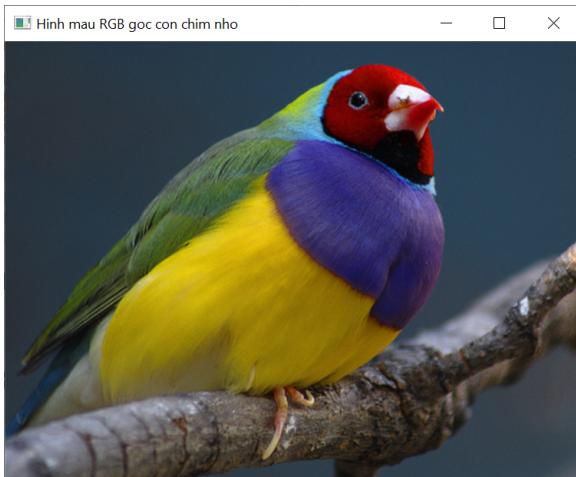
Kết quả C#.NET:



Kết quả Python:



So sánh hình gốc với hình sắc nét với mẫu bird_mall (Python):



MINI-PROJECT 13

SEGMENTATION IN RGB VECTOR SPACE

- 🎯 Hiểu lý thuyết làm phân đoạn (segmentation) ảnh màu
- 🎯 Lập trình làm phân đoạn (segmentation) ảnh màu bằng C#.NET
- 🎯 Lập trình làm phân đoạn (segmentation) ảnh màu bằng Python

Videos bài giảng của thầy:

1. UTEX#16 - Thuật toán phân đoạn ảnh màu RGB (Color Segmentation)

<https://youtu.be/J8nIJvbcmuY>

📝 #SEGMENTATION có nghĩa tiếng Việt là PHÂN ĐOẠN ẢNH. Mình hay nghe từ XÓA PHÔNG, hoặc khi đóng phim thì các diễn viên sẽ diễn trước phòng màn màu xanh, sau đó hậu kỳ sẽ ghép các cảnh diễn đó vào các cảnh đồ họa khác,... những cái đó là ứng dụng của SEGMENTATION. Các em xem kết quả của một ví dụ phân đoạn ảnh như trong hình.

🌹 Tài liệu tham khảo tại mục 6.7.2 trang 445 trong cuốn tài liệu Rafael C. Gonzalez, Richard E. Woods - Digital Image Processing (2008, Prentice Hall).

🌹 Hình xử lý hoặc hình BIRD hoặc hình LENA.

🌺👉 Công thức chính sử dụng trong project này là hàm tính khoảng cách Euclidean Distance, xem công thức 6.7-1 trang 445. Công thức này thực ra rất đơn giản và rất quen thuộc với chúng ta, nhưng giờ nghe tên tiếng Anh thì có vẻ hơi lạ, khi thầy ra nước ngoài học thầy cũng bỡ ngỡ vì không biết nó là gì, nhưng thực ra nó giống như công thức tính khoảng cách giữa 2 điểm A(xA, yA, zA) và B(xB, yB, zB):

$$D = \text{SQRT}[(xA - xB)^2 + (yA - yB)^2 + (zA - zB)^2]$$

Trong đó: SQRT(X) là hàm tính căn bậc hai của số X

🌺 Trong công thức 6.7-1 có hai vector a và z, trong đó vector a gọi là VECTOR MÀU TRUNG BÌNH (AVERAGE COLOR).

🌺 Cách tính vector a như sau:

- Trong ảnh cần làm segmentation, mình chọn một vùng ảnh từ vị trí (x1, y1) đến vị trí (x2, y2), xem hình minh họa thầy đính kèm, thì vùng ảnh thầy chọn sẽ từ vị trí (80, 400) đến vị trí (150, 500), vùng này thuộc vùng màu tóc của cô gái, nên sau khi chạy segmentation thì màu tóc của cô gái hoặc những màu khác gần gần giống sẽ bị remove (như hình kết quả bên trái).
- Vector a = [aR aG aB], chứa ba thành phần trung bình màu cho từng kênh R-G-B. Tại mỗi kênh màu R-G-B chúng ta tiến hành tính trung bình cộng của tất cả các điểm ảnh (pixel) thuộc vùng ảnh đã chọn ở trên.

🌺 Còn vector z chính là điểm ảnh tại vị trí (x, y) mà chúng ta đang muốn tính xem nó là điểm thuộc nền (background) hay thuộc đối tượng (object). Tức là vector z được xác định như sau:

$$z(x, y) = [R(x,y) G(x,y) B(x,y)] = [zR zG zB]$$

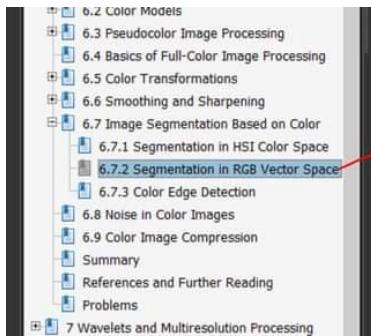
🌺 Như vậy, áp dụng công thức 6.7-1 để tính Euclidean Distance giữa hai vector a và z sẽ như sau:

$$D(z,a) = \text{SQRT}[(zR - aR)^2 + (zG - aG)^2 + (zB - aB)^2]$$

💡 Sau khi tính được giá trị $D(z,a)$, chúng ta sẽ so sánh với giá trị ngưỡng (threshold) D_0 để xác định xem điểm $z(x,y)$ đang xét là background hay object, như sau:

- Nếu $D(z,a) \leq D_0$ thì $z(x,y)$ là background, chúng ta có thể set màu cho điểm này về màu trắng (255) hoặc đen (0) hay bất kỳ màu gì tùy các em chọn.
- Ngược lại thì $z(x,y)$ là object, chúng ta giữ nguyên màu cho điểm này.

👉 Thầy tổng hợp các hình chụp bài giảng của thầy để các e xem lại:



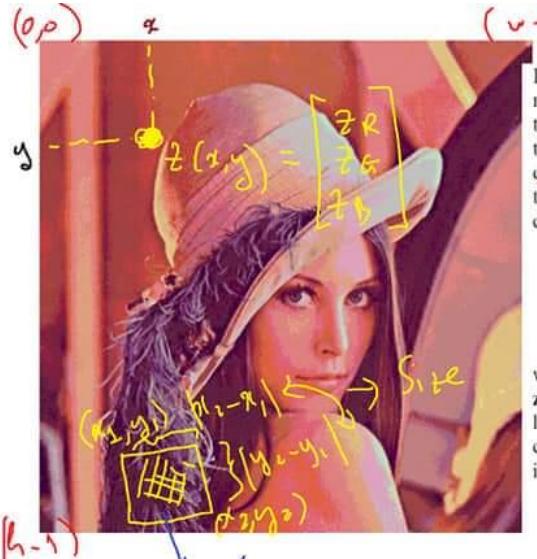
6.7.2 Segmentation in RGB Vector Space

Although, as mentioned numerous times in this chapter, is more intuitive, segmentation is one area in which better obtained by using RGB color vectors. The approach is pose that the objective is to segment objects of a specific RGB image. Given a set of sample color points represen

Let this average color be denoted by the RGB vector \mathbf{a} . The objective of segmentation is to classify each RGB pixel in a given image as having a color in the specified range or not. In order to perform this comparison, it is necessary to have a measure of similarity. One of the simplest measures is the Euclidean distance. Let \mathbf{z} denote an arbitrary point in RGB space. We say that \mathbf{z} is *similar* to \mathbf{a} if the distance between them is less than a specified threshold, D_0 . The Euclidean distance between \mathbf{z} and \mathbf{a} is given by

$$\begin{aligned}
 D(\mathbf{z}, \mathbf{a}) &= \|\mathbf{z} - \mathbf{a}\| \\
 &= [(\mathbf{z} - \mathbf{a})^T(\mathbf{z} - \mathbf{a})]^{\frac{1}{2}} \\
 &= [(z_R - a_R)^2 + (z_G - a_G)^2 + (z_B - a_B)^2]^{\frac{1}{2}}
 \end{aligned} \tag{6.7-1}$$

where the subscripts R , G , and B denote the RGB components of vectors \mathbf{a} and \mathbf{z} . The locus of points such that $D(\mathbf{z}, \mathbf{a}) \leq D_0$ is a solid sphere of radius D_0 , as il-



(h-1)

Vectō màu trung bình
(Average Color Vector)

$$\mathbf{a} = \begin{bmatrix} a_R \\ a_G \\ a_B \end{bmatrix}$$

for $i = x_1 \rightarrow x_2$
 for $j = y_1 \rightarrow y_2$
 | get Pixel(i, j) $\rightarrow p_{(i,j)}$ $\rightarrow R_{i,j}$
 | \Rightarrow Công thức \bar{a}_{ij}

(v-1)

Let this average color be denoted by the RGB vector \mathbf{a} . The objective of segmentation is to classify each RGB pixel in a given image as having a color in the specified range or not. In order to perform this comparison, it is necessary to have a measure of similarity. One of the simplest measures is the Euclidean distance. Let \mathbf{z} denote an arbitrary point in RGB space. We say that \mathbf{z} is similar to \mathbf{a} if the distance between them is less than a specified threshold, D_0 . The Euclidean distance between \mathbf{z} and \mathbf{a} is given by

$$D(\mathbf{z}, \mathbf{a}) = \|\mathbf{z} - \mathbf{a}\|$$

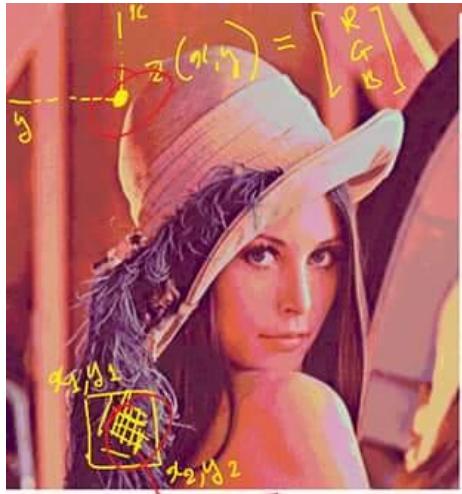
Công thức tính khoảng cách Euclidean
 $= [(\mathbf{z} - \mathbf{a})^T (\mathbf{z} - \mathbf{a})]^{\frac{1}{2}}$
 $= [(z_R - a_R)^2 + (z_G - a_G)^2 + (z_B - a_B)^2]^{\frac{1}{2}}$

where the subscripts R , G , and B denote the RGB components of vectors \mathbf{a} and \mathbf{z} . The locus of points such that $D(\mathbf{z}, \mathbf{a}) \leq D_0$ is a solid sphere of radius D_0 , as illustrated in Fig. 6.43(a). Points contained within the sphere satisfy the specified color criterion; points outside the sphere do not. Coding these two sets of points in the image with, say, black and white, produces a binary segmented image.

A useful generalization of Eq. (6.7-1) is a distance measure of the form

$$D(\mathbf{z}, \mathbf{a}) = [(\mathbf{z} - \mathbf{a})^T \mathbf{C}^{-1} (\mathbf{z} - \mathbf{a})]^{\frac{1}{2}} \quad (6.7-2)$$

trong D_0 (threshold)



Let this average color be denoted by the RGB vector \mathbf{a} . The objective of segmentation is to classify each RGB pixel in a given image as having a color in the specified range or not. In order to perform this comparison, it is necessary to have a measure of similarity. One of the simplest measures is the Euclidean distance. Let \mathbf{z} denote an arbitrary point in RGB space. We say that \mathbf{z} is *similar* to \mathbf{a} if the distance between them is less than a specified threshold, D_0 . The Euclidean distance between \mathbf{z} and \mathbf{a} is given by

$$\begin{aligned} D(\mathbf{z}, \mathbf{a}) &= \|\mathbf{z} - \mathbf{a}\| \\ &= \sqrt{(\mathbf{z}^T \mathbf{z} - \mathbf{z}^T \mathbf{a} - \mathbf{a}^T \mathbf{z} + \mathbf{a}^T \mathbf{a})} \\ &= \sqrt{(z_R - a_R)^2 + (z_G - a_G)^2 + (z_B - a_B)^2} \end{aligned} \quad (6.7-1)$$

where the subscripts R , G , and B denote the RGB components of vectors \mathbf{a} and \mathbf{z} . The locus of points such that $D(\mathbf{z}, \mathbf{a}) \leq D_0$ is a solid sphere of radius D_0 , as illustrated in Fig. 6.43(a). Points contained within the sphere satisfy the specified color criterion; points outside the sphere do not. Coding these two sets of points in the image with, say, black and white, produces a binary segmented image.

A useful generalization of Eq. (6.7-1) is a distance measure of the form

$$D(\mathbf{z}, \mathbf{a}) = \sqrt{(\mathbf{z}^T \mathbf{C}^{-1} \mathbf{z} - 2\mathbf{z}^T \mathbf{C}^{-1} \mathbf{a} + \mathbf{a}^T \mathbf{C}^{-1} \mathbf{a})} \quad (6.7-2)$$

$$\mathbf{a} = \begin{bmatrix} a_R \\ a_G \\ a_B \end{bmatrix}$$

Euclidean distance

$A(x_b, y_b)$

$A(x_a, y_a)$

$$d_{AB} = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$

$$\mathbf{z} = \begin{bmatrix} z_R \\ z_G \\ z_B \end{bmatrix} : \text{RGB}$$

$$\mathbf{a} = \begin{bmatrix} a_R \\ a_G \\ a_B \end{bmatrix} \Rightarrow D(z, a) = \sqrt{\dots}$$

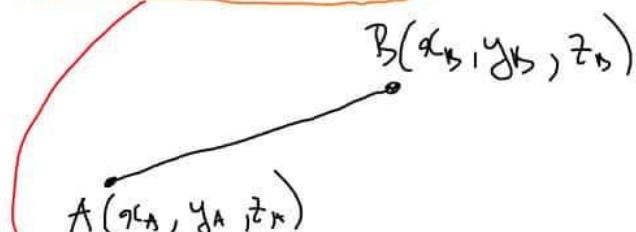
$$D(z, a) = \sqrt{(z_R - a_R)^2 + (z_G - a_G)^2 + (z_B - a_B)^2}$$

$$\Rightarrow \begin{cases} a_R = \frac{x_R}{\text{size}} \\ a_G = \frac{y_G}{\text{size}} \\ a_B = \frac{z_B}{\text{size}} \end{cases}$$

$\text{Size} = |x_2 - x_1| * |y_2 - y_1|$

$$z = \begin{bmatrix} z_R \\ z_G \\ z_B \end{bmatrix} \quad a = \begin{bmatrix} a_R \\ a_G \\ a_B \end{bmatrix} \quad \Rightarrow$$

Euclidean Distance:



$$D(A, B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2}$$

$$\Rightarrow D(z, a) = \sqrt{(z_R - a_R)^2 + (z_G - a_G)^2 + (z_B - a_B)^2}$$

↓ Threshold ↑ (dcm) ↑ (max)

Euclidean Distance

$\Rightarrow \text{if } (D < D_0) \Rightarrow z \in \text{Background}$

$\text{else } \Rightarrow z \in \text{Object}$

Hintergrund Objekt

$z(x,y) = \begin{bmatrix} z_R \\ z_G \\ z_B \end{bmatrix}$

$a = \begin{bmatrix} a_R \\ a_G \\ a_B \end{bmatrix}$: Average Color Vector

```

double[] VectorA(Bitmap HinhDoc, int x1, y1, x2, y2)
{
    double[] a = new double[3];
    a[0] = a[1] = a[2] = 0;
    for (int x = x1; x <= x2; x++)
        for (int y = y1; y <= y2; y++)
        {
            Color pixel = HinhDoc.GetPixel(x, y);
            a[0] += pixel.R;
            a[1] += pixel.G;
            a[2] += pixel.B;
        }
    double size = Math.Abs(x2 - x1) * Math.Abs(y2 - y1);
    a[0] /= size;
    a[1] /= size;
    a[2] /= size;
    return a;
}

```

$D(z, a) = \sqrt{(z_R - a_R)^2 + (z_G - a_G)^2 + (z_B - a_B)^2}$

If $(D \leq D_0) \Rightarrow z(x,y) \in \text{Background}$
 else $\Rightarrow z(x,y) \in \text{Object}$

Phân đoạn ảnh - Segmentation

Chọn vùng ảnh để tính Average Color:

x1: 80	x2: 150
y1: 400	y2: 500

Chọn ngưỡng D0: 45 Phân Đoán Ánh

Hình RGB gốc:

$z(x,y) = \begin{bmatrix} z_R \\ z_G \\ z_B \end{bmatrix}$

$a = \begin{bmatrix} a_R \\ a_G \\ a_B \end{bmatrix}$: Average Color Vector

Hình được phân đoạn ảnh (Segmented image)

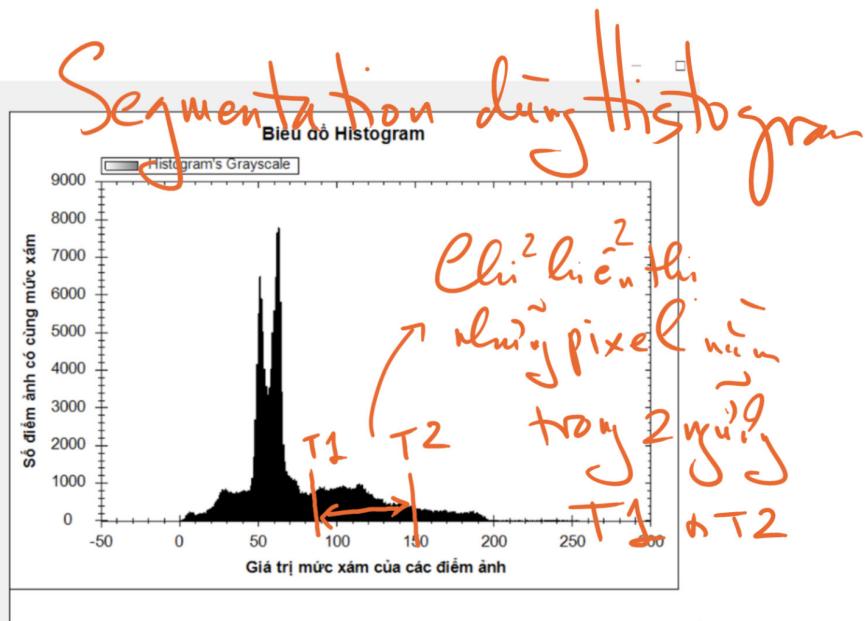
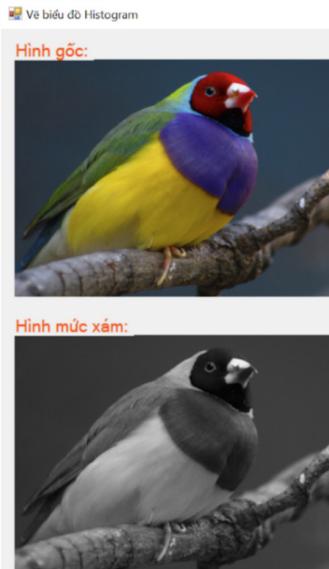
Let this average color be denoted by the RGB vector a . The objective of segmentation is to classify each RGB pixel in a given image as having a color in the specified range or not. In order to perform this comparison, it is necessary to have a measure of similarity. One of the simplest measures is the Euclidean distance. Let z denote an arbitrary point in RGB space. We say that z is similar to a if the distance between them is less than a specified threshold, D_0 . The Euclidean distance between z and a is given by

$$D(z, a) = \|z - a\| = \|(z - a)^T(z - a)\|^{\frac{1}{2}} = \|(z_R - a_R)^2 + (z_G - a_G)^2 + (z_B - a_B)^2\|^{\frac{1}{2}} \quad (6.7-1)$$

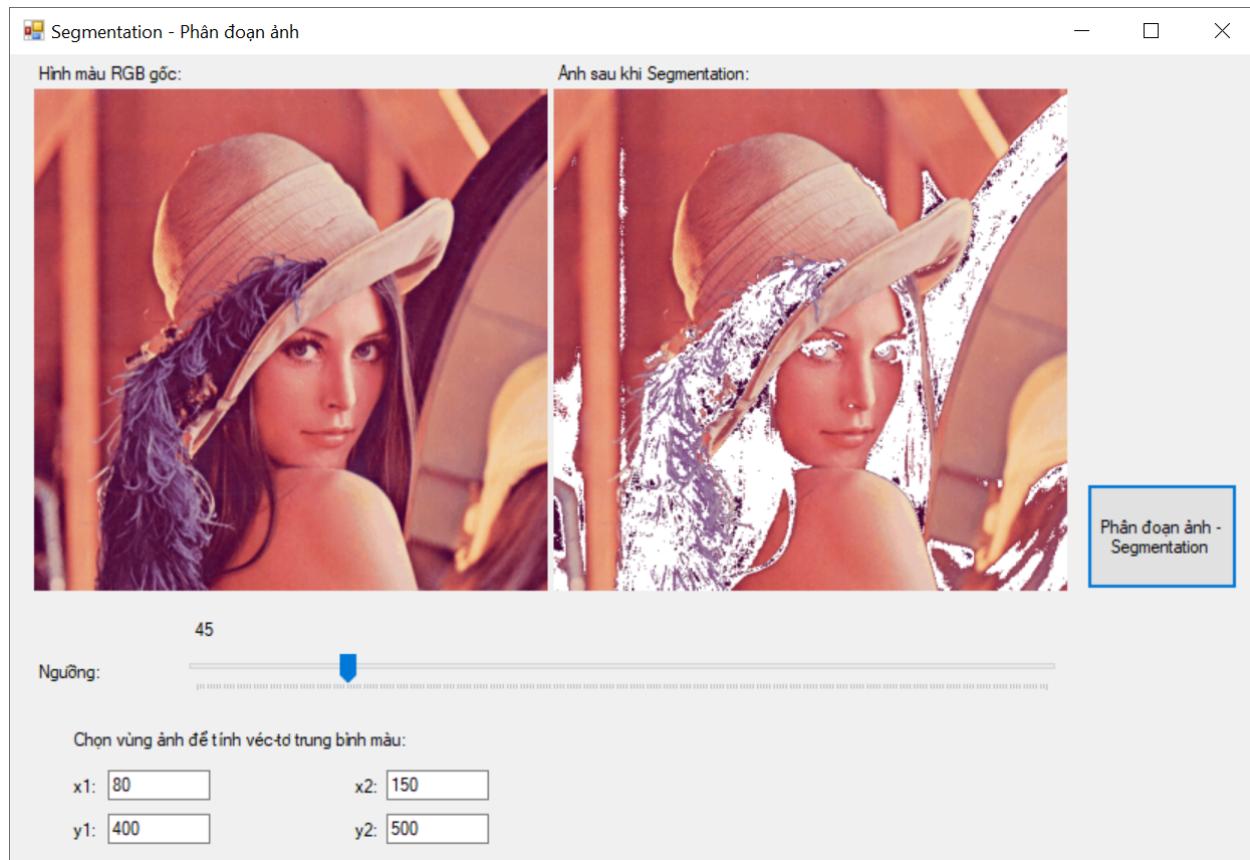
where the subscripts R , G , and B denote the RGB components of vectors a and z . The locus of points such that $D(z, a) \leq D_0$ is a solid sphere of radius D_0 , as il-

$D_0: \text{ngưỡng} (threshold)$

$D(z, a) = \sqrt{(z_R - a_R)^2 + (z_G - a_G)^2 + (z_B - a_B)^2}$
 Euclidean distance



Kết quả C#.NET (phương pháp Euclidean Distance):



Kết quả Python (phương pháp Euclidean Distance):



MINI-PROJECT 14

Edge Detection - Nhận dạng đường biên dùng phương pháp Sobel cho ảnh mức xám (grayscale image).

- ⌚ Hiểu lý thuyết nhận dạng đường biên cho ảnh mức xám
- ⌚ Lập trình nhận dạng đường biên cho ảnh mức xám bằng C#.NET
- ⌚ Lập trình nhận dạng đường biên cho ảnh mức xám bằng Python

 Videos bài giảng của thầy:

1. UTEX#17 - Thuật toán nhận dạng đường biên (Edge Detection) dùng phương pháp Sobel cho ảnh xám

<https://youtu.be/wtKaA3gt5i0>

 Có khá nhiều phương pháp dùng để nhận dạng đường biên, như Sobel, Gradient, Roberts, Prewitt, Canny,...

 Phương pháp nhận dạng đường biên dùng phương pháp Sobel cho ảnh mức xám (grayscale image):

 Nếu ảnh đầu vào là ảnh màu RGB thì chúng ta chuyển sang ảnh mức xám, có thể dùng các phương pháp chuyển đổi ảnh màu RGB sang ảnh mức xám như Average, Lightness hay Luminance tùy các em.

 Giả sử xét điểm ảnh $f(x, y)$ để biết pixel này là thuộc background (nền) hay edge (biên).

 Đầu tiên phải đi tính #Gradient của $f(x, y)$ theo công thức 10.2-9 trang 706.

$$\text{gradient}(f) = [gx \ gy]$$

Trong đó:

- gx = đạo hàm riêng bậc một của f theo x
- gy = đạo hàm riêng bậc một của f theo y

 Tiếp theo tính #Magnitude (length), hay còn gọi là biên độ, của vector $\text{gradient}(f)$ theo công thức 10.2-10. Tuy nhiên, để cho đơn giản hơn cho việc tính toán, giảm tải tính toán cho CPU máy tính thì chúng ta có thể sử dụng công thức tính tương đương 10.2-20 để tính giá trị biên độ này:

$$M(x, y) = |gx| + |gy|$$

 Để biết điểm $f(x, y)$ đang xét là thuộc background hay edge thì chúng ta so sánh giá trị $M(x, y)$ với một giá trị ngưỡng (threshold) D_0 (do người dùng tự nhập).

Nếu $M \leq D_0$ thì $f(x, y)$ là thuộc background, ngược lại thì $f(x, y)$ là thuộc edge

 Vậy vấn đề còn lại là làm sao để tính gx và gy mà thôi. Để tính gx , gy thì có nhiều phương pháp như trong mục 10.2.5. Basic Edge Detection ở Chương 10, trang 706: Gradient Operators, ma trận Roberts, ma trận Prewitt, ma trận Sobel,...

 Trong phần giải thích này thầy sẽ hướng dẫn phương pháp Sobel.

 Phương pháp Sobel tính gx và gy bằng hai công thức 10.2-18 và 10.2-19 ở trang 709.

$$gx = (z7 + 2*z8 + z9) - (z1 + 2*z2 + z3)$$

$$gy = (z3 + 2*z6 + z9) - (z1 + 2*z4 + z7)$$

Giả sử điểm đang xét là $f(x, y)$ thì mình sẽ có các điểm lân cận (hàng xóm) của nó như sau:

	$f(x-1, y-1)$	$f(x, y-1)$	$f(x+1, y-1)$	
	$f(x-1, y)$	$f(x, y)$	$f(x+1, y)$	
	$f(x-1, y+1)$	$f(x, y+1)$	$f(x+1, y+1)$	

Để đơn giản thì thay các $f(i, j)$ trong mặt nạ ở trên bằng các phần tử zi cho gọn:

	$z1$	$z2$	$z3$	
	$z4$	$z5$	$z6$	
	$z7$	$z8$	$z9$	

Như vậy $z5$ chính là điểm $f(x, y)$ đang xét, còn lại là các điểm hàng xóm của nó.

 Thay vì tính gx , gy bằng các công thức như thầy đã giải thích ở trên thì ông Sobel ỗng dùng các ma trận như sau:

* Ma trận Sobel theo phương x:

Từ công thức tính

$$gx = (z7 + 2*z8 + z9) - (z1 + 2*z2 + z3)$$

Sobel xây dựng nên ma trận sau:

	-1	-2	-1	
	0	0	0	
	1	2	1	

* Ma trận Sobel theo phương y:

Từ công thức tính

$$gy = (z3 + 2*z6 + z9) - (z1 + 2*z4 + z7)$$

Sobel xây dựng nên ma trận sau:

	-1	0	1	
	-2	0	2	
	-1	0	1	

 Như vậy, khi tính gx , gy thì Sobel lấy vùng ảnh $3x3$ với tâm vùng ảnh là điểm $f(x, y)$ đang xét, nhân tích chập (Convolution) với các ma trận Sobel theo phương x (để tính gx) và ma trận Sobel theo phương y (để tính gy).

 Ví dụ như sau để các em rõ Nhân tích chập vùng ảnh với các ma trận Sobel. Giả sử vùng ảnh $3x3$ đang xét tại điểm $f(16,13) = 5$ có các giá trị pixel như sau:

	3	8	9	
--	---	---	---	--

$$\begin{array}{r}
 | 4 5 7 | \\
 | 9 2 6 | \\
 \Rightarrow gx = 3*(-1) + 8*(-2) + 9*(-1) + \\
 4*0 + 5*0 + 7*0 + \\
 9*1 + 2*2 + 6*1 \\
 = -9 \\
 \text{và } gy = 3*(-1) + 8*0 + 9*1 + \\
 4*(-2) + 5*0 + 7*2 + \\
 9*(-1) + 2*0 + 6*1 \\
 = 9
 \end{array}$$

❤️ Thầy tổng hợp các hình chụp bài giảng của thầy để các e xem lại:

The screenshot shows a software interface with a left sidebar containing a table of contents for 'Morphological Image Processing'. The sidebar includes sections like 'Image Segmentation', 'Fundamentals', 'Point, Line, and Edge Detection' (which is expanded), 'Thresholding', 'Region-Based Segmentation', 'Segmentation Using Morphological Watersheds', 'Motion Segmentation', and 'Summary'. A red callout highlights the section '10.2.5 Basic Edge Detection' on the slide. The slide itself is titled 'Chapter 10' and 'Image Segmentation'. The main content of the slide discusses edge detection, mentioning first-order derivatives and second-order derivatives.

706

Chapter 10 ■ Image Segmentation

10.2.5 Basic Edge Detection

As illustrated in the previous section, the purpose of finding edges can be achieved by computing derivatives. We discuss first-order derivatives in Section 10.2.6.

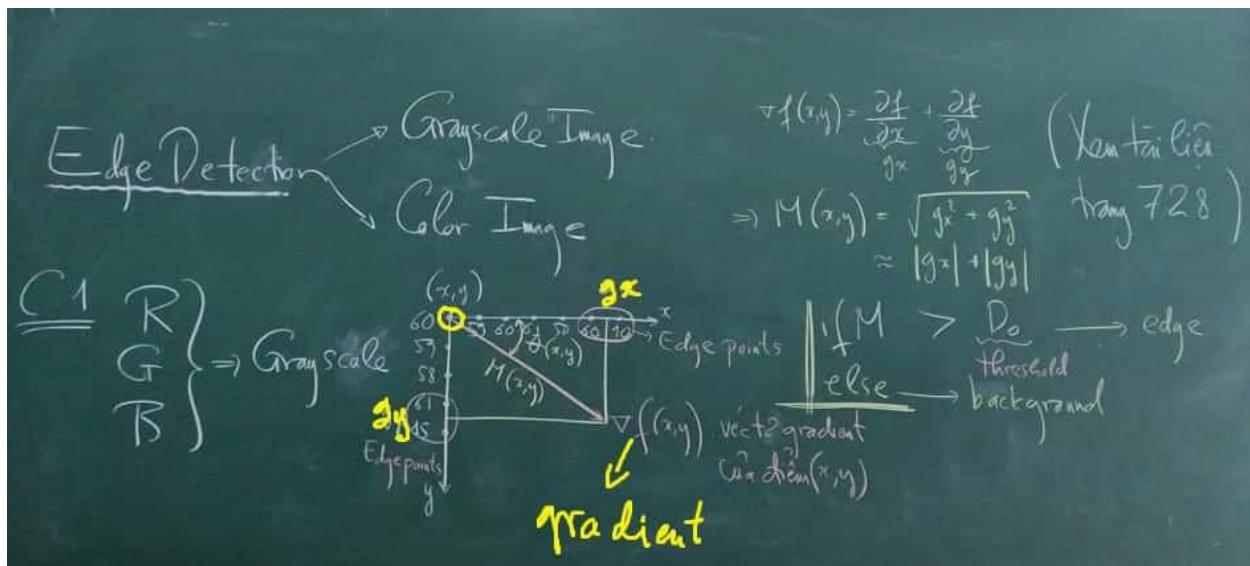
The image gradient and its properties

The tool of choice for finding edges is the

purpose of finding edges can be accomplished with derivatives. We discuss first-order derivatives in this section and second-order derivatives in Section 10.2.6.

The image gradient and its properties

The tool of choice for finding edge strength a



10.2.5 Basic Edge Detection

As illustrated in the previous section, detecting changes in intensity for the purpose of finding edges can be accomplished using first- or second-order derivatives. We discuss first-order derivatives in this section and work with second-order derivatives in Section 10.2.6.

The image gradient and its properties

The tool of choice for finding edge strength *and* direction at location (x, y) of an image, f , is the gradient, denoted by ∇f , and defined as the *vector*

For convenience, we repeat here some equations from Section 3.6.4.

$$\nabla f \equiv \text{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \quad (10.2-9)$$

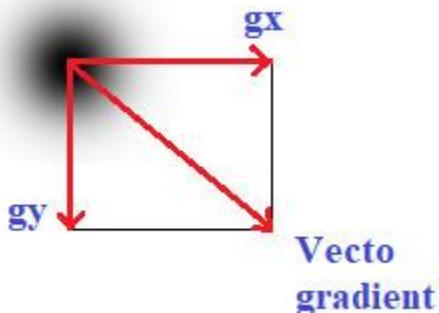
gradient this plus $\frac{\partial f}{\partial x}$
gradient this plus $\frac{\partial f}{\partial y}$

This vector has the important geometrical property that it points in the direction of the greatest rate of change of f at location (x, y) .

The magnitude (length) of vector ∇f , denoted as $M(x, y)$, where

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2} \quad (10.2-10)$$

So, similar magnitude to edge detection



The image gradient and its properties

The tool of choice for finding edge strength *and* direction at location (x, y) of an image, f , is the gradient, denoted by ∇f , and defined as the vector

$$\nabla f = \text{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \quad (10.2-9)$$

gradient của pixel theo phím x

Tính theo trục
hàm là trục y

phím y

This vector has the important geometrical property that it points in the direction of the greatest rate of change of f at location (x, y) .

The magnitude (length) of vector ∇f , denoted as $M(x, y)$, where

Bí quyết *vua gradient*

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2} \quad (10.2-10)$$

is the value of the rate of change in the direction of the gradient vector. Note that g_x , g_y , and $M(x, y)$ are images of the same size as the original, created when x and y are allowed to vary over all pixel locations in f . It is common practice to refer to the latter image as the *gradient image*, or simply as the *gradient* when the meaning is clear. The summation, square, and square root operations are *array operations*, as defined in Section 2.6.1.

if $(M(x, y) > D_0)$ \Rightarrow edge
ngưỡng (threshold)
else \Rightarrow background

The magnitude (length) of vector ∇f , denoted as $M(x, y)$, where

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2} \quad (10.2-10) \quad \times$$

$$M(x, y) \approx |g_x| + |g_y| \quad (10.2-20) \quad \checkmark$$

| Chi lấp trống thi sử dụng công thức 10.2-20
để tính M thay cho công thức 10.2-10
nhưng giảm tốn thời gian do CPU

\Rightarrow **Vău đê~**
 Tính g_x & g_y

$$g_x \rightarrow \frac{\partial f(x,y)}{\partial x}$$

$$g_y \rightarrow \frac{\partial f(x,y)}{\partial y}$$

Gradient operators

Obtaining the gradient of an image requires computing the partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ at every pixel location in the image. We are dealing with digital quantities, so a digital approximation of the partial derivatives over a neighborhood about a point is required. From Section 10.2.1 we know that

$$g_x = \frac{\partial f(x,y)}{\partial x} = f(x+1,y) - f(x,y) \quad (10.2-12)$$

and

$$g_y = \frac{\partial f(x,y)}{\partial y} = f(x,y+1) - f(x,y) \quad (10.2-13)$$

	$y-1$	y	$y+1$
$x-1$	30	6	7
x	3	(25)	8
$x+1$	10	7	7

$$\partial_x = f(x+1,y) - f(x,y) = -17$$

$$\partial_y = f(x,y+1) - f(x,y) = -16$$

$$\Rightarrow M(x,y) = \sqrt{|g_x| + |g_y|} = \sqrt{|-17| + |-16|} = 33$$

Sobel

S_x : ma trận Sobel theo phím x

$$\Rightarrow g_x$$

	$y-1$	y	$y+1$
$x-1$	30	6	7
x	3	(25)	8
$x+1$	10	7	7

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
1	0	1

S_y : ma trận Sobel theo phím y

$$\Rightarrow g_y$$

$$\begin{aligned} g_x &= 30 \times (-1) + 6 \times (-2) + 7 \times (-1) + 0 + 0 + \\ &10 \times (1) + 3 \times 2 + 7 \times 1 = -14 \\ g_y &= 30 \times (-1) + 6 \times 0 + 7 \times 1 + 3 \times (-2) + 25 \times 0 + \\ &8 \times 2 + 10 \times (-1) + 3 \times 0 + 7 \times 1 = -16 \end{aligned}$$

S_y : ma trận Sobel theo y

$$\Rightarrow M = \sqrt{|g_x| + |g_y|} = \sqrt{|-14| + |-16|} = 30$$

Yêu cầu: Code lần lượt với các măt nạ: SOBEL, PREWITT và ROBERTs:

-1	0
0	-1
0	1

Roberts

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Prewitt

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Sobel

Handwritten pseudocode for a nested loop structure:

```

for x: 0 ÷ (w-1)
  for y: 0 ÷ (h-1)
    {
      for i: (x) ÷ (x+1)
        for j: y ÷ (y+1)
          {
            g_x += p. R(i, j) * Rx [x - (i), y - (j)]
            g_y += p. Ry [x - i, y - j]
          }
    }
  
```

Diagram illustrating the Roberts operator:

The diagram shows two 3x3 kernel matrices. The top matrix is labeled $R(x)$ and has values: $x \quad x+1$, $0 \quad 0$, $-1 \quad 0$, $0 \quad 1$. The bottom matrix is labeled $R(y)$ and has values: $0 \quad -2$, $1 \quad 0$, $0 \quad 1$. A central pixel at position (i, j) is highlighted with a circle. Arrows point from the central pixel to the corresponding elements in both kernels.

Handwritten note: $R(x)$ and $R(y)$ are circled.

Sobel Operator: It is a discrete differentiation operator. It computes the gradient approximation of image intensity function for image edge detection. At the pixels of an image, the Sobel operator produces either the normal to a vector or the corresponding gradient vector. It uses two 3×3 kernels or masks which are convolved with the input image to calculate the vertical and horizontal derivative approximations respectively –

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Advantages:

1. Simple and time efficient computation
2. Very easy at searching for smooth edges

Limitations:

1. Diagonal direction points are not preserved always
2. Highly sensitive to noise
3. Not very accurate in edge detection
4. Detect with thick and rough edges does not give appropriate results

Prewitt Operator: This operator is almost similar to the sobel operator. It also detects vertical and horizontal edges of an image. It is one of the best ways to detect the orientation and magnitude of an image. It uses the kernels or masks –

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad M_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Advantages:

1. Good performance on detecting vertical and horizontal edges
2. Best operator to detect the orientation of an image

Limitations:

1. The magnitude of coefficient is fixed and cannot be changed
2. Diagonal direction points are not preserved always

Robert Operator: This gradient-based operator computes the sum of squares of the differences between diagonally adjacent pixels in an image through discrete differentiation. Then the gradient approximation is made. It uses the following 2×2 kernels or masks –

$$M_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad M_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

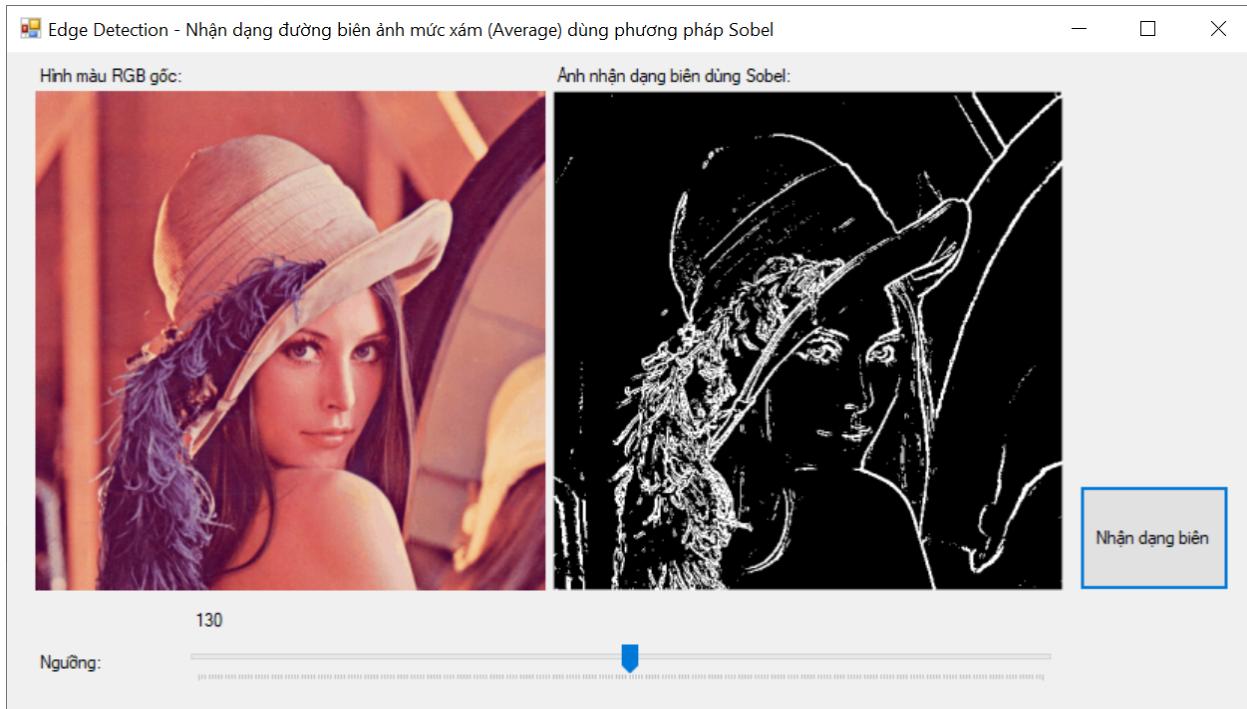
Advantages:

1. Detection of edges and orientation are very easy
2. Diagonal direction points are preserved

Limitations:

1. Very sensitive to noise
2. Not very accurate in edge detection

Kết quả C#.NET (phương pháp Sobel):



Kết quả Python (phương pháp Sobel):



MINI-PROJECT 15

Edge Detection - Nhận dạng đường biên dùng phương pháp Sobel cho ảnh màu RGB

- ⌚ Hiểu lý thuyết nhận dạng đường biên cho ảnh màu RGB
- ⌚ Lập trình nhận dạng đường biên cho ảnh màu RGB bằng C#.NET
- ⌚ Lập trình nhận dạng đường biên cho ảnh màu RGB bằng Python

 Videos bài giảng của thầy:

1. UTEX#18 - Thuật toán nhận dạng đường biên (Edge Detection) dùng phương pháp Sobel cho ảnh màu RGB
<https://youtu.be/S9RO8a5xAW0>

 CÁC EM XEM TÀI LIỆU MỤC 6.7.3 COLOR EDGE DETECTION TRANG 447, CÁC CÔNG THỨC 6.7-3 ĐẾN 6.7-9

 BÀI TOÁN: Giả sử xét điểm ảnh màu RGB $f(x, y)$, Edge Detection là đi tính xem điểm ảnh $f(x, y)$ là thuộc đường biên (edge) hay là nền (background).

 Tại điểm ảnh $f(x, y)$ chúng ta có thể trích xuất 3 giá trị $R(x, y)$ là giá trị màu kênh Red tại vị trí (x, y) , $G(x, y)$ là giá trị màu kênh Green tại vị trí (x, y) , và $B(x, y)$ là giá trị màu kênh Blue tại vị trí (x, y) .

 Như vậy, ứng với mỗi kênh màu R-G-B, chúng ta sẽ tính được các giá trị:

* Kênh R:

$$gx(R) = dR/dx$$

$$gy(R) = dR/dy$$

* Kênh G:

$$gx(G) = dG/dx$$

$$gy(G) = dG/dy$$

* Kênh B:

$$gx(B) = dB/dx$$

$$gy(B) = dB/dy$$

Tương tự như khi tính $[gx\ gy]$ đối với ảnh mức xám, chúng ta sẽ tính $[gx(R)\ gy(R)]$, $[gx(G)\ gy(G)]$ và $[gx(B)\ gy(B)]$ bằng cách nhân ma trận mặt nạ 3x3 các điểm ảnh hàng xóm của điểm đang xét tại vị trí (x, y) với ma trận Sobel theo phương x (để tính gx) và theo phương y (để tính gy).

 Tiếp theo, áp dụng các công thức (6.7-5) đến (6.7-7) để tính các giá trị gxx, gyy và gxy như sau:

$$gxx = |gx(R)|^2 + |gx(G)|^2 + |gx(B)|^2$$

$$gyy = |gy(R)|^2 + |gy(G)|^2 + |gy(B)|^2$$

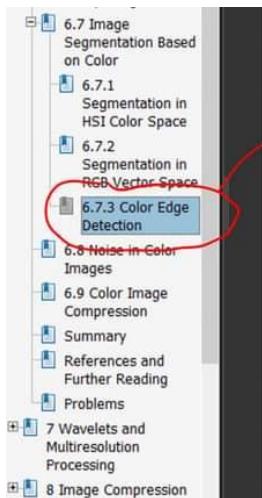
$$gxy = gx(R)*gy(R) + gx(G)*gy(G) + gx(B)*gy(B)$$

 Áp dụng công thức 6.7-8 để tính giá trị góc theta:

$$\theta(x, y) = [\arctan(2*gxy/(gxx-gyy))] / 2$$

Ap dụng công thức 6.7-9 để tính giá trị F0(x, y). Công thức này hơi dài nên thầy không gõ lại, các em xem trong tài liệu nha.

Cuối cùng, chúng ta so sánh giá trị F0 với một giá trị ngưỡng D0, nếu $F0 \leq D0$ thì điểm đang xét $f(x, y)$ thuộc background, ngược lại thì $f(x, y)$ thuộc edge.



6.7.3 Color Edge Detection

Trang 447

As discussed in Chapter 10, edge detection is an important tool for image segmentation. In this section, we are interested in the issue of computing edges on an individual-image basis versus computing edges directly in color vector space. The details of edge-based segmentation are given in Section 10.2.

Edge detection by gradient operators was introduced in Section 3.6.4 in connection with image sharpening. Unfortunately, the gradient discussed in Section 3.6.4 is not defined for vector quantities. Thus, we know immediately that computing the gradient on individual images and then using the results to form a color image will lead to erroneous results. A simple example will help illustrate the reason why.

6.7 ■ Image Segmentation Based on Color

449

Let \mathbf{r} , \mathbf{g} , and \mathbf{b} be unit vectors along the R , G , and B axis of RGB color space (Fig. 6.7), and define the vectors

and

$$\mathbf{u} = \frac{\partial R}{\partial x} \mathbf{r} + \frac{\partial G}{\partial x} \mathbf{g} + \frac{\partial B}{\partial x} \mathbf{b} \quad (6.7-3)$$

$$\mathbf{v} = \frac{\partial R}{\partial y} \mathbf{r} + \frac{\partial G}{\partial y} \mathbf{g} + \frac{\partial B}{\partial y} \mathbf{b} \quad (6.7-4)$$

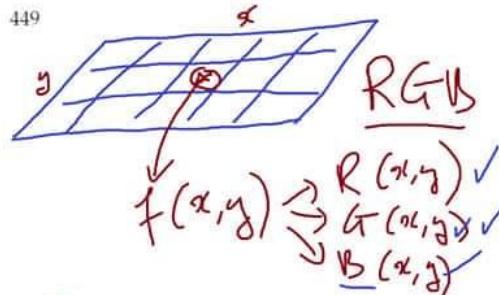
Let the quantities g_{xx} , g_{yy} , and g_{xy} be defined in terms of the dot product of these vectors, as follows:

$$g_{xx} = \mathbf{u} \cdot \mathbf{u} = \mathbf{u}^T \mathbf{u} = \left| \frac{\partial R}{\partial x} \right|^2 + \left| \frac{\partial G}{\partial x} \right|^2 + \left| \frac{\partial B}{\partial x} \right|^2 \quad (6.7-5)$$

and

$$g_{yy} = \mathbf{v} \cdot \mathbf{v} = \mathbf{v}^T \mathbf{v} = \left| \frac{\partial R}{\partial y} \right|^2 + \left| \frac{\partial G}{\partial y} \right|^2 + \left| \frac{\partial B}{\partial y} \right|^2 \quad (6.7-6)$$

$$g_{xy} = \mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v} = \frac{\partial R}{\partial x} \frac{\partial R}{\partial y} + \frac{\partial G}{\partial x} \frac{\partial G}{\partial y} + \frac{\partial B}{\partial x} \frac{\partial B}{\partial y} \quad (6.7-7)$$



$$g_{xx} \Rightarrow u \cdot u \quad (6.7.3) \quad g_{yy} \Rightarrow v \cdot v \quad (6.7.4)$$

$$\begin{cases} u = \frac{\partial R}{\partial x} + \frac{\partial G}{\partial x} + \frac{\partial B}{\partial x} \\ v = \frac{\partial R}{\partial y} + \frac{\partial G}{\partial y} + \frac{\partial B}{\partial y} \end{cases}$$

g_{xx} , g_{yy} & g_{xy}

$$g_{xx} = \mathbf{u} \cdot \mathbf{u} = \mathbf{u}^T \mathbf{u} = \left| \frac{\partial R}{\partial x} \right|^2 + \left| \frac{\partial G}{\partial x} \right|^2 + \left| \frac{\partial B}{\partial x} \right|^2 \quad (6.7-5)$$

$$g_{yy} = \mathbf{v} \cdot \mathbf{v} = \mathbf{v}^T \mathbf{v} = \left| \frac{\partial R}{\partial y} \right|^2 + \left| \frac{\partial G}{\partial y} \right|^2 + \left| \frac{\partial B}{\partial y} \right|^2 \quad (6.7-6)$$

$$g_{xy} = \mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v} = \frac{\partial R}{\partial x} \frac{\partial R}{\partial y} + \frac{\partial G}{\partial x} \frac{\partial G}{\partial y} + \frac{\partial B}{\partial x} \frac{\partial B}{\partial y} \quad (6.7-7)$$

Keep in mind that R , G , and B , and consequently the g 's, are functions of x and y . Using this notation, it can be shown (Di Zenzo [1986]) that the direction of maximum rate of change of $\mathbf{c}(x, y)$ is given by the angle

$$\theta(x, y) = \frac{1}{2} \tan^{-1} \left[\frac{2g_{xy}}{g_{xx} - g_{yy}} \right] \quad (6.7-8)$$

and that the value of the rate of change at (x, y) , in the direction of $\theta(x, y)$, is given by

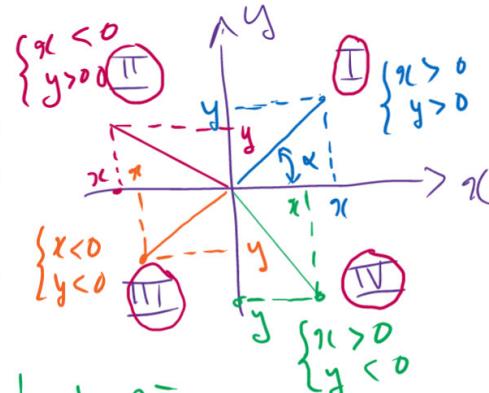
$$F_\theta(x, y) = \left\{ \frac{1}{2} [(g_{xx} + g_{yy}) + (g_{xx} - g_{yy}) \cos 2\theta(x, y) + 2g_{xy} \sin 2\theta(x, y)] \right\}^{\frac{1}{2}} \quad (6.7-9)$$

if ($F_\theta > D$) \rightarrow edge
 w^{20%} (threshold)
 else \rightarrow $\chi(x, y)$ to background

Hỗn: Dùng hàm atan2 trong C#
để tính θ :

$$\theta = \text{Math.atan2}(g_y, g_x)$$

$$\alpha = \text{atan2}(y, x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0, \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$



Trong hàm atan2 có phần chia các trường hợp góc nằm ở phần tử nào trong 4 góc phán tử, mặt phẳng \Rightarrow sẽ cho ra giá trị góc chính xác nhất.

Nếu dùng atan để tính θ
thì sẽ bị truy cập lỗi sin:

Vdu $\begin{cases} g_x > 0 \\ g_y > 0 \end{cases} \Rightarrow \theta$ sẽ ra góc

thuộc \textcircled{I} , nhưng nếu

$\begin{cases} g_x < 0 \\ g_y < 0 \end{cases} \Rightarrow \theta = \tan^{-1}\left(\frac{g_y}{g_x}\right)$

số âm

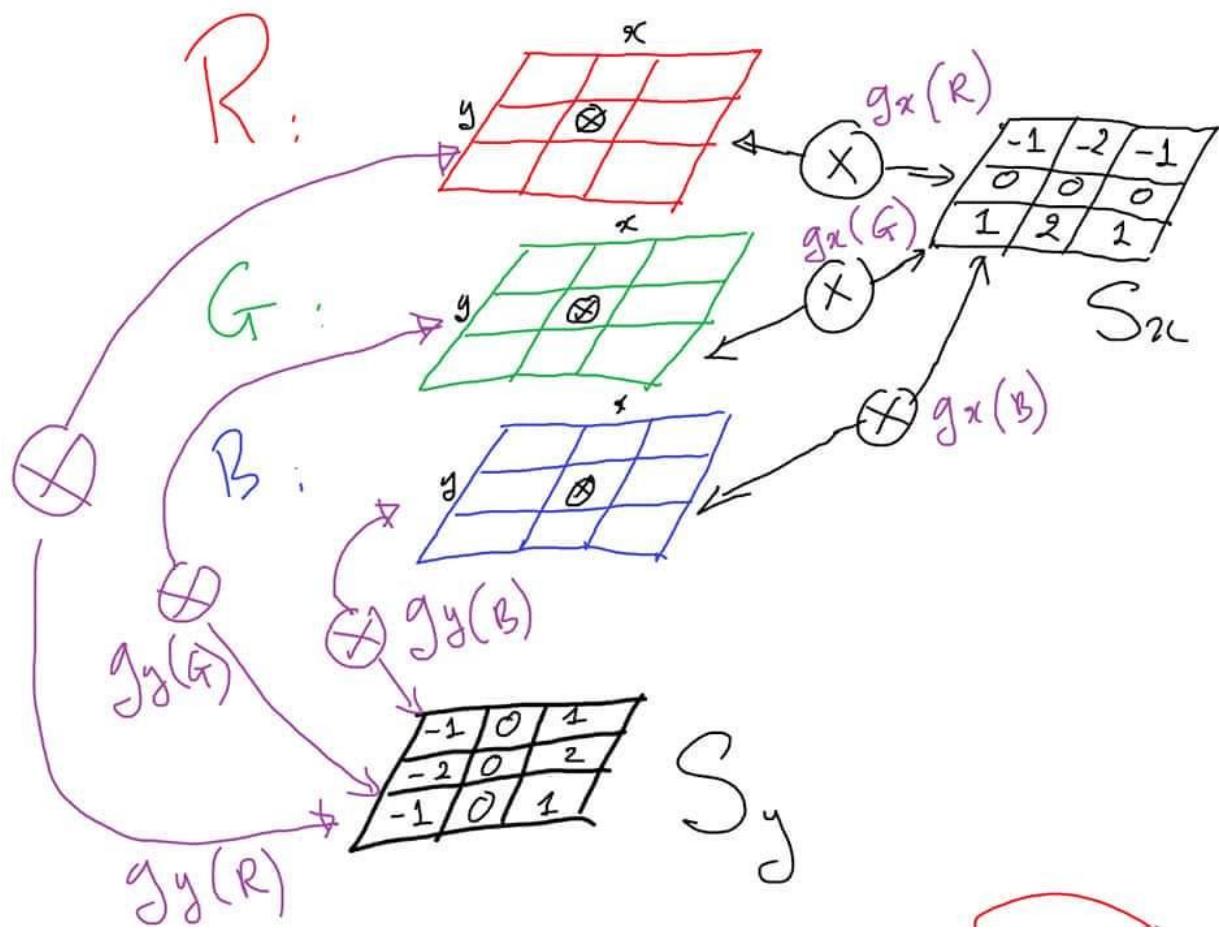
$\frac{\hat{a}}{\hat{a}}$ \rightarrow dương

$\Rightarrow \theta$ vẫn ra góc thuộc \textcircled{I}
trong khi thực tế là $\in \textcircled{III}$

XEM LINK SAU ĐỂ HIỂU RỎ HÀM ATAN2:

<https://en.wikipedia.org/wiki/Atan2>

$$\begin{array}{l}
 \left\{ \begin{array}{l} g_x(r) \\ g_x(G) \\ g_x(B) \end{array} \right. \rightarrow \left\{ \begin{array}{l} g_{xx} = |g_{xc}(r)|^2 + |g_{xG}(G)|^2 + |g_{xB}(B)|^2 \\ g_{yy} = |g_{yc}(r)|^2 + |g_{yG}(G)|^2 + |g_{yB}(B)|^2 \\ g_{xy} = g_{xc}(r) \times g_{yc}(r) + g_{xG}(G) \times g_{yG}(G) + \\ \quad g_{xB}(B) \times g_{yB}(B) \end{array} \right. \\
 \downarrow \\
 f(x,y) \rightarrow F_o(x,y) \Rightarrow \text{So. Sil} \\
 \text{u. j. d. j.}
 \end{array}$$



Q8.1

$$\begin{aligned} [R] \otimes [S_x] &= g_x(R) \\ [G] \otimes [S_x] &= g_x(G) \\ [B] \otimes [S_x] &= g_x(B) \end{aligned}$$

Sobel
theo
x

$$\begin{cases} g_x(R) \\ g_x(G) \\ g_x(B) \end{cases} = \begin{cases} g_x(R) \\ g_x(G) \\ g_x(B) \end{cases}$$

$$\begin{aligned} [R] \otimes [S_y] &= g_y(R) \\ [G] \otimes [S_y] &= g_y(G) \\ [B] \otimes [S_y] &= g_y(B) \end{aligned}$$

Sobel
theo
y

$$\begin{cases} g_y(R) \\ g_y(G) \\ g_y(B) \end{cases}$$

Côl 2 (Trang 469: Color Edge Detection)

$\begin{cases} R(x,y) \Rightarrow \nabla R(x,y) = \frac{\partial R}{\partial x} + \frac{\partial R}{\partial y} = g_x(R) + g_y(R) \\ G(x,y) \Rightarrow \nabla G(x,y) = \frac{\partial G}{\partial x} + \frac{\partial G}{\partial y} = g_x(G) + g_y(G) \\ B(x,y) \Rightarrow \nabla B(x,y) = \frac{\partial B}{\partial x} + \frac{\partial B}{\partial y} = g_x(B) + g_y(B) \end{cases}$

Color Image R/G/B

$\begin{matrix} 1 & 0 & 1 \\ -1 & 0 & -1 \\ 0 & 2 & 0 \end{matrix} S_x$

$\begin{matrix} 1 & 0 & 1 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{matrix} S_y$

$\begin{matrix} 1 & 0 & 1 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{matrix} S_z$

$\nabla R(x,y) = \nabla R(x,y) + \nabla G(x,y) + \nabla B(x,y)$

$\nabla R(x,y) = \sqrt{g_x(R)^2 + g_y(R)^2}$

$\nabla G(x,y) = \sqrt{g_x(G)^2 + g_y(G)^2}$

$\nabla B(x,y) = \sqrt{g_x(B)^2 + g_y(B)^2}$

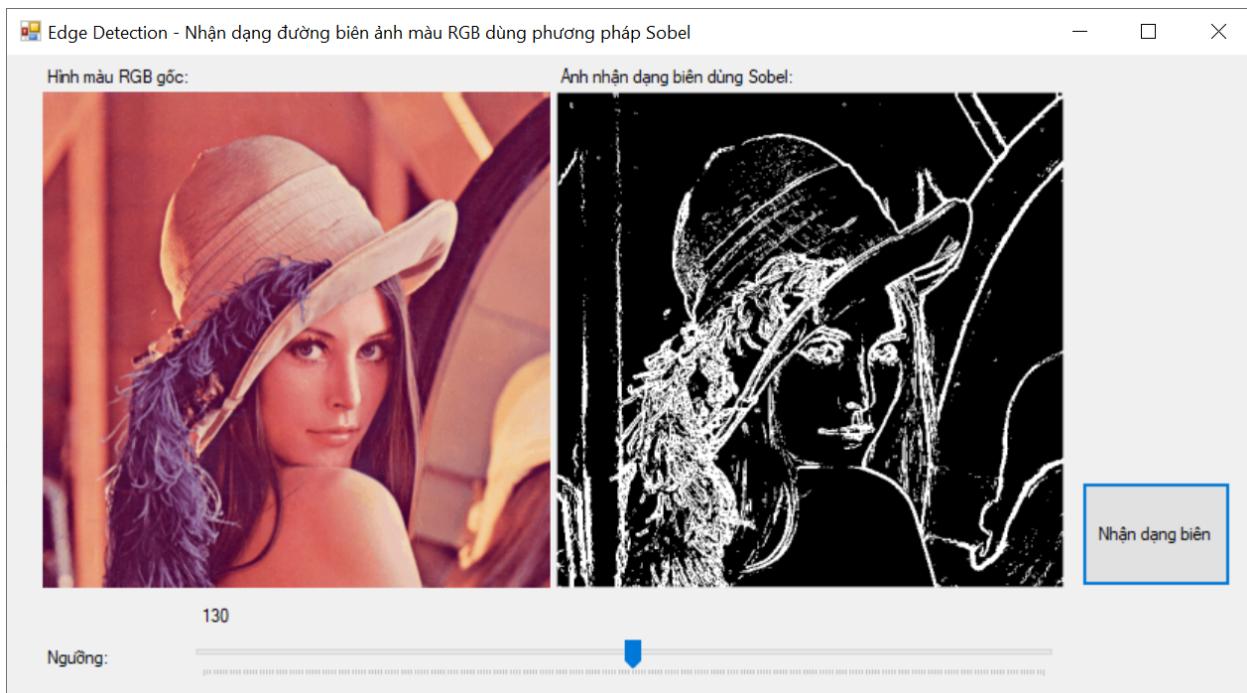
The formula CT (6,7-9) = $\sqrt{F_o(x,y)}$

Detailed Dotted edge

$\begin{cases} [R] \otimes [S_x] \rightarrow g_x(R) \\ [R] \otimes [S_y] \rightarrow g_y(R) \\ [G] \otimes [S_x] \rightarrow g_x(G) \\ [G] \otimes [S_y] \rightarrow g_y(G) \\ [B] \otimes [S_x] \rightarrow g_x(B) \\ [B] \otimes [S_y] \rightarrow g_y(B) \end{cases}$

$\begin{cases} g_{xx} = [g_x(R)]^2 + [g_x(G)]^2 + [g_x(B)]^2 \\ g_{yy} = [g_y(R)]^2 + [g_y(G)]^2 + [g_y(B)]^2 \\ g_{xy} = [g_x(R) \cdot g_y(R)] + [g_x(G) \cdot g_y(G)] + [g_x(B) \cdot g_y(B)] \end{cases}$

Kết quả C#.NET:



Kết quả Python:



CÁC PHƯƠNG PHÁP EDGE DETECTION NÂNG CAO

1. The Marr-Hildreth edge detector

714 Chapter 10 ■ Image Segmentation

reduced number of broken edges; for instance, compare the 45° edges in Figs. 10.20(a) and (b). Of course, edges whose intensity values were severely attenuated due to blurring (e.g., the edges in the tile roof) are likely to be totally eliminated by thresholding. We return to the problem of broken edges in Section 10.2.7.

10.2.6 More Advanced Techniques for Edge Detection

The edge-detection methods discussed in the previous section are based simply on filtering an image with one or more masks, with no provisions being made for edge characteristics and noise content. In this section, we discuss more advanced techniques that make an attempt to improve on simple edge-detection methods by taking into account factors such as image noise and the nature of edges themselves.

Marr 3x3 Gaussian		
$\frac{1}{16} \times$	3.5	Smoothing Spatial Filters
1	2	1
2	4	2
1	2	1

FIGURE 3.32 Two 3×3 smoothing (averaging) filter masks. The constant multiplier in front of each mask is equal to 1 divided by the sum of the values

The Marr-Hildreth edge-detection algorithm may be summarized as follows:

1. Filter the input image with an $n \times n$ Gaussian lowpass filter obtained by sampling Eq. (10.2-21).
2. Compute the Laplacian of the image resulting from Step 1 using, for example, the 3×3 mask in Fig. 10.4(a). [Steps 1 and 2 implement Eq. (10.2-25).]
3. Find the zero crossings of the image from Step 2.

STEP 3 \rightarrow Tính slope: $slope = \frac{1}{2} \sum |L(x,y)|$ $(w \times L)$ \rightarrow Cúi điểm ảnh sau Laplacian

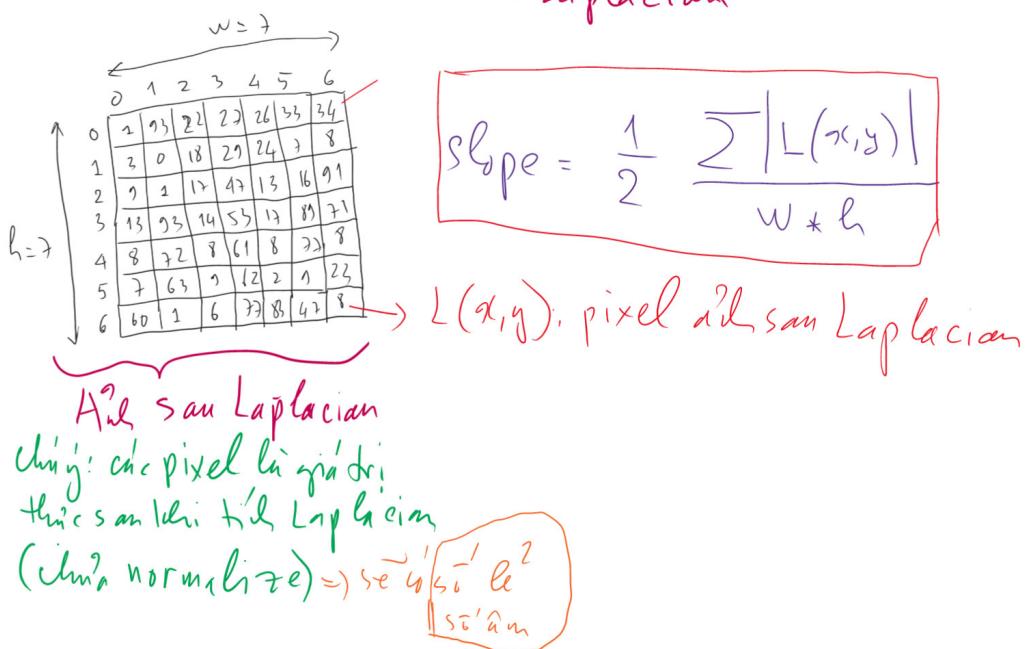
Xác định zero-crossing \rightarrow Kép thíc ảnh

if (① || ② || ③ || ④) \Rightarrow edge = 88

10.2 ■ Point, Line, and Edge Detection 697

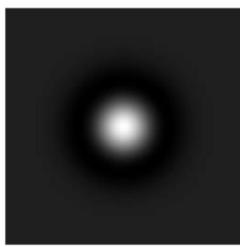
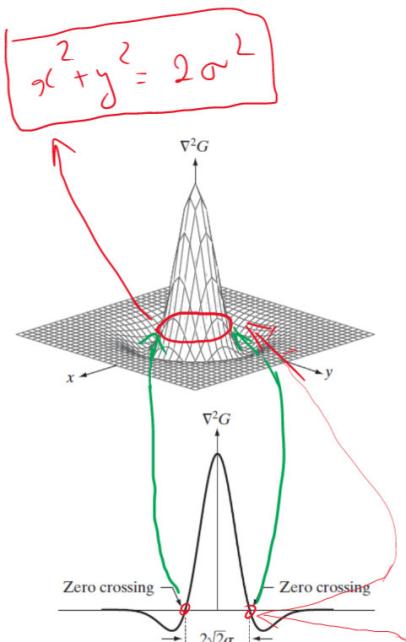
FIGURE 10.4
(a) Point detection (Laplacian) mask.
(b) X-ray image of turbine blade with a porosity. The porosity

* Tính slope ch. với
sau Laplacian



2. To find the slope of the image

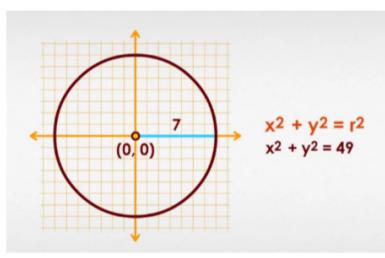
The absolute differences between the negative and positive values give the slope of the crossing, which is a measure of the strength of the edges. Slope is a very important parameter in implementing edge detection algorithms because it helps to find the strongest edges that are needed.



a b
c d

FIGURE 10.21
 (a) Three-dimensional plot of the negative of the LoG. (b) Negative of the LoG displayed as an image. (c) Cross section of (a) showing zero crossings. (d) 5×5 mask approximation to the shape in (a). The negative of this mask would be used in practice.

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0



Collecting terms gives the final expression:

$$\nabla^2 G(x, y) = \left[\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right] e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (10.2-23)$$

This expression is called the *Laplacian of a Gaussian* (LoG).

Figures 10.21(a) through (c) show a 3-D plot, image, and cross section of the negative of the LoG function (note that the zero crossings of the LoG occur at $x^2 + y^2 = 2\sigma^2$, which defines a circle of radius $\sqrt{2}\sigma$ centered on the origin).

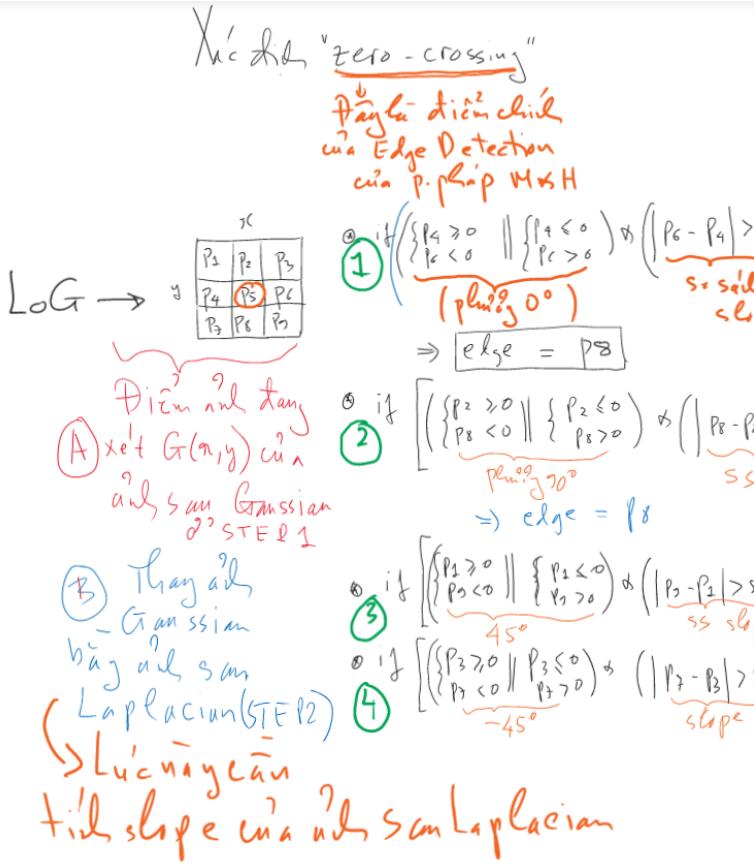
zero-crossing xia'f lin +
duīg drin

duīg drin

3. To look for zero crossing

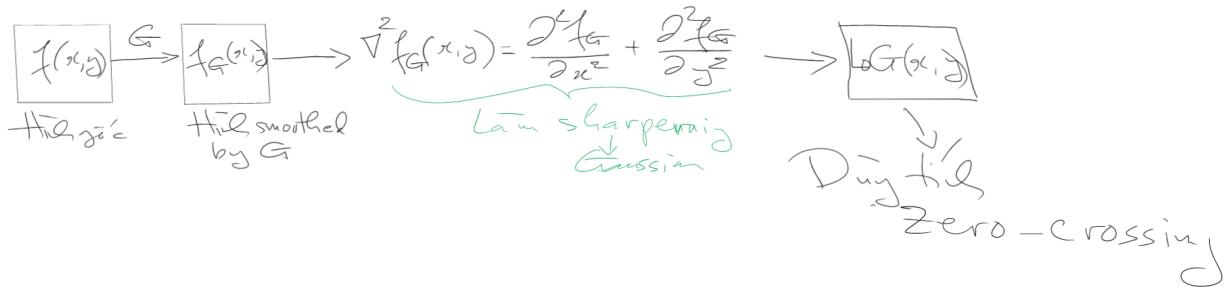
After applying LOG and make the image blurred and free of noises, it's time to find the edges which happen when there are changes in brightness of that image. There is a technique that we can measure these changes which is called zero crossing. Zero crossings occur wherever a positive value is followed by a negative value, or vice versa.

Or when there is zero in between a positive and negative value. Find this pattern help to find edges on the image.



⑤ Thúc:
 if $(1) \parallel (2) \parallel (3) \parallel (4)$
 $\Rightarrow \text{edge} = P_8$

TOÀN BỘ CÁC BƯỚC CỦA THUẬT TOÁN EDGE DETECTOR THEO MARR-HILDRETH:



2. The Canny edge detector

10.2 ■ Point, Line, and Edge Detection 719

The Canny edge detector

Although the algorithm is more complex, the performance of the Canny edge detector (Canny [1986]) discussed in this section is superior in general to the edge detectors discussed thus far. Canny's approach is based on three basic objectives:

1. *Low error rate.* All edges should be found, and there should be no spurious responses. That is, the edges detected must be as close as possible to the true edges.
2. *Edge points should be well localized.* The edges located must be as close as possible to the true edges. That is, the distance between a point marked as an edge by the detector and the center of the true edge should be minimum.
3. *Single edge point response.* The detector should return only one point for each true edge point. That is, the number of local maxima around the true edge should be minimum. This means that the detector should not identify multiple edge pixels where only a single edge point exists.

Summarizing, the Canny edge detection algorithm consists of the following basic steps:

1. Smooth the input image with a Gaussian filter.
2. Compute the gradient magnitude and angle images.
3. Apply nonmaxima suppression to the gradient magnitude image.
4. Use double thresholding and connectivity analysis to detect and link edges.

STEP 3

Lý giải: ?
STEP 2 với phím duy nhất sau Gaussian
(phím giữ nguyên giá trị sau tệp Gaussian
chỉ khi phím giá trị normalized)

STEP 1:

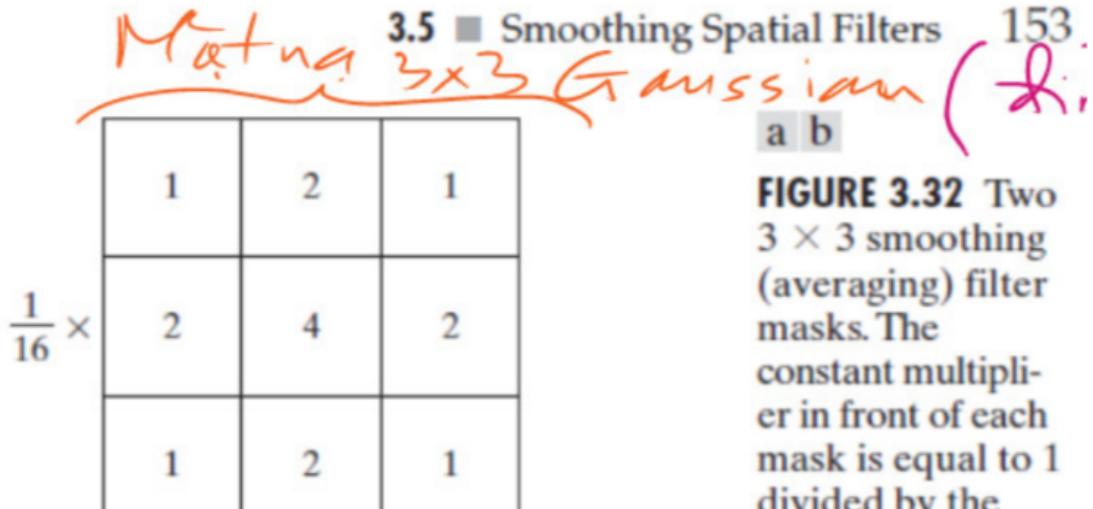


FIGURE 3.32 Two 3×3 smoothing (averaging) filter masks. The constant multiplier in front of each mask is equal to 1 divided by the sum of the values

STEP 2

$$\left\{ \begin{array}{l} g_x \rightarrow \frac{\partial f(x,y)}{\partial x} \\ g_y \rightarrow \frac{\partial f(x,y)}{\partial y} \end{array} \right.$$

Gradient operators

Obtaining the gradient of an image requires computing the partial derivatives $\partial f / \partial x$ and $\partial f / \partial y$ at every pixel location in the image. We are dealing with digital quantities, so a digital approximation of the partial derivatives over a neighborhood about a point is required. From Section 10.2.1 we know that

$$g_x = \frac{\partial f(x,y)}{\partial x} = f(x+1,y) - f(x,y) \quad (10.2-12)$$

and

$$g_y = \frac{\partial f(x,y)}{\partial y} = f(x,y+1) - f(x,y) \quad (10.2-13)$$

$y-1$	$x-1$	x	$x+1$
$y-1$	30	6	7
y	3	25	8
$y+1$	10	7	7

$$\partial x = f(x+1,y) - f(x,y) = -17$$

$$\partial y = f(x,y+1) - f(x,y) = -16$$

$$\Rightarrow M(x,y) = \sqrt{|g_x|^2 + |g_y|^2} = \sqrt{|-17|^2 + |-16|^2} = 33$$

$$\hookrightarrow \theta = \tan^{-1}\left(\frac{g_y}{g_x}\right)$$

STEP 2 có thể dùng Sobel để tính $\begin{bmatrix} g_x \\ g_y \end{bmatrix}$

$$S_x: matrița Sobel theo phương pháp \Rightarrow g_x$$

$$S_y: matrița Sobel theo y \Rightarrow g_y$$

$$g_x = 30 \times (-1) + 6 \times (-2) + 7 \times (-1) + 0 + 0 + 10 \times (1) + 9 \times 2 + 7 \times 1 = -14$$

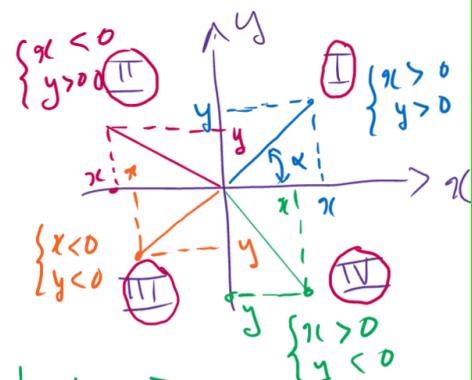
$$g_y = 36 \times (-1) + 6 \times 0 + 7 \times 1 + 3 \times (-2) + 25 \times 0 + 8 \times 2 + 10 \times (-1) + 9 \times 0 + 7 \times 1 = -16$$

$$\Rightarrow M = \sqrt{|g_x|^2 + |g_y|^2} = \sqrt{(-14)^2 + (-16)^2} = 30$$

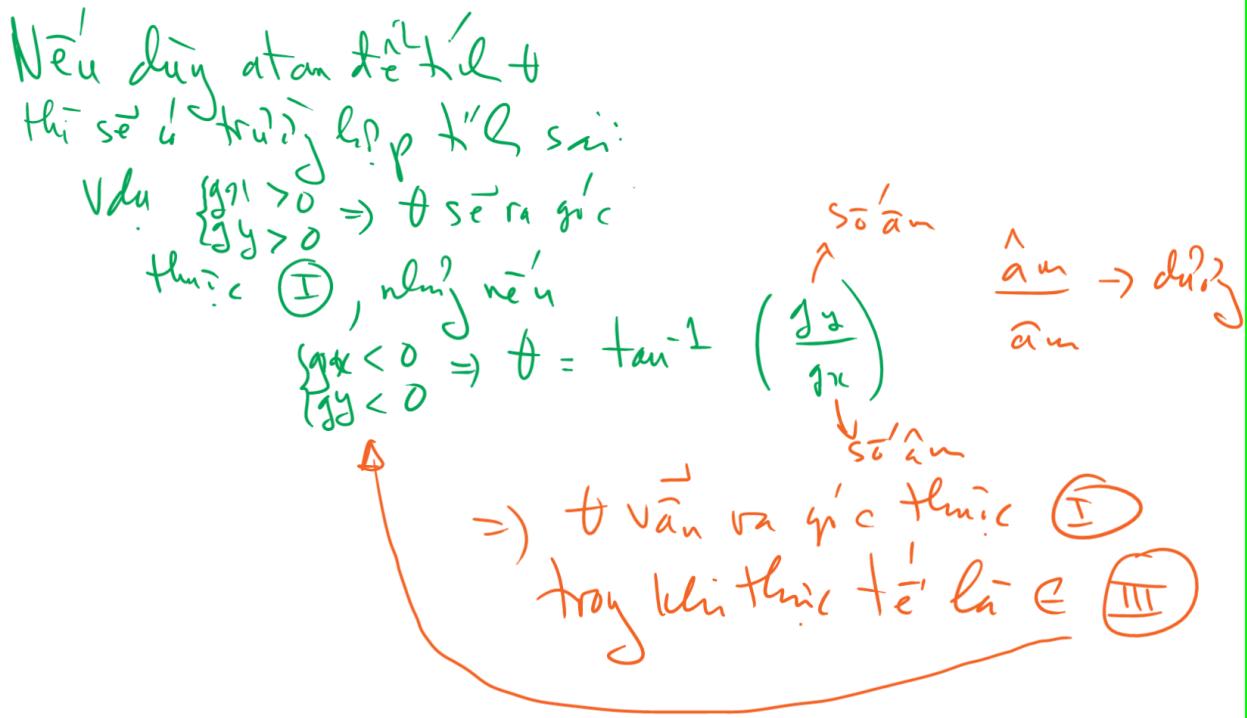
lưu ý: Dùng hàm atan2 trong C# để tính θ :

$$\theta = \text{Math.atan2}(g_y, g_x)$$

$$\theta = \text{atan2}(y, x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0, \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$



Trong hàm atan2 có phần chia các trường hợp góc thuộc phân trai nào trong 4 góc phân trai, mặt phẳng \Rightarrow sẽ cho ra giá trị góc chính xác nhất.

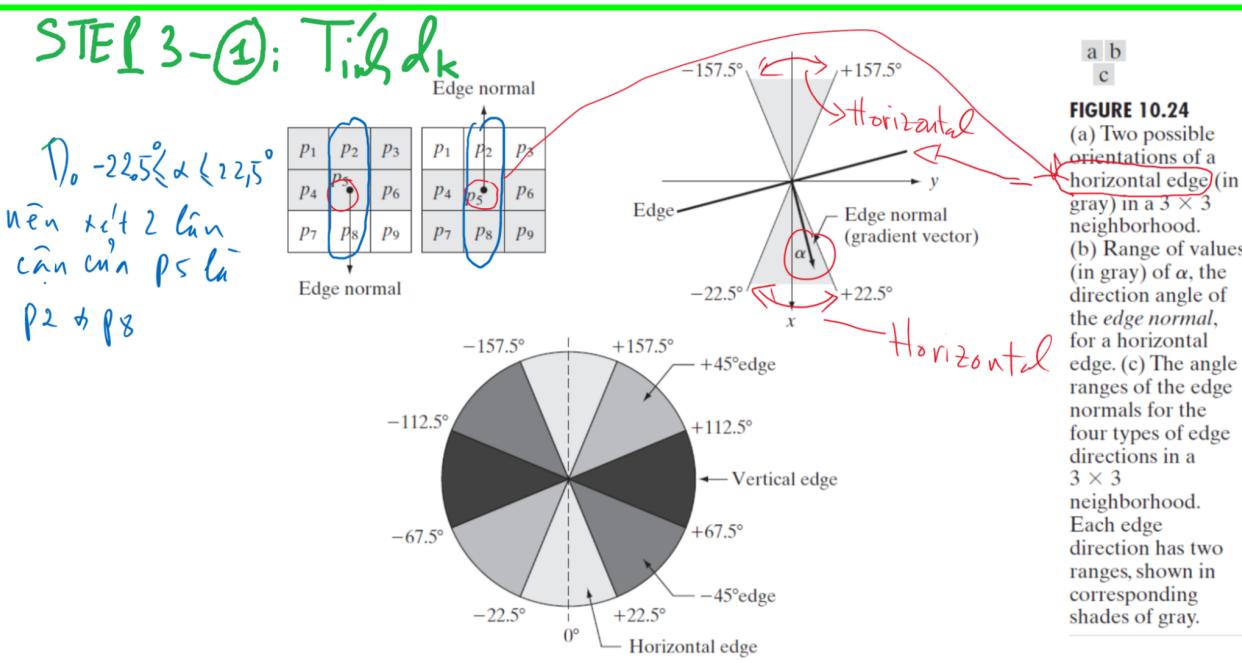


STEP 3 \rightarrow có 2 bước nhau $\textcircled{1} \rightarrow \textcircled{2}$:

Let d_1, d_2, d_3 , and d_4 denote the four basic edge directions just discussed for a 3×3 region: horizontal, -45° , vertical, and $+45^\circ$, respectively. We can formulate the following nonmaxima suppression scheme for a 3×3 region centered at every point (x, y) in $\alpha(x, y)$:

1. Find the direction d_k that is closest to $\alpha(x, y)$.
2. If the value of $M(x, y)$ is less than at least one of its two neighbors along d_k , let $g_N(x, y) = 0$ (suppression); otherwise, let $g_N(x, y) = M(x, y)$

Tiếp theo g N



Doan code (Python) tinh dk:

```
for x in range(width):
    for y in range(height):
        if (sobeloutdir[x][y]<22.5 and sobeloutdir[x][y]>=0) or \
           (sobeloutdir[x][y]>=157.5 and sobeloutdir[x][y]<202.5) or \
           (sobeloutdir[x][y]>=337.5 and sobeloutdir[x][y]<=360):
            sobeloutdir[x][y]=0 → dk = 0° (horizontal)
        elif (sobeloutdir[x][y]>=22.5 and sobeloutdir[x][y]<67.5) or \
              (sobeloutdir[x][y]>=202.5 and sobeloutdir[x][y]<247.5):
            sobeloutdir[x][y]=45 → dk = 45° (chéo 45°)
        elif (sobeloutdir[x][y]>=67.5 and sobeloutdir[x][y]<112.5)or \
              (sobeloutdir[x][y]>=247.5 and sobeloutdir[x][y]<292.5):
            sobeloutdir[x][y]=90 → dk = 90° (vertical)
        else:
            sobeloutdir[x][y]=135 → dk = 135° (chéo -45°)
```

Fázal g_N

2. If the value of $M(x, y)$ is less than at least one of its two neighbors along d_k , let $g_N(x, y) = 0$ (suppression); otherwise, let $g_N(x, y) = M(x, y)$

```
for x in range(1, width-1):
    for y in range(1, height-1):
        if sobeloutdir[x][y]==0:
            if (sobeloutmag[x][y]<=sobeloutmag[x][y+1]) or \
               (sobeloutmag[x][y]<=sobeloutmag[x][y-1]):
                mag_sup[x][y]=0
        elif sobeloutdir[x][y]==45:
            if (sobeloutmag[x][y]<=sobeloutmag[x-1][y+1]) or \
               (sobeloutmag[x][y]<=sobeloutmag[x+1][y-1]):
                mag_sup[x][y]=0
        elif sobeloutdir[x][y]==90:
            if (sobeloutmag[x][y]<=sobeloutmag[x+1][y]) or \
               (sobeloutmag[x][y]<=sobeloutmag[x-1][y]):
                mag_sup[x][y]=0
        else:
            if (sobeloutmag[x][y]<=sobeloutmag[x+1][y+1]) or \
               (sobeloutmag[x][y]<=sobeloutmag[x-1][y-1]):
                mag_sup[x][y]=0
```

Tiếp theo là bài gNL và gNH :

Canny's algorithm

attempts to improve on this situation by using *hysteresis thresholding* which, as we discuss in Section 10.3.6, uses two thresholds: a low threshold, T_L and a high threshold, T_H . Canny suggested that the ratio of the high to low threshold should be two or three to one.

We can visualize the thresholding operation as creating two additional images

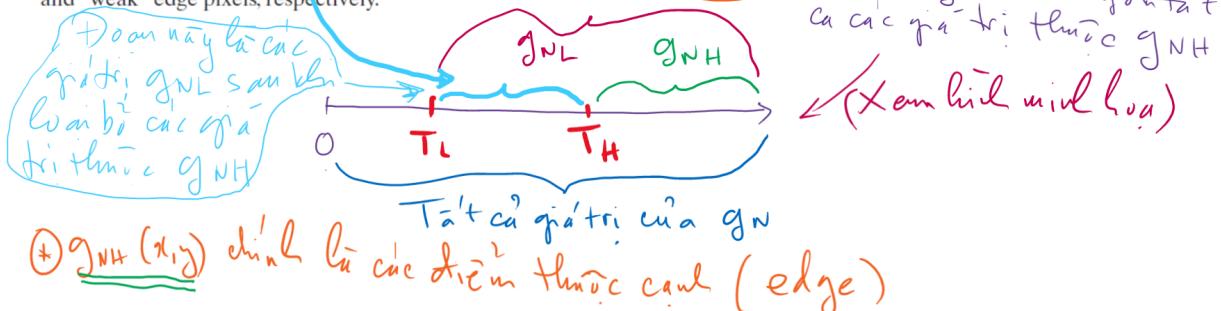
$$g_{NH}(x, y) = g_N(x, y) \geq T_H \quad (10.2-33)$$

$$\text{and} \quad g_{NL}(x, y) = g_N(x, y) \geq T_L \quad (10.2-34)$$

where, initially, both $g_{NH}(x, y)$ and $g_{NL}(x, y)$ are set to 0. After thresholding, $g_{NH}(x, y)$ will have fewer nonzero pixels than $g_{NL}(x, y)$ in general, but all the nonzero pixels in $g_{NH}(x, y)$ will be contained in $g_{NL}(x, y)$ because the latter image is formed with a lower threshold. We eliminate from $g_{NL}(x, y)$ all the nonzero pixels from $g_{NH}(x, y)$ by letting

$$g_{NL}(x, y) = g_{NL}(x, y) - g_{NH}(x, y) \quad (10.2-35)$$

The nonzero pixels in $g_{NH}(x, y)$ and $g_{NL}(x, y)$ may be viewed as being "strong" and "weak" edge pixels, respectively.



$$g_{NH}(x, y) = g_N(x, y) \geq T_H \quad (10.2-33)$$

$$\text{and} \quad g_{NL}(x, y) = g_N(x, y) \geq T_L \quad (10.2-34)$$

where, initially, both $g_{NH}(x, y)$ and $g_{NL}(x, y)$ are set to 0.

```
m = numpy.max(mag_sup)
th = 0.2*m
tl = 0.1*m

gnh = numpy.zeros((width, height))
gnl = numpy.zeros((width, height))

for x in range(width):
    for y in range(height):
        if mag_sup[x][y]>=th:
            gnh[x][y]=mag_sup[x][y]
        if mag_sup[x][y]>=tl:
            gnl[x][y]=mag_sup[x][y]

gnl = gnl-gnh
```

Tạo 2 image rỗng
vì set all pixel = 0
 $\begin{cases} g_{NH}(x,y) = 0 \\ g_{NL}(x,y) = 0 \end{cases}$

Đoạn code (python)
tính $g_{NH} \oplus g_{NL}$

After the thresholding operations, all strong pixels in $g_{NH}(x, y)$ are assumed to be valid edge pixels and are so marked immediately. Depending on the value of T_H , the edges in $g_{NH}(x, y)$ typically have gaps. Longer edges are formed using the following procedure:

- (a) Locate the next unvisited edge pixel, p , in $g_{NH}(x, y)$.
- (b) Mark as valid edge pixels all the weak pixels in $g_{NL}(x, y)$ that are connected to p using, say, 8-connectivity.
- (c) If all nonzero pixels in $g_{NH}(x, y)$ have been visited go to Step d. Else, return to Step a.
- (d) Set to zero all pixels in $g_{NL}(x, y)$ that were not marked as valid edge pixels.

At the end of this procedure, the final image output by the Canny algorithm is formed by appending to $g_{NH}(x, y)$ all the nonzero pixels from $g_{NL}(x, y)$.

Doan code (Python):

H
am
de qui

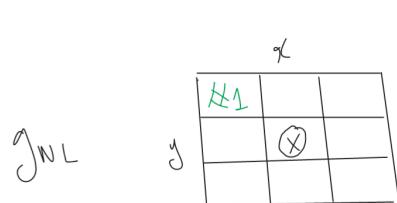
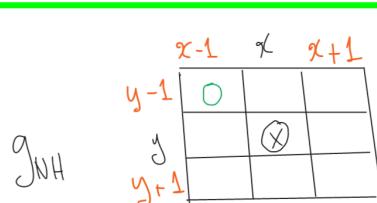
```
def traverse(i, j):
    x = [-1, 0, 1, -1, 1, 1, 0, 1]
    y = [-1, -1, -1, 0, 0, 1, 1, 1]
    for k in range(8):
        if gnh[i+x[k]][j+y[k]]==0 and gnl[i+x[k]][j+y[k]]!=0:
            gnh[i+x[k]][j+y[k]]=1
            traverse(i+x[k], j+y[k])

    for i in range(1, width-1):
        for j in range(1, height-1):
            if gnh[i][j]:
                gnh[i][j]=1
                traverse(i, j)
```

-1	0	1
-1	-1	-1
0	X	1

8-connectivity

④ $\underbrace{k=0}_{\text{H}} : \begin{cases} g_{NH}[i-1, j-1] & ? 0 : \cancel{\times} 0, \text{if } 0 \\ g_{NL}[i-1, j-1] & ? 0 : \cancel{\times} 0 \end{cases} \Rightarrow \begin{cases} 1 & \rightarrow g_{NH}(i-1, j-1)=1 \\ 1 & \downarrow L \end{cases}$
 $\text{traverse}(i-1, j-1);$



$\Rightarrow \begin{cases} g_{NH}=0 \\ g_{NL} \neq 0 \end{cases} \Rightarrow \text{Edge}$

\downarrow

$\boxed{g_{NH}=1}$

CANNY EDGE DETECTOR:



**PHẦN MỞ RỘNG, NHỮNG EM NÀO HOÀN
THÀNH ĐỦ PHẦN NÀY SẼ ĐƯỢC CỘNG THÊM
15% ĐIỂM TÍCH LŨY QUÁ TRÌNH**

🌹 Ex-01 Lập trình bằng Python phân đoạn ảnh
bằng Histogram

🌹 Ex-02 Wavelets and Multiresolution Processing
(Chương 7, trang 461 trong sách)

🎯 Hiểu thuật toán Wavelets.

🎯 Lập trình bằng Python thuật toán Wavelets.

Ex-03 Morphological Image Processing (Chương 9, trang 627 trong sách)

⦿ Hiểu các thuật toán Morphological Image Processing.

⦿ Lập trình bằng Python các thuật toán Morphological Image Processing.

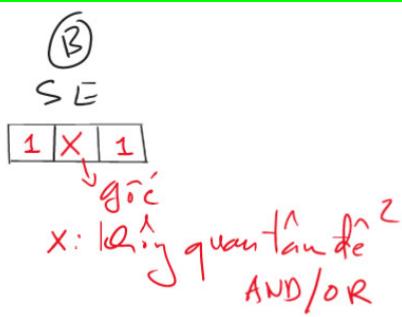
Ví dụ 1 tính Erosion & Dilation:

$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$	 → gốc của mask
(A) A^{NL} $\textcircled{1}$ Erosion (AND) Gỗm	(B) SE: Structuring Element Do milti chon $\textcircled{2}$ NM SM FM $\text{FM: Full match } \rightarrow \textcircled{1}$ $\text{SM: Some match } \rightarrow 0$ $\text{NM: No match } \rightarrow 0$
$\textcircled{1}$ Erosion (AND) $\rightarrow A - B$ $\text{FM: Full match } \rightarrow \textcircled{1}$ $\text{SM: Some match } \rightarrow 0$ $\text{NM: No match } \rightarrow 0$	
$\textcircled{2}$ <u>Dilation (OR)</u> : $A \oplus B$ $\left\{ \begin{array}{l} \text{FM} \rightarrow \textcircled{1} \\ \text{SM} \rightarrow 0 \\ \text{NM} \rightarrow \textcircled{0} \end{array} \right.$	

Ví dụ 2 tính Erosion & Dilation:

0	0	0	0	0	0
0	1	1	1	0	0
0	1	0	1	0	0
0	1	0	1	1	0
0	0	0	0	1	1
0	0	0	0	0	0

(*) Erosion:



0	0	0	0	0	0
0	1	X	1	0	0
0	1	X	1	0	0
0	1	X	1	1	0
0	0	0	0	1	1
0	0	0	0	0	0

Erosion:

0	0	0	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

(*) Dilation:

0	0	0	0	0	0
0	1	X	1	1	0
0	1	0	1	0	0
0	1	X	1	1	0
0	0	0	0	1	1
0	0	0	0	0	0

Dilation

0	0	0	0	0	0
1	1	1	1	0	0
1	0	1	0	1	0
1	0	1	1	1	1
0	0	0	1	1	1
0	0	0	0	0	0

Ví dụ 3 tính Erosion & Dilation:

(A)

0	0	0	0	0	0
0	1	1	1	0	
0	1	1	1	0	
0	1	1	1	0	
0	0	0	0	0	

SE

0	1	0
1	1	1
0	1	0

góc /

Chú ý:

SE: mảng đ小姑娘
tâm ô 1, khung
AND/OR XOR

④ Erosion: A ⊖ B

0	0	0	0	0	0
0	0	1	0	0	
0	1	1	1	0	
0	0	1	0	0	
0	0	0	0	0	

①
Erosion

0	0	0	0	0	0
0	0	0	0	0	0
0	0	1	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

⑤ Dilation: A ⊕ B

0	0	0	1	0	0
0	1	1	1	0	0
0	1	0	1	0	0
0	1	1	1	0	
0	0	0	0	0	

①
Dilation

0	1	1	1	0	
1	1	1	1	1	
1	1	1	1	1	
1	1	1	1	1	
0	1	1	1	0	

YÊU CẦU ĐỐI VỚI FINAL-PROJECT:

🌹 Mỗi nhóm thực hiện một video từ A-Z về FINAL-PROJECT của nhóm mình, mỗi thành viên đều góp giọng vào trong video. Thầy sẽ gửi link để các em nộp.

🌹 Khi chạy chương trình, màn hình giao diện đầu tiên sẽ bao gồm các thông tin sau:

- Logo trường
- Tên ngành
- Khoa
- Tên đề tài
- Tên GVHD: TS. Nguyễn Văn Thái
- Tên các SVTH
- Cuối cùng là nút bấm để chuyển đến giao diện chương trình chính của các em.

👉 Mỗi em đều phải góp giọng thuyết trình trong video của nhóm mình.

❤️ CÁC EM LƯU Ý THÀY SẼ PUBLIC ONLINE TẤT CẢ CÁC PROJECT CỦA CÁC EM, DO VẬY CẦN PHẢI LÀM VIDEO CHO ĐÀNG HOÀNG VÌ ĐÓ SẼ LÀ THƯƠNG HIỆU CỦA CÁC EM, THẬM CHÍ SAU NÀY ĐI XIN VIỆC CÁC EM CÓ THỂ SỬ DỤNG VIDEO ONLINE NÀY ĐỂ SHOW CHO NHÀ TUYỂN DỤNG THÁY.

🍁 Các nhóm submit các files trong FINAL PROJECT của mình theo như format thầy giải thích ở dưới.

🍁 Mỗi nhóm tự tạo một thư mục cho nhóm mình, rồi upload các file vào thư mục của nhóm theo qui cách như sau:

STT

|---STT.doc/pdf
|---STT-Code.rar/zip
|---STT.mp4
|---STT.jpg/png

Ví dụ: Nhóm ở vị trí số thứ tự 3 trong danh sách các nhóm, các em sẽ tạo thư mục trong Google Drive của thầy với tên như sau:

03

|---03.doc/pdf
|---03-Code.rar/zip
|---03.mp4
|---03.png

🍁 Những nhóm có số thứ tự từ 1-9 cần phải thêm số 0 phía trước nha.

🍁 03.doc/pdf là file báo cáo WORD hoặc PDF

🍁 03-code.rar/zip là file nén chứa tất cả files code trên Python hoặc C#.Net

🍁 03.mp4 là file video của nhóm (video được sản xuất theo qui cách thầy đã yêu cầu trước đó)

🍁 03.png là file hình đại diện cho dự án của nhóm (là hình ảnh xuất hiện đầu tiên khi mở YouTube, nhằm thu hút sự quan tâm của người xem khi xem list video trên YouTube)

⏰ 🌟 🦅 CHÚ Ý QUAN TRỌNG: CÁC NHÓM BẮT BUỘC PHẢI LÀM THEO ĐÚNG QUI CÁCH ĐẶT TÊN NHƯ TRÊN ĐỂ THÀY DỄ DÀNG TRUY XUẤT ĐÚNG NHÓM VÀ CHẤM ĐIỂM VÀ RÁP ĐÚNG ĐIỂM CHO NHÓM, NẾU NHÓM NÀO LÀM SAI HOẶC KHÔNG CÓ SẼ KHÔNG CÓ PHẦN ĐIỂM CHO PHẦN NÀY.