TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT

THỊ GIÁC MÁY

BÀI TẬP THỊ GIÁC MÁY

Sinh viên thực hiện: Nguyễn Hoàng Long **Lớp: DCCDTD68B**

Giảng viên hướng dẫn: GVC.TS Đoàn Công Luận

Hà Nội, 01/2026

MỤC LỤC

LỜI NÓI ĐẦU

Trong quá trình làm bài tập này, em đã tìm hiểu thêm tìm hiểu thêm các tài liệu chuyên ngành để hoàn thiện bài tập một cách chỉn chu nhất.

Tuy nhiên, do kiến thức và kinh nghiệm thực tế còn hạn chế, bài tập chắc chắn không tránh khỏi những thiếu sót. Em rất mong nhận được những lời nhận xét và góp ý quý báu của Thầy để em có thể rút kinh nghiệm và hoàn thiện bản thân trong các bài tập tiếp theo.

Em xin chân thành cảm ơn Thầy!

CHƯƠNG 1. GIỚI THIỆU

1.1 Khái niệm

Trong phần này, em sẽ nhắc lại và đi qua các khái niệm chung của bài toán.

Xử lý ảnh: Là việc sử dụng các thuật toán máy tính để thực hiện xử lý trên các ảnh kỹ thuật số. Đầu vào của quá trình là một bức ảnh và đầu ra có thể là một bức ảnh đã được cải thiện hoặc các thông số đặc trưng của ảnh đó. Theo định nghĩa của Gonzalez và Woods, một bức ảnh số có thể được biểu diễn dưới dạng một hàm hai chiều $f(x, y)$, trong đó x và y là tọa độ không gian, và giá trị của f tại bất kỳ cặp tọa độ nào được gọi là cường độ hoặc mức xám của điểm ảnh đó [1].

Thị giác máy: Là lĩnh vực khoa học liên ngành đề cập đến cách máy tính có thể đạt được sự hiểu biết cao cấp từ hình ảnh hoặc video kỹ thuật số. Nếu Xử lý ảnh tập trung vào việc biến đổi ảnh, thì Thị giác máy tính tập trung vào việc phân tích ảnh để ra quyết định [2].

1.2. Tại sao cần xử lý ảnh?

Đơn giản hóa dữ liệu: Một bức ảnh màu chuẩn chứa lượng thông tin rất lớn và dư thừa. Ví dụ, để nhận diện thông tin về màu sắc thường gây nhiễu. Việc chuyển đổi sang ảnh xám hoặc ảnh nhị phân giúp giảm chiều dữ liệu, tăng tốc độ xử lý cho CPU/GPU mà vẫn giữ lại được các đặc trưng cấu trúc quan trọng [3].

Chuẩn hóa đầu vào cho các mô hình học máy: Các hệ thống AI hiện đại yêu cầu dữ liệu đầu vào phải đồng nhất. Việc tiền xử lý như thay đổi kích thước, chuẩn hóa màu sắc là bước bắt buộc để đảm bảo độ chính xác của thuật toán.

Tương thích hệ thống: Các thiết bị thu nhận ảnh và các thiết bị hiển thị đôi khi sử dụng các không gian màu khác nhau (ví dụ: RGB, BGR, YUV). Kỹ thuật chuyển đổi không gian màu là cầu nối thiết yếu để hệ thống vận hành trơn tru.

1.3. Lịch sử phát triển

Khởi đầu (1966): Lịch sử ghi nhận dự án "The Summer Vision Project" tại MIT năm 1966 là cột mốc khởi đầu. Giáo sư Seymour Papert đã giao cho sinh viên một bài tập hè với mục tiêu: kết nối camera vào máy tính và yêu cầu nó mô tả những gì nó nhìn thấy. Mặc dù mục tiêu này không thể hoàn thành trong một mùa hè, nó đã đặt nền móng cho ngành Thị giác máy hiện đại [4].

Sự ra đời của OpenCV và chuẩn BGR (1999): Dự án OpenCV (Open Source Computer Vision Library) được Intel khởi xướng vào năm 1999 nhằm mục đích cung cấp một cơ sở hạ tầng chung cho các ứng dụng thị giác máy.

Một điểm đặc thù cần lưu ý trong bài thực hành này là việc OpenCV sử dụng chuẩn màu **BGR (Blue-Green-Red)** mặc định thay vì RGB. Theo Satya Mallick (CEO của OpenCV.org), lý do bắt nguồn từ lịch sử phần cứng: Vào thời điểm OpenCV ra đời, các phần mềm giao tiếp với camera và thư viện đồ họa trên Windows (như Video for Windows) sử dụng cấu trúc bộ nhớ BGR. Để đảm bảo tính tương thích và hiệu năng tối đa với phần cứng thời đó, Intel đã chọn BGR và duy trì nó đến tận ngày nay như một tiêu chuẩn di sản [5].

1.4. Yêu cầu bài toán

Dựa trên nền tảng lý thuyết trên, bài tập này em tập trung thực hiện các kỹ thuật tiền xử lý ảnh cơ bản sử dụng ngôn ngữ Python và thư viện OpenCV. Các tác vụ bao gồm: đọc/ghi ảnh, thao tác trên các kênh màu, và các phép biến đổi không gian màu từ ảnh màu sang ảnh xám và ảnh nhị phân.

1.5. Công nghệ sử dụng

Mục này nói về các công nghệ sử dụng để giải quyết bài toán.

Ngôn ngữ: Python.

Thư viện: OpenCV, NumPy [8] [9].

Theo bảng xếp hạng IEEE Spectrum Top Programming Languages 2026, Python đang là vị trí số 1, và cách xa so với những ngôn ngữ khác, mặc dù bản chất Python cần biên dịch thông qua C rồi mới qua mã máy nhưng nó có cộng đồng hỗ trợ lớn, ứng dụng rộng rãi [6] [7] .

CHƯƠNG 2. THIẾT KẾ

2.1. Cấu trúc dự án

Cấu trúc dự án đi theo chuẩn tài liệu PEP 8 (Python Enhancement Proposal 8)[10].

basic_image_processing/

```
├── assets/                # Thư mục chứa tài nguyên
│   ├── sample.jpg         # Ảnh đầu vào (Input)
│   └── output_swapped.jpg  # Ảnh kết quả sau khi xử lý (Output)
├── src/                   # Thư mục mã nguồn chính
│   └── image_processor.py  # Module chứa Class xử lý ảnh
├── main.py                # File chạy chương trình
├── requirements.txt       # Danh sách các thư viện phụ thuộc
└── README.md              # Tài liệu hướng dẫn sử dụng
```

2.2. Thiết kế lớp

Tên lớp: ImageProcessor

Thuộc tính:

self.file_path: Đường dẫn đến file ảnh đầu vào.

self.image: Ma trận chứa dữ liệu điểm ảnh gốc (được load vào bộ nhớ)

self.processed_images: Lưu trữ tạm thời các ảnh đã qua xử lý.

Các phương thức:

Tên phương thức	Tham số đầu vào	Mô tả chức năng
__init__	file_path	Khởi tạo đối tượng, check đường dẫn và load ảnh.
show_image	window_name, image	Tạo cửa sổ hiển thị ảnh
show_separate_channels	Không	Tách ảnh thành R,G,B và print.
swap_channels	order	Swap vị trí trong matrix
to_grayscale	Không	Chuyển đổi sảnh màu sang xám
to_binary	threshold_val	Chuyển đổi ảnh xám sang ảnh nhị phân dựa trên threshold_val
save_image	image, path	Ghi ma trận ảnh từ bộ nhớ xuống ổ cứng dưới dạng file .jpg

CHƯƠNG 3. KẾT QUẢ BÀI TOÁN

Liên kết tới project: [Nhấp vào đây]

3.1. Môi trường chạy

Hệ điều hành: Windows 11 (64-bit).

Ngôn ngữ lập trình: Python 3.14.2.

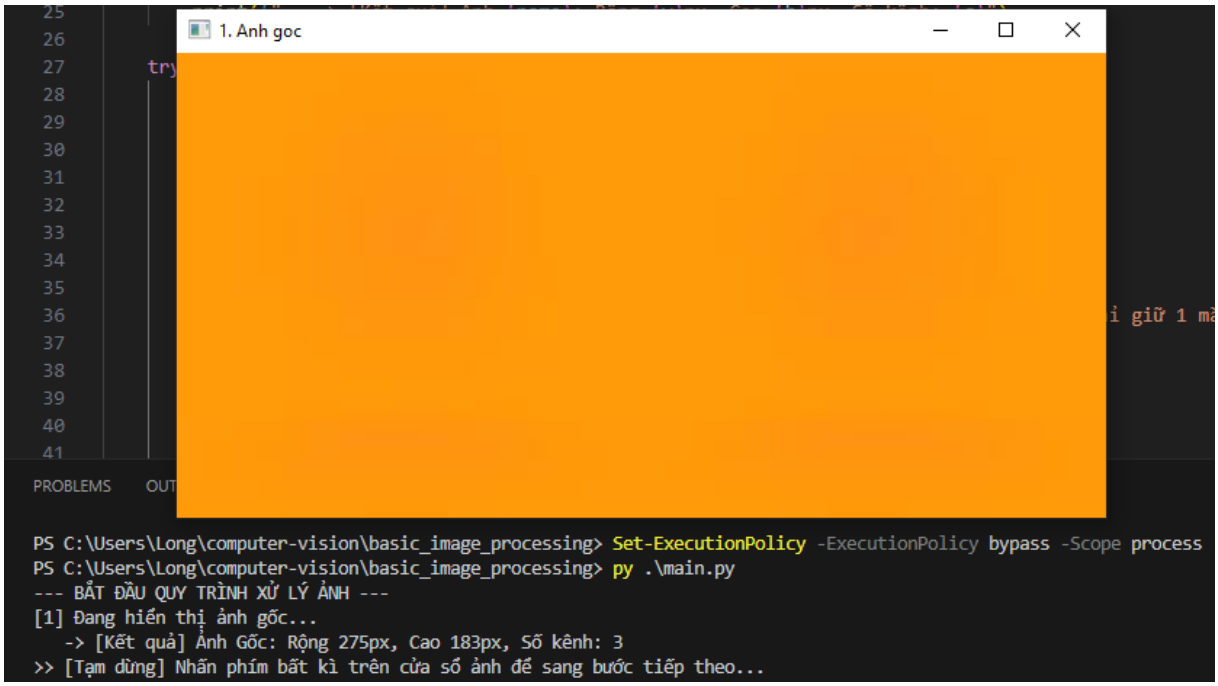
Môi trường phát triển: VSCode.

Thư viện hỗ trợ: OpenCV 4.8.0, NumPy: 1.24.0.

3.2. Kết quả đọc và hiển thị thông tin ảnh

Thực hiện: Chương trình đọc file ảnh đầu vào sample.jpg, nạp vào bộ nhớ dưới dạng đối tượng numpy.ndarray và trích xuất thông tin metadata.

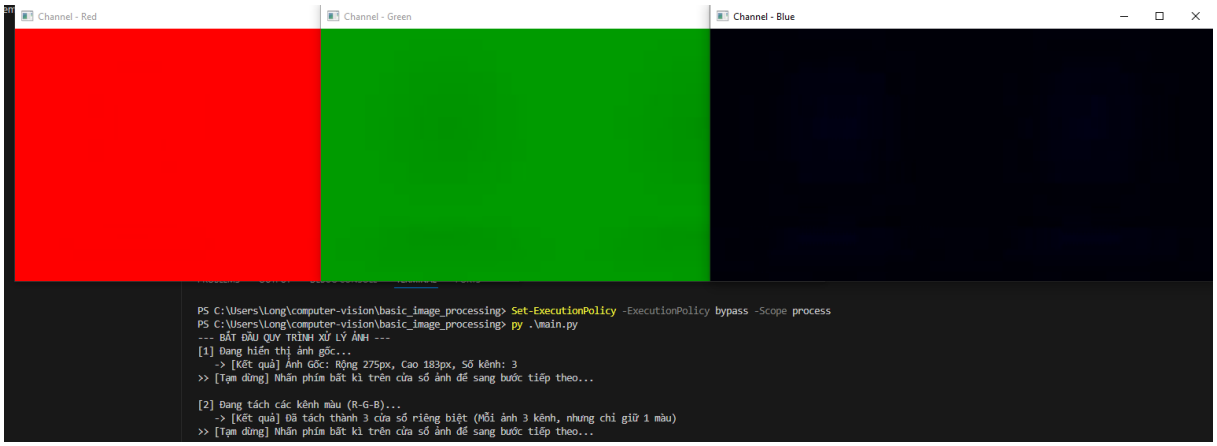
Kết quả:



Ảnh 1. Giao diện dòng lệnh hiển thị kích thước và số kênh màu.

3.3. Kết quả tách các kênh màu

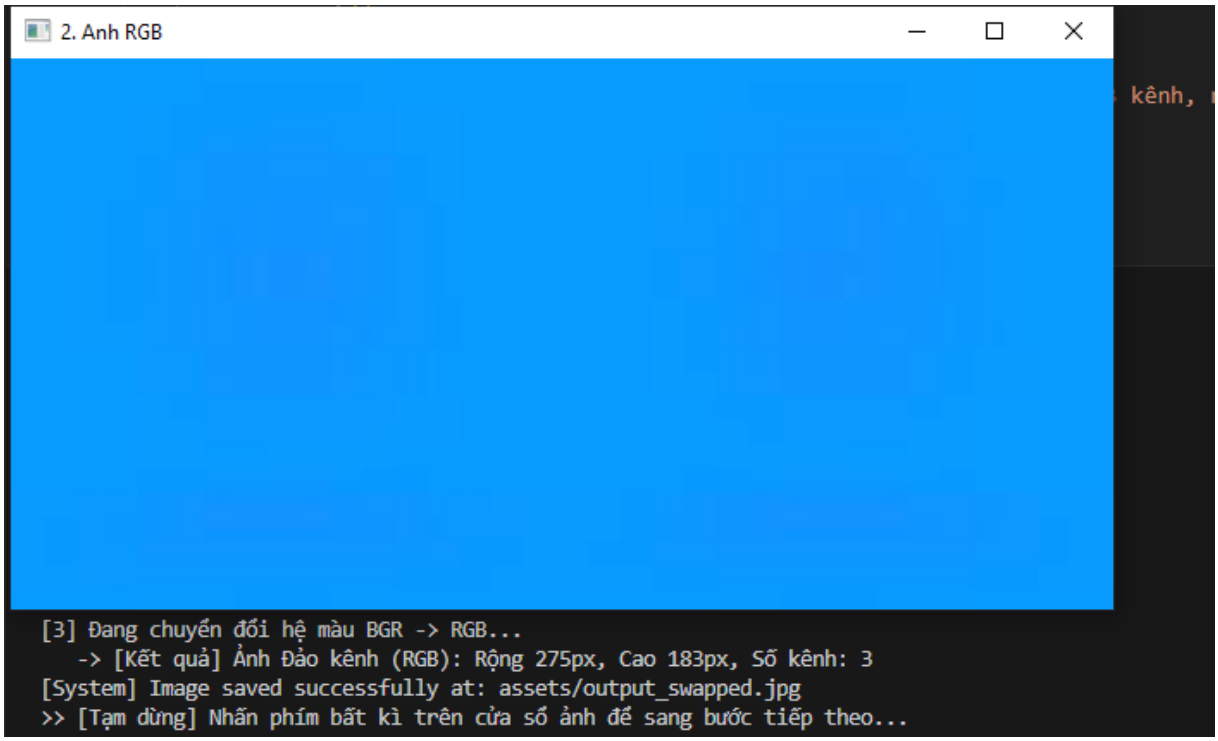
Thực hiện: Tách ma trận ảnh gốc thành 3 ma trận con tương ứng với 3 kênh Blue, Green, Red. Để trực quan hóa, các kênh này được hiển thị màu bằng cách gán 0 cho 2 kênh còn lại.



Ảnh 2. Kênh Blue, Kênh Green, Kênh Red

3.4. Kết quả chuyển đổi/đảo kênh màu

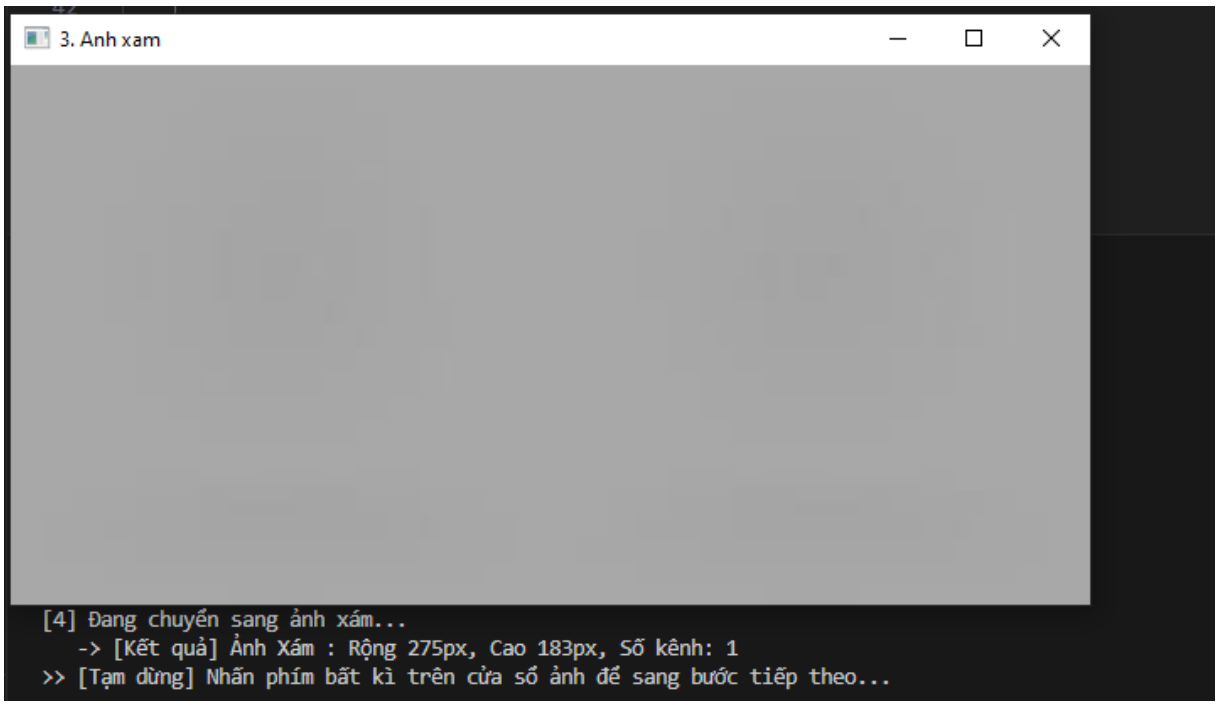
Thực hiện: Hoán đổi thứ tự các lớp ma trận từ chuẩn mặc định BGR (Blue-Green-Red) của OpenCV sang chuẩn RGB (Red-Green-Blue).



Ảnh 3. RGB

3.5. Kết quả chuyển đổi sang ảnh xám

Thực hiện: Áp dụng công thức Luma coding để loại bỏ thông tin màu sắc, chỉ giữ lại độ sáng.



Ảnh 4. Xám

3.6. Kết quả nhị phân hóa

Thực hiện: Áp dụng ngưỡng T = 127 lên ảnh xám. Tất cả điểm ảnh có giá trị >127 được gán bằng 255 (Trắng), ngược lại gán bằng 0 (Đen).

Ảnh 5. Nhị Phân

TÀI LIỆU THAM KHẢO

[1] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 4th ed. New York, NY, USA: Pearson, 2018.

[2] R. Szeliski, Computer Vision: Algorithms and Applications, 2nd ed. Springer, 2022.

[3] G. Bradski and A. Kaehler, Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library. O'Reilly Media, 2016.

[4] S. Papert, "The Summer Vision Project," MIT Artificial Intelligence Group, Memo No. 100, July 1966.

[5] C. R. Harris et al., "Array programming with NumPy," Nature, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

[6] S. Cass, "Top Programming Languages 2023," IEEE Spectrum, Aug. 29, 2025. [Online]. Available: https://spectrum.ieee.org/top-programming-languages-2025.

[7] S. Mallick, "Why does OpenCV use BGR color format?," LearnOpenCV.com. [Online]. Available: https://learnopencv.com/why-does-opencv-use-bgr-color-format/.

[8] F. Chollet, Deep Learning with Python, 2nd ed. Manning Publications, 2021.

[9] T. E. Oliphant, "Python for Scientific Computing," Computing in Science & Engineering, vol. 9, no. 3, pp. 10–20, 2007.

[10] G. van Rossum, B. Warsaw, and N. Coghlan, "PEP 8 – Style Guide for Python Code," Python Enhancement Proposals, Jul. 05, 2001. [Online]. Available: https://peps.python.org/pep-0008/.