Nhóm: ……..

- **……MSSV:…….**
- **……MSSV:……..**

----------o0o----------

BÀI 5B: XỬ LÝ ẢNH TRONG MATLAB

I. Giới Thiệu Tổng Quan Xử Lý Ảnh

Thị Giác Máy Tính (Computer Vision) và Xử Lý Ảnh (Image Processing) đã và đang rất phát triển từ thập niên những năm 70. Những phương pháp xử lý ảnh cho phép máy tính có thể tiếp nhận, phân tích, xử lý, hiểu được nội dung bức ảnh và, một cách tổng quát, tiếp thu được nhiều chiều dữ liệu trong thế giới thực. Từ đó, mở ra một cánh cổng cho phép con người giao tiếp với máy tính theo một cách thức mới, dễ dàng và trực quan hơn.

Xử lý ảnh hiện đại là một ngành nghiên cứu khoa học rất rộng lớn, bởi vì tính chất rộng lớn của nó nên cũng rất khó để phân loại cho chính xác từng lĩnh vực. Về cơ bản có thể chia làm 4 lĩnh vực lớn bao gồm:

1. Tăng cường ảnh (Image Enhancement): có thể hiểu một cách căn bản là biến ảnh xấu thành ảnh đẹp. Gồm nhiều kỹ thuật như: lọc nhiễu, nội suy, tăng cường màu, cân bằng trắng, image thinning, tự chỉnh tiêu cự, làm nét ảnh, v.v… và rất rất nhiều kỹ thuật khác.
2. Nhận diện (Detection/Recognition): nhận diện nội dung trong ảnh. Có thể là nhận diện khuôn mặt, nhận diện vật thể, tracking vật thể, đếm số lượng vật, phân loại các kiểu (phân loại gạo tốt/xấu, phân loại gạch lành/bể, phân loại nhiễm sắc thể bệnh/lành, v.v….), v.v… và rất rất nhiều thứ nhận diện/phân loại khác.
3. Nén ảnh (Compressing): được chia làm hai lĩnh vực chính là nén ảnh tĩnh và nén video. Việc nén ảnh tĩnh đa phần không quan tâm đến tốc độ nén mà trade-off giữa chất lượng ảnh và tỉ lệ nén. Việc nén video thì khó khăn hơn khi phải quan tâm đến tốc độ giải nén để kịp đẩy video ra ngoài theo thời gian thực. Nén video là sự cân bằng giữa tam giác tốc độ - chất lượng - tỉ lệ nén.
4. Đồ họa (Graphic): là một lĩnh vực khá độc lập so với 3 lĩnh vực trên. Trong khi ba lĩnh vực trên thì đầu vào là một bức ảnh sẵn có và ta xử lý chúng. Còn với đồ họa là ta tạo ra một bức ảnh hoặc video bằng kỹ thuật số.

Ngoài những lĩnh vực lớn như trên ra, thì xử lý ảnh còn bao gồm rất nhiều lĩnh vực nhỏ khác. Ngoài ra, cũng phải kể đến sự kết hợp giữa xử lý ảnh và nhiều ngành khoa học khác ngoài xử lý ảnh nữa. Trong đó có thể kể đến lĩnh vực nghiên cứu trí thông minh nhân tạo (Artificial Intelligence) có sự gắn bó khá mật thiết với lĩnh vực xử lý ảnh. Bằng việc áp dụng xây dựng mạng neuron nhân tạo để giải quyết một số bài toán trong xử lý ảnh, cũng như áp dụng các thuật toán máy học (Machine Learning) trong AI vào xử lý ảnh đã tạo nên một lĩnh vực nghiên cứu xử lý ảnh đương đại. Thậm chí, gần đây có người còn kết hợp xử lý ảnh với thuyết tiến hóa của Darwin mở ra một lĩnh vực nghiên cứu hoàn toàn mới.

II. Image Filter

Dịch tiếng việt thì đó là lọc ảnh. Tất cả về filter là nhân chập hình với một ma trận vuông. Lý thuyết nhân chập ma trận có thể diễn giải bằng đoạn code giả sau:

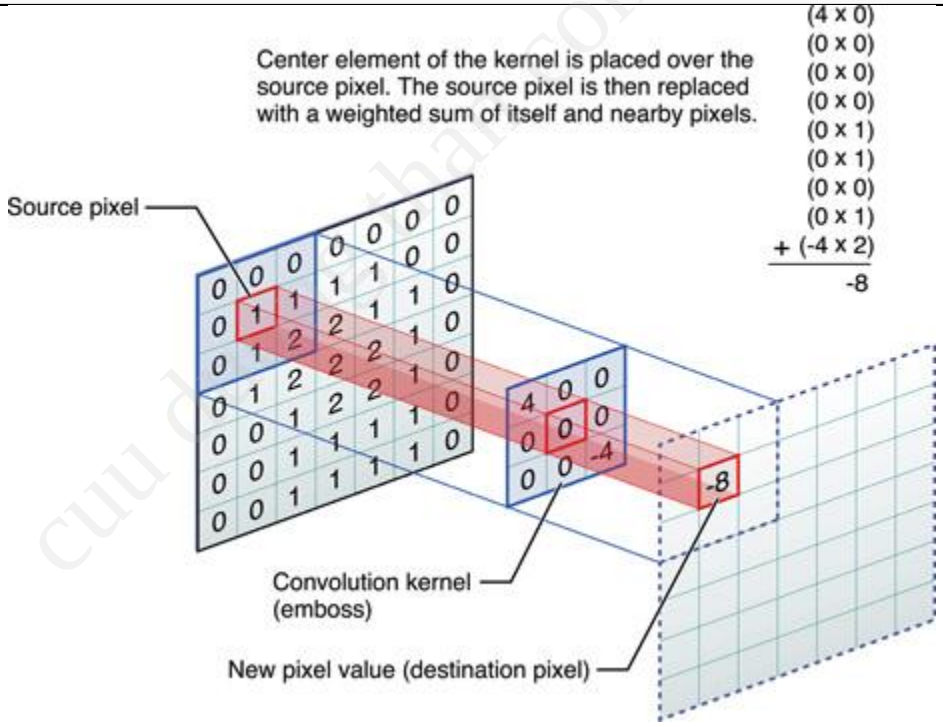
```
for each image row in output image:
   for each pixel in image row:

      set running total to zero

      for each kernel row in kernel:
         for each element in kernel row:

            multiply element value by corresponding* pixel value
            add result to running total

      set output image pixel to value of running total
```



1. **Chồng** ma trận vuông **lên** tấm hình và **lướt** nó đi **từng pixel một theo hàng**, **từng hàng theo cột** cho đến hết tấm hình.
2. **Tại mỗi thời điểm** ma trận vuông chồng lên tấm hình thì nó sẽ khoanh một vùng pixel tương ứng. Lấy **từng cặp chồng nhau** giữa pixel – giá trị ma trận **nhân lại** với nhau.
3. **Sum** lại hết **các cặp giá trị** để ra **một giá trị** cuối cùng. Sau đó **ghi** giá trị cuối cùng đó tại **tọa độ trung tâm** của **ảnh ngõ ra**.

Như vậy, về lý thuyết một tấm hình có độ phân giải MxN nhân chập với một ma trận vuông 3x3 thì ảnh ngõ ra sẽ có độ phân giải là (M-2)x(N-2). Nếu hình MxN nhân một ma trận 5x5 thì ảnh ngõ ra sẽ là (M-4)x(N-4), v.v…. Như vậy ảnh ngõ ra sẽ bị mất ngoài rìa, nhân chập như vậy ta gọi là **"valid"**. Còn nếu muốn ảnh ngõ ra có độ phân giải bằng ảnh ngõ vào thì ta nhân chập **"same"**. Ngoài ra, nhân chập **"full"** sẽ làm tăng độ phân giải ảnh ngõ ra, ví dụ MxN nhân chập full với 3x3 sẽ ra (M+4)x(N+4), MxN nhân chập full với 5x5 sẽ ra (M+8)x(N+8), v.v….

Lệnh nhân chập trong matlab sẽ là: conv2(A, B, <option>). Trong đó A là hình ảnh ngõ vào, B là ma trận vuông (còn gọi là ma trận trọng số hay ma trận nhân chập). Và "option" là các tùy chỉnh 'same', 'valid', hay 'full'. Ví dụ: conv2(A, B, 'same'); conv2(A, B, 'full'); conv2(A, B, 'valid').

III. Lọc nhiễu

Giả sử ta có một tấm hình có nhiễu như thế này.

Và ta có thể lọc nhiễu nó cho ra như thế này:

Phương pháp lọc nhiễu thì rất rất đa dạng đã được nghiên cứu và phát triển từ xưa đến nay, từ một bài thực hành của sinh viên đại học đến một luận văn tiến sĩ. Trong phạm trù bài thực hành này mình sẽ thực hiện hai phương pháp cổ điển nhất:

1. Lọc trung bình (Mean Filter).
2. Lọc trung vị (Median Filter).

Ý tưởng chung của lọc nhiễu là xem những cái nhiễu này là những đột biến trong hình, và đột biết này mang tính chất cục bộ (local) chứ không phải toàn cục (global). Ví dụ một pixel bị đột biến mang giá trị 0 thì ít ra những pixel ở lân cận ngay quanh nó sẽ rất khó bị đột biến. Hay nói cách khác, xác xuất để cho hai đột biến nhiễu nằm liền kề nhau là rất thấp. Do đó ta có thể xem (một cách lý tưởng) như không có chuyện hai nhiễu nằm liền kề nhau.

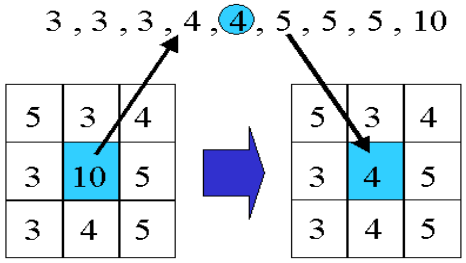
Vì vậy, ta sẽ "tìm và diệt" nhiễu bằng cách dùng một khung cửa sổ lướt đi trên hình và xét từng vùng local trong cửa sổ đó để tìm và triệt nhiễu. Lấy ví dụ ta dùng một khung cửa sổ 3x3 lướt trên tấm hình, tại mỗi thời điểm cửa sổ sẽ bốc ra một vùng 3x3 = 9 pixels lân cận nhau. Trong số 9 pixels liền kề nhau sẽ mang giá trị màu gần gần nhau, nếu trong vùng 3x3 pixels đó có nhiễu thì ta sẽ dễ nhận ra vì nó là đột biến khác hẳn tính chất của 8 pixels còn lại.

A. Lọc Trung Bình (Mean Filter)

Ta dùng một ô cửa sổ 3x3 lướt qua tấm hình, tại mỗi thời điểm ta có một vùng 3x3 pixels, ta sẽ cộng hết 9 giá trị đó lại và chia cho 9 (lấy giá trị trung bình của vùng 3x3), sau đó ghi đè lên ô trung tâm của cửa sổ. Như vậy, lọc trung bình chẳng khác nào ta nhân chập tấm hình với ma trận vuông 3x3 như sau [1/9 1/9 1/9; 1/9 1/9 1/9; 1/9 1/9 1/9].

B. Lọc Trung Vị (Median Filter)

Tương tự như lọc trung bình, ta cũng dùng một cửa sổ 3x3 lướt qua tấm hình. Tại mỗi thời điểm ta có một vùng 3x3 pixels, ta sẽ sắp xếp 9 giá trị này theo thứ tự tăng dần. Sau đó chọn ra giá trị nằm ở trung tâm của dãy 9 giá trị sau khi sắp xếp, đó chính là giá trị median. Sau đó ta sẽ thay thế giá trị median này lên giá trị trung tâm của cửa sổ. Ví dụ trong hình:

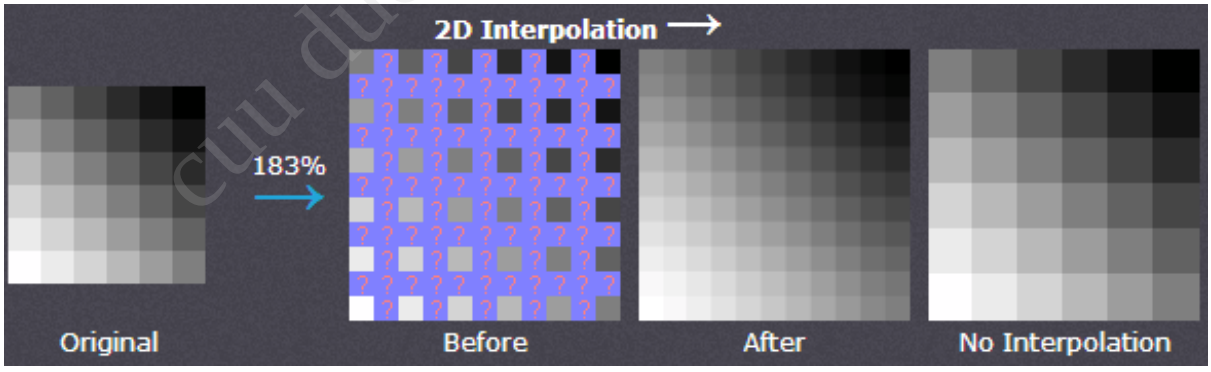


IV. Nội Suy Ảnh (Interpolation)

Nội suy ảnh là một phương pháp biến ảnh độ phân giải thấp thành độ phân giải cao làm sao cho càng đẹp càng tốt. Giống như lọc nhiễu, các phương pháp nội suy ảnh cũng cực kỳ đa dạng và trải rộng từ những bài thực hành sinh viên đến tiến sĩ vẫn còn làm về nội suy ảnh. Trong phạm trù bài thực hành này sẽ giới thiệu 3 phương pháp cổ điển nhất gồm:

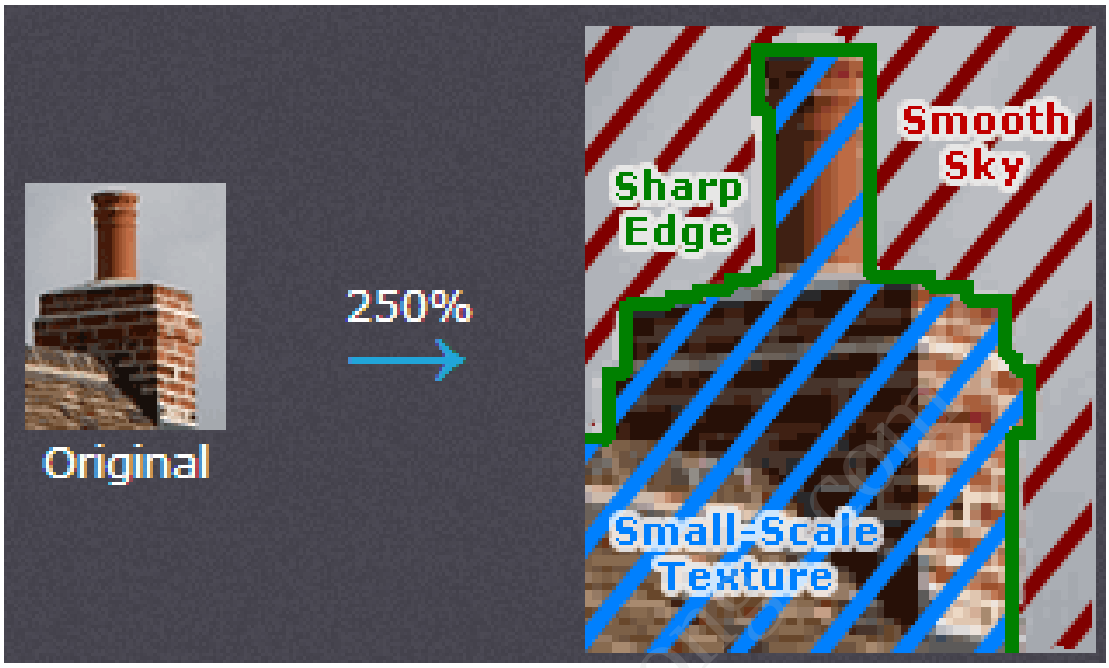
1. Bilinear.
2. Bicubic
3. Nearest Neighbor.

Ý tưởng chung của nội suy ảnh có thể hiểu như sau:



Từ một tấm ảnh gốc ban đầu có độ phân giải thấp, ta "kéo" nó ra thành độ phân giải cao bằng cách lấy những giá trị **pixel ban đầu** trong ảnh gốc **làm chuẩn**. Những pixel **"bị thiếu"** sẽ được bù đắp bằng cách dựa trên **thông tin "chuẩn"** nằm xung quanh nó.

Lấy lý tưởng rằng tấm hình gốc là "sạch", nghĩa là không chịu một nhiễu nào hết. Thì việc nội suy ảnh sẽ gặp phải 3 vấn đề sau:



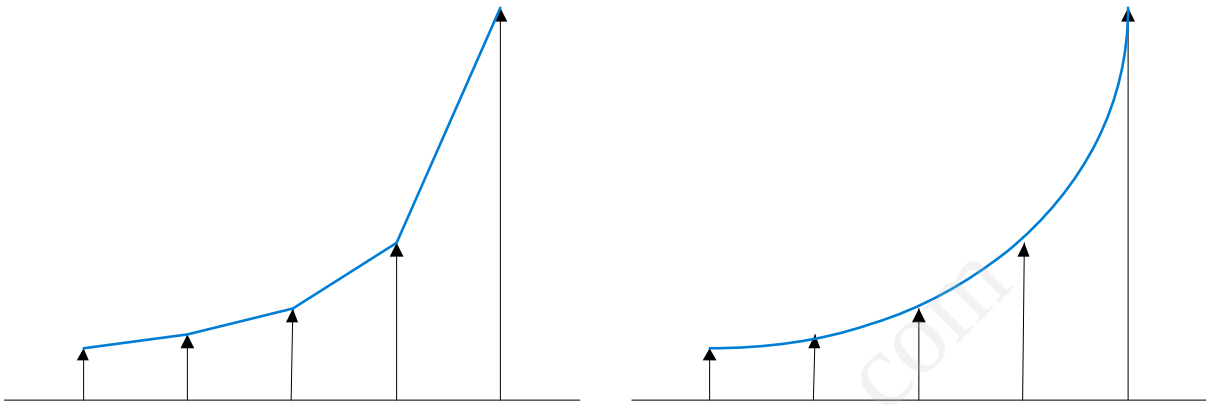
1. "Smooth sky": những vùng đồng nhất thì nội suy đồng nhất, giá trị nội suy sẽ mang tính chất trung bình của những pixel lân cận.
2. "Sharp edge": những vùng "chuyển" phải bật rõ được cạnh chuyển. Điều này là mâu thuẫn trực tiếp với "Smooth sky". Hình dung rằng ta có một thuật toán đáp ứng "Smooth sky" rất tốt thì sẽ bị mờ các cạnh (hay edge). Còn một thuật toán thỏa được "Sharp edge" thì những vùng đồng nhất sẽ thành "khó mà đồng nhất". Thỏa được cả hai yếu tố này là rất khó, mà thường là ta phải chịu nhân nhượng, trade-off giữa hai thứ (được này mất kia).
3. "Small scale texture": vấn đề này thậm chí còn khó hơn việc đạt được một "thỏa thuận" trade-off giữa "Smooth sky" và "sharp edge". Đây chính là những chi tiết quá nhỏ mà trong hình gốc (ảnh độ phân giải nhỏ) vốn đã nhìn không rõ. Việc phóng lớn hình ra sẽ càng phóng đại sự "không rõ" vốn có. Việc biến những chi tiết nhỏ không rõ nét trong hình nhỏ thành rõ nét trong hình lớn là một bài toán không thể giải trong xử lý ảnh cổ điển. Tuy nhiên, trong xử lý ảnh đương đại có rất nhiều phương pháp để giải bài toán này nhưng không được nêu ra trong phạm trù bài thực hành này.

A. Nearest Neighbor

Pixel bị thiếu sẽ được suy ra bằng cách lấy giá trị trung bình của những pixel "chuẩn" xung quanh lân cận gần nhất. Hay nói cách khác, ta sum các giá trị pixel "chuẩn" trong một khoảng cách 1 pixel (hoặc hơn nếu muốn) xung quanh điểm nội suy, sau đó chia lấy trung bình và lấy đó làm giá trị cho điểm nội suy.

B. Bilinear

Về cơ bản bilinear quan tâm đến xu hướng tăng/giảm các giá trị pixel theo hướng x và theo hướng y. Nói cách khác đó chính là gia tốc x và gia tốc y, nghĩa là đạo hàm của giá trị pixel theo hướng x và hướng y. Để dễ diễn giải, theo dõi hình sau.



Nội suy theo kiểu giá trị trung bình của nearest neighbor giống như ta nối thẳng các điểm lại vậy. Còn nội suy theo kiểu đạo hàm thì ta sẽ vẽ được đường cong mượt mà hơn (nhờ xét trên gia tốc chứ không phải giá trị).

C. Bicubic

Phương pháp bicubic hoàn toàn cùng ý tưởng với bilinear. Điểm khác nhau nằm ở chỗ.

- Bilinear: nội suy dựa trên đạo hàm hai trục (f'_x, f'_y).
- Bicubic: nội suy dựa trên đạo hàm bốn trục $(f'_{xx}, f'_{xy}, f'_{yx}, f'_{yy})$. Nghĩa là xét thêm hai trục đường chéo nữa.

Bicubic giống một phiên bản nâng cấp từ bilinear khi xét đến 4 trục đối xứng, cho kết quả nội suy được tròn vẹn hơn.

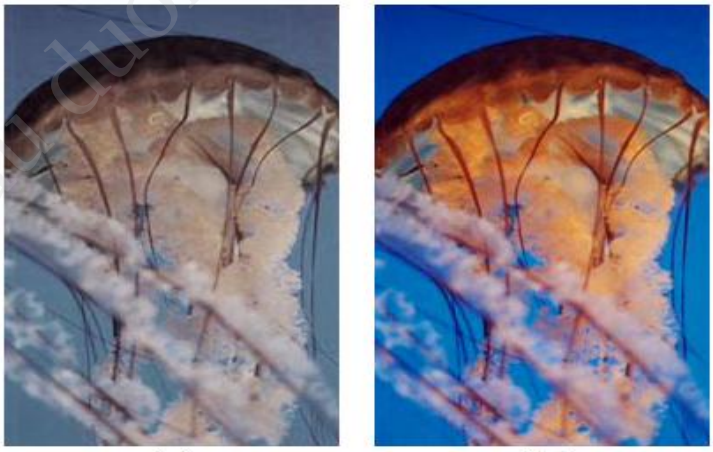
V. Tăng cường màu (Saturation)

Về cơ bản mọi loài sinh vật đều có một sự cảm thụ màu sắc hoàn toàn khác nhau. Ngay cả hai sinh vật trong cùng một loài cũng đã có sự sai khác. Điều khá thú vị khi ta biết rằng màu sắc mà ta cảm nhận được là hoàn toàn khác với màu sắc mà người ngồi ngay kế bên mình cảm nhận. Chẳng hạn cả hai người đều cùng nhìn một vật màu xanh, dĩ nhiên là cả hai đều sẽ đồng ý với nhau rằng đấy đúng là màu xanh. Nhưng sự đồng tình này là vốn dĩ đạt được là do định nghĩa "màu xanh" ở đây mang tính tương đối. Còn nếu đem so hai màu xanh

do hai người khác nhau cảm nhận và phát lên hai màn hình để cạnh nhau, ta sẽ nhận ra sự sai khác màu ngay lập tức. Như vậy, một câu hỏi đặt ra là, thế rốt màu đúng của vật thể là gì ?! Liệu nó có phải là màu xanh hay không, hay qua mắt của một sinh vật khác không phải con người thì lại thấy nó là màu đỏ ?! Có một sự thật ta phải chấp nhận rằng màu sắc thật sự của vật thể không phải là màu sắc mà ta cảm nhận được và ngược lại. Về cơ bản, con người hay bất kỳ một sinh vật nào khác đều không thể cảm nhận được đúng màu thật sự của thế giới xung quanh.

Như vậy, màu sắc mà camera ghi nhận được thông qua CCD (một dạng cảm biến ADC ánh sáng đặc thù) sẽ lại khác xa với sự cảm nhận chung của loài người. Nếu như ta phát thẳng tấm hình mà camera nhận được lên màn hình, chúng ta sẽ thấy màu sắc đó bị tái tái, tối tối so với cái màu mà ta thấy ngoài thực tế. Đấy không phải là lỗi do camera hay lỗi do hệ thống. Vấn đều màu "không đúng" mang tính chất sâu xa hơn. Cụ thể như sau:

- Camera ghi nhận tuyến tính theo cường độ ánh sáng (một đường thẳng). Cường độ ánh sáng của Red, Green, và Blue được đối xử như nhau không phân biệt.
- Mắt người ghi nhận phi tuyến theo cường độ ánh sáng. Đó là một đường cong lồi lên trên so với đường thẳng tuyến tính của camera, đấy là lý do sao ta nhìn hình gốc từ camera có vẻ tái tái, tối tối. Sự phi tuyến của mắt người lại không đồng đều giữa Red, Green, và Blue. Cụ thể mắt người nhạy Green gần gấp đôi so với hai màu còn lại.



(a) (b)

Hình bên (a) là hình gốc ghi nhận từ camera. Hình bên (b) là mắt người cảm nhận được từ thế giới bên ngoài. Như vậy, ta muốn có màu đẹp thì sẽ có một phương pháp tăng cường màu từ camera lên cho gần giống với mắt người cảm nhận. Phương pháp đó gọi là saturation.

Tại mỗi pixel ta có 3 giá trị Red, Green, và Blue, viết tắt là R, G, và B. Giờ từ (R, G, B) gốc ta sẽ chuyển thành (R', G', B') mới theo công thức sau.

$$\begin{cases} R' = (0.299 + 0.701 * K) * R + \big(0.587 * (1 - K)\big) * G + \big(0.114 * (1 - K)\big) * B \\ G' = \big(0.299 * (1 - K)\big) * R + (0.587 + 0.413 * K) * G + \big(0.114 * (1 - K)\big) * B \\ B' = \big(0.299 * (1 - K)\big) * R + \big(0.587 * (1 - K)\big) * G + (0.114 + 0.886 * K) * B \end{cases}$$

Trong đó, K được gọi là hệ số saturation. Nếu $K = 1$ thì đồng nghĩa màu không đổi so với màu gốc. $K = 0$ thì màu gần như thành trắng đen. Theo lẽ tự nhiên, K nằm trong khoảng 1.5 đến 1.8 là cho màu đẹp nhất. Sau đây là một ví dụ ảnh với nhiều hệ số K khác nhau.

Hình ở góc trên cùng bên trái là màu gốc từ camera (đồng nghĩa với K = 1). Cột thứ nhất (cột bên trái) là giảm K (nghĩa là K nhỏ hơn 1). Và hình ở dưới cùng bên trái tương ứng với K = 0, lúc này màu gần như là trắng đen. Cột thứ hai (cột bên phải) là tăng K (K lớn hơn 1). Hình dưới cùng bên phải tương ứng với K = 2, lúc này màu đã bị đẩy lên quá trớn.

Lệnh	Miêu Tả
A = imread(FILENAME)	Đọc ảnh từ file vào mảng A
imwrite(A, FILENAME)	Ghi ảnh từ mảng A ra file
imshow(A)	Hiển thị ảnh A
imhist(GRAY)	Hiển thị histogram cho ảnh GRAY
GRAY = rgb2gray(RGB)	Chuyển ảnh RGB sang grayscale
B = imresize(A, SCALE)	B là ảnh có kích thước bằng SCALE * kích thước gốc của A
B = imrotate(A, ANGLE)	B là ảnh xoay ANGLE độ so với A
B = conv2(A, M)	B là ảnh kết quả nhân chập giữa ảnh gốc A và ma trận nhân chập M
B = medfilt2(A, [m n])	B là kết quả lọc nhiễu median từ ảnh gốc A với cửa sổ mxn
I = uint8(X)	Chuyển đổi X thành số nguyên 8 bit không dấu