Tài Liệu Hướng Dẫn Lập Trình Xử Lý Ảnh

Chương II: Xử Lý Ảnh

Xử lý ảnh (DIP) có ứng dụng rất rộng và gần như tất cả các lĩnh vực kỹ thuật đều bị ảnh hưởng bởi DIP, sau đây là một số các ứng dụng chính của xử lý ảnh: khôi phục hình ảnh, chỉnh sửa, điều chỉnh độ phân giải, trong lĩnh vực y tế (tầm soát và phát hiện các triệu chứng bệnh), trong do thám, thám hiểm, thị giác máy tính, robot, lĩnh vực nhận dạng. Trong phần tiếp theo này, chúng ta sẽ cùng tìm hiểu các kỹ thuật cơ bản để xử lý ảnh.
Một ảnh số được lưu trữ trên máy tính là một ma trận các điểm ảnh (hay pixel). Trong OpenCV nó được biểu diễn dưới dạng cv::Mat. Ta xét một kiểu ảnh thông thường nhất, đó là ảnh RGB. Với ảnh này, mỗi pixel ảnh quan sát được là sự kết hợp của các thành phần màu R (Red), Green (Green) và Blue (Blue). Sự kết hợp này theo những tỉ lệ R, G, B khác nhau sẽ tạo ra vô số các màu sắc khác nhau. Giả sử ảnh được mã hóa bằng 8 bit với từng kênh màu, khi đó mỗi giá trị của R, G, B sẽ nằm trong khoảng [0, 255]. Như vậy ta có thể biểu diễn tới 255*255*255 ~ 1.6 triệu màu sắc từ ba màu cơ bản trên. Ta có thể xem cách biểu diễn ảnh trong OpenCV ở định dạng cv::Mat qua hình ảnh sau:

	Cột 0			**Cột 1**			…			**Cột m**		
Hàng 0	0, 0	0, 0	0, 0	0, 1	0, 1	0, 1	…	…	…	0, m	0, m	0, m
Hàng 1	1, 0	1, 0	1, 0	1, 1	1, 1	1, 1	…	…	…	1, m	1, m	1, m
…	…	…	…	…	…	…	…	…	…	…	…	…
Hàng n	n, 0	n, 0	n, 0	n, 1	n, 1	n, 1	…	…	…	n, m	n, m	n, m

Như vậy, mỗi ảnh sẽ có *n* hàng và *m* cột, *m* gọi là chiều dài của ảnh (width) và *n* gọi là chiều cao của ảnh (heigh). Mỗi pixel ở vị trí *(i,j)* trong ảnh sẽ tương ứng với 3 kênh màu kết hợp trong nó. Để truy xuất tới từng pixel ảnh với những kênh màu riêng rẽ ta sử dụng mẫu sau:

```
img.at<cv::Vec3b>(i,j)[k]
```

Trong đó, *i ,j* là pixel ở hàng thứ *i* và cột thứ *j, img* là ảnh mà ta cần truy xuất tới các pixel của nó. *cv::Vec3b* là kiểu vector uchar 3 thành phần, dung để biểu thị 3 kênh màu tương ứng. *k* là kênh màu thứ k, *k = 0, 1, 2* tương ứng với kênh màu B, G, R. Chú ý là trong OpenCV, hệ màu RGB được biểu diễn theo thứ tự chữ cái là BGR.

1. Điều chỉnh độ sáng và độ tương phản trong ảnh

Sau đây ta sẽ áp dụng kiến thức trên để làm tăng, giảm độ sáng và tương phản của một ảnh màu, việc làm này cũng hoàn toàn tương tự đối với ảnh xám, chỉ khác biệt là ảnh xám dùng một kênh duy nhất để biểu diễn.
Chương trình tăng, giảm độ sáng và độ tương phản của một ảnh.
Giả sử *f* là một hàm biểu diễn cho một ảnh nào đó, *f(x,y)* là giá trị của pixel trong ảnh ở vị trí *(x,y).*
Đặt $g(x,y) = \alpha f(x,y) + \beta$.
Khi đó, nếu $\alpha \neq 1$, thì ta nói ảnh g(x,y) có độ tương phản gấp α lần so với ảnh f(x,y).
Nếu $\beta \neq 0$ ta nói độ sáng của ảnh g(x,y) đã thay đổi một lượng là β.

Bài tập: Dựa vào công thức trên, viết chương trình thay đổi độ sáng và tương phản của 1 ảnh mẫu.

2. Ảnh nhị phân, nhị phân hoá với ngưỡng động

Ảnh nhị phân là ảnh mà giá trị của các điểm ảnh chỉ được biểu diễn bằng hai giá trị 0 hoặc 255 (1) tương ứng với hai màu đen hoặc trắng. Nhị phân hóa một ảnh là quá trình biến một ảnh xám thành ảnh nhị phân. Gọi *f(x,y)* là giá trị cường độ sáng của một điểm ảnh ở vị trí *(x,y)*, *T* là ngưỡng nhị phân. Khi đó, ảnh xám *f* sẽ được chuyển thành ảnh nhị phân dựa vào công thức:
f(x,y) = 0 nếu $f(x,y) \leq T$ và *f(x,y) = 255(1)* nếu $f(x,y) > T$
Trong OpenCV ta dung hàm threshold() để nhị phân hoá ảnh. Nguyên mẫu hàm như sau:

threshold(cv::InputArray *src*, cv::OutputArray *dst*, double *thresh*, int *maxval*, int *type*)

Trong đó:
src là ảnh đầu vào một kênh màu (ảnh xám …)
dst là ảnh sau khi được nhị phân hóa
thresh là ngưỡng giá trị để nhị phân
maxval là giá trị lớn nhất trong ảnh (maxval = 255 đối với ảnh xám)

type là kiểu nhị phân có thể là một trong các kiểu sau: CV_THRESH_BINARY (nhị phân hóa thông thường), CV_THRESH_BINARY_INV (nhị phân hóa ngược), CV_THRESH_OTSU (nhị phân hóa theo thuật toán Otsu)
Kết quả của việc nhị phân hóa một ảnh phụ thuộc vào ngưỡng T, có nghĩa là với mỗi ngưỡng T khác nhau thì ta có những ảnh nhị phân khác nhau.

Bài tập: Viết chương trình chuyển đổi một ảnh mẫu sang ảnh nhị phân với các ngưỡng lần lượt như sau: T = 50, T = 100 và T = 150.

Để thu được một ảnh nhị phân tốt mà không cần phải quan tâm tới các điều kiện ánh sáng khác nhau (không cần quan tâm tới ngưỡng T), người ta dùng một kĩ thuật sao cho với mọi ngưỡng xám khác nhau ta luôn thu được một ảnh nhi phân tốt. Kĩ thuật đó gọi là kĩ thuật nhị phân hóa với ngưỡng động (Dymamic threshold) hay nhị phân thích nghi (Adaptive threshold)
Có nhiều phương pháp khác khác nhau để thực hiện việc này, tuy nhiên chúng đều dựa trên ý tưởng chính là chia ảnh ra thành những vùng nhỏ, với mỗi vùng áp dụng việc nhị phân cho vùng đó với những ngưỡng nhị phân khác nhau.Các ngưỡng nhị phân ở các vùng được tính toán dựa trên độ lớn mức xám của chính các pixel trên vùng đó. Giả sử ta tính toán ngưỡng cho một vùng nào đó dựa trên độ trung bình của các pixel trong vùng đó (ta có thể xem một vùng là một cửa sổ). Ta xét quá trình nhị phân với ngưỡng động trong một vùng cửa sổ 5x5:

55	10	100	20	90
80	30	100	65	40
100	40	60	80	120
25	30	120	88	155
35	67	15	45	70

=>

0	0	255	0	255
255	0	255	0	0
255	0	0	255	255
0	0	255	255	255
0	255	0	0	255

Vùng ảnh nhị phân thu được ở trên là vùng ảnh được nhị phân với ngưỡng là trung bình cộng của tất cả các ô trong cửa sổ *T = (55 + 10 + 100 + ...)/25 = 65.6*
Trong OpenCV, ta dùng hàm adaptiveThreshold để nhị phân hoá với ngưỡng động. Nguyên mẫu của hàm như sau:
cv::adaptiveThreshold(cv::InputArray src, OutputArray dst, double maxValue,
int adaptiveMethod, int thresholdType, int blockSize, double C)
Trong đó:
src là ảnh xám cần nhị phân
dst là ảnh kết quả thu được,
maxValue là giá trị lớn nhất trong ảnh xám (thông thường là 255)
adaptiveMethod là cách thức nhị phân với ngưỡng động
thresholdType như các *type* trong hàm threshold
blockSize là kích thước của sổ áp dụng cho việc tính toán ngưỡng động
C là một thông số để bù trừ trong trường hợp ảnh có độ tương phản quá lớn

Bài tập: Viết chương trình nhị phân hoá ảnh mẫu với ngưỡng động.

3. Histogram, cân bằng histogram trong ảnh

Histogram của một ảnh là một biểu đồ nói lên mối quan hệ giữa các giá trị của pixel ảnh (điểm ảnh) và tần suất xuất hiện của chúng. Nhìn vào biểu đồ histogram ta có thể đoán được một ảnh sáng tối như thế nào.
Nếu một ảnh có histogram lệch về phía phải biểu đồ, ta nói ảnh đó thừa sáng. Nếu lệch về phía trái thì ảnh đó thiếu sáng. Đối với ảnh màu, ta có thể tính toán histogram cho từng kênh màu một.
Trong OpenCV hàm calcHist dùng để tính toán histogram của ảnh. Nguyên mẫu của hàm như sau:
*cv:: calcHist(const cv::Mat *images, int nimages, const int *channels,*
cv::InputArray mask, cv::OutputArray hist, int dims, const int histSize,
*const float **ranges, bool uniform = true, bool accumulate = false)*
trong đó:
images là ảnh đầu vào
hist lưu kết quả tính toán histogram
nimages là số lượng ảnh đầu vào
channels là danh sách chiều các kênh dùng để tính histogram

mask là một mặt nạ tùy chỉnh, nếu không dung gì thì để là *cv::Mat()*
dims là chiều của histogram
histSize là kích thước dãy histogram mỗi chiều, hai tham số cuối có thể để mặc định
Cân bằng histogram (histogram equalization) là phương pháp làm cho biểu đồ histogram của ảnh được phân bố một cách đồng đều. Đây là một biến đổi khá quan trọng giúp nâng cao chất lượng ảnh, thông thường đây là bước tiền xử lý của một ảnh đầu vào cho các bước tiếp theo.
Để cân bằng histogram, ta dùng hàm: *equalizeHist(cv::InputArray src, cv::OuputArray dst)*
trong đó: *src* là ảnh đầu vào một kênh màu (ảnh xám chăn hạn), *dst* là ảnh sau khi cân bằng.

Bài tập: Viết chương trình tính toán histogram của một ảnh và cân bằng histogram của ảnh đó.

4. Phóng to, thu nhỏ và xoay ảnh

Ảnh số thực chất là một ma trận các điểm ảnh, do đó để có thể phóng to, thu nhỏ hay xoay một tấm ảnh ta có thể sử dụng các thuật toán tương ứng trên ma trận.
Ta sẽ sử dụng biến đổi affine để quay và thay đổi tỉ lệ to, nhỏ của một ma trận.

Biến đổi affine:

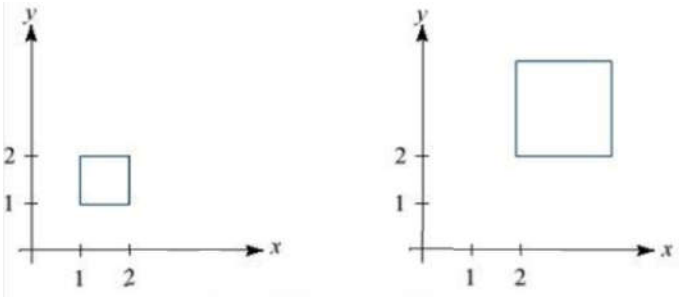
Giả sử ta có vector $p = [x, y]^T$ và ma trận M là ma trận 2x2. Phép biến đổi affine trong không gian hai chiều có thể được định nghĩa $p' = Mp$ trong đó $p' = [x', y']^T$. Viết một cách tường minh ta có:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha & \delta \\ \gamma & \beta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

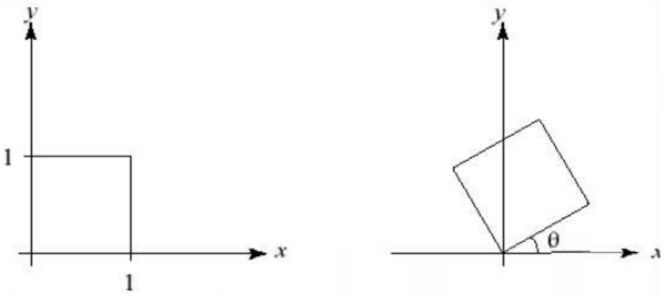
Hay $x' = \alpha x + \delta y,\ y' = \gamma x + \beta y$

Xét ma trận $\begin{bmatrix} \alpha & \delta \\ \gamma & \beta \end{bmatrix}$.

Nếu $\delta = \gamma = 0$, khi đó $x' = \alpha x$ và $y' = \beta y$, phép biến đổi này làm thay đổi tỉ lệ của ma trận. Nếu là trong ảnh nó sẽ phóng to hoặc thu nhỏ ảnh. Hình sau mô tả phép biến đổi với tỉ lệ $\alpha = \beta = 2$



Nếu ta định nghĩa ma trận M = $\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$ thì phép biến sẽ quay p thành p' với góc quay là θ



Nếu ma trận M được định nghĩa thành M = $\begin{bmatrix} \alpha.\cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \beta.\cos(\theta) \end{bmatrix}$ thì phép biến đổi sẽ vừa là phép biến đổi tỉ lệ vừa là phép quay góc θ.

Trong OpenCV, hàm *cv::getRotationMatrix2D*(*cv::Point center*, *double angle*, *double scale*) sẽ tạo ra ma trận với tâm quay tại điểm *center*, góc quay *angle* và tỉ lệ *scale.* Ma trận này được tính trong OpenCV là ma trận sau: M = $\begin{bmatrix} \alpha & \delta & (1-\alpha).center.x - \beta.center.y \\ \gamma & \beta & \beta.center.x - (1-\alpha).center.y \end{bmatrix}$

Với $\alpha = scale.\cos(angle)$ và $\beta = scale.\sin(angle)$

Ta thấy rằng ma trận này là hoàn toàn tương đương với ma trận của phép biến đổi affine đã nói ở trên, ngoại trừ thành phần thứ 3 là thành phần giúp dịch chuyển tâm quay vào chính giữa của bức ảnh. Chú ý là có sự khác biệt một chút về chiều của hệ toạ độ trong ảnh, hệ toạ độ trong ảnh lấy góc trên bên trái làm gốc toạ độ (0, 0) còn hệ toạ độ thông thường ta hay lấy điểm dưới bên trái làm gốc, do đó có sự ngược chiều.

Bài tập: Viết chương trình để quay ảnh một góc 45 độ và phóng to với tỉ lệ 1.5

Ngoài hai phép biến đổi là tỉ lệ và quay như trên, ta có thể thực hiện các biến đổi khác của phép biến đổi affine như phép trượt (shearing), hoặc phép phản chiếu (reflection) bằng việc định nghĩa lại ma trận *M*. Ta thử định nghĩa lại ma trận M để được một ảnh trượt của ảnh gốc. Quay lại ma trận M như trên, nếu ta định nghĩa α=β=1 còn δ nhận một giá trị bất kì, khi đó ta sẽ có:

$\begin{cases} x' = x + \delta y \\ y' = y \end{cases}$ ta sẽ định nghĩa ma trận M = $\begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ để trượt ảnh ban đầu thành ảnh mới với hệ số trượt δ = 0.5, chú ý là thành phần thứ ba $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ định nghĩa ma trận trong Opencv sẽ thể hiện độ dịch chuyển, giống như trong ví dụ trên ta chuyển tâm quay về tâm của bức ảnh chẳng hạn.

Bài tập: Thay ma trận *M* là ma trận ta tự định nghĩa mat_rot bên dưới, so sánh ảnh đầu ra so với ảnh gốc.

```
double I[2][3] = {1,0.5,0, 0,1,0}; // cac phan tu cua ma tran
Mat mat_rot(2,3,CV_64F, I); // khoi tao ma tran
warpAffine(src, dst, mat_rot, src.size());
```

5. Lọc số trong ảnh

Lọc số trong ảnh có ý nghĩa quan trọng trong việc tạo ra các hiệu ứng trong ảnh, một số hiệu ứng nhờ sử dụng các bộ lọc như làm mờ ảnh (Blur), làm trơn ảnh (Smooth), … Nguyên tắc chung của phương pháp lọc ảnh là cho ảnh nhân chập với một ma trận lọc $I_{dest} = M * I_{src}$

Trong đó: Isrc, Idst là ảnh gốc và ảnh sau khi thực hiện phép lọc ảnh bằng cách nhân với ma trận lọc *M*. Ma trận *M* đôi khi còn được gọi là mặt nạ (mask), nhân (kernel). Với mỗi phép lọc ta có những ma trận lọc *M* khác nhau, không có quy định cụ thể nào cho việc xác định *M,* tuy nhiên ma trận này có một số đặc điểm như sau:

- Kích thước của ma trận thường là một số lẻ như: 3x3, 5x5, 7x7, 9x9… Khi đó, tâm của ma trận sẽ nằm ở giao của hai đường chéo và là điểm áp đặt lên ảnh mà ta cần tính nhân chập.
- Tổng các phần tử trong ma trận thông thường bằng 1. Nếu tổng này lớn hơn 1, ảnh qua phép lọc sẽ có độ sáng lớn hơn ảnh ban đầu. Ngược lại ảnh thu được sẽ tối hơn ảnh ban đầu.

Ví dụ về ma trận lọc (ma trận lọc Sobel theo hướng x, y và ma trận Gausian Blur)

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Công thức cụ thể cho việc lọc ảnh như sau:

$$I_{dest}(x,y) = I_{src}(x,y) * M(u,v) = \sum_{u=-n}^{n} \sum_{v=-n}^{n} I(x-u, y-v) \times M(u,v)$$

Trong đó, ta đang tính phép nhân chập cho điểm ảnh có tọa độ *(x,y)* và vì ta lấy tâm của ma trận lọc là điểm gốc nên *u* chạy từ *-n* (điểm bên trái) và *v* chạy từ *-n* (điểm phía trên) đến *n*, với *n = (kích thước mặt nạ - 1)/2*. Để dễ hiểu hơn ta xét một ví dụ về việc làm trơn nhờ sử dụng một ma trận lọc như sau:

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} * \begin{bmatrix} 100 & 100 & 100 & 100 & 100 \\ 100 & 200 & 205 & 203 & 100 \\ 100 & 195 & 200 & 200 & 100 \\ 100 & 200 & 205 & 195 & 100 \\ 100 & 100 & 100 & 100 & 100 \end{bmatrix} = \begin{bmatrix} 100 & 100 & 100 & 100 & 100 \\ 100 & 144 & 205 & 203 & 100 \\ 100 & 195 & 200 & 200 & 100 \\ 100 & 200 & 205 & 195 & 100 \\ 100 & 100 & 100 & 100 & 100 \end{bmatrix}$$

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} * \begin{bmatrix} 100 & 100 & 100 & 100 & 100 \\ 100 & 200 & 205 & 203 & 100 \\ 100 & 195 & 200 & 200 & 100 \\ 100 & 200 & 205 & 195 & 100 \\ 100 & 100 & 100 & 100 & 100 \end{bmatrix} = \begin{bmatrix} 100 & 100 & 100 & 100 & 100 \\ 100 & 144 & 167 & 203 & 100 \\ 100 & 195 & 200 & 200 & 100 \\ 100 & 200 & 205 & 195 & 100 \\ 100 & 100 & 100 & 100 & 100 \end{bmatrix}$$

….
Kết quả cuối cùng ta có:

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} * \begin{bmatrix} 100 & 100 & 100 & 100 & 100 \\ 100 & 200 & 205 & 203 & 100 \\ 100 & 195 & 200 & 200 & 100 \\ 100 & 200 & 205 & 195 & 100 \\ 100 & 100 & 100 & 100 & 100 \end{bmatrix} = \begin{bmatrix} 100 & 100 & 100 & 100 & 100 \\ 100 & 144 & 167 & 145 & 100 \\ 100 & 167 & 200 & 168 & 100 \\ 100 & 144 & 166 & 144 & 100 \\ 100 & 100 & 100 & 100 & 100 \end{bmatrix}$$

Ta thấy rằng ảnh ban đầu có sự chênh lệch lớn giữa các giá trị của các pixel (ảnh có độ tương phản lớn 100, 200). Sau khi lọc ảnh, độ tương phản của ảnh đã giảm đi đáng kể.
Sau đây, chúng ta cùng xem xét một số bộ lọc của OpenCV.

a. Lọc Blur:

Ma trận lọc của phương pháp này có dạng:
Bộ lọc này có tác dụng làm trơn ảnh, khử nhiễu hạt. Trong OpenCV, hàm lọc Blur có dạng như sau:

cv::blur(const Mat& src, const Mat& dst, Size ksize, Point anchor, int borderType)

trog đó: *src* và *dst* là ảnh gốc và ảnh sau phép lọc, *ksize* là kích thước ma trận lọc, *ksize = Size(rows, cols)*, anchor là điểm neo của ma trận lọc, nếu để mặc định là *(-1,-1)* thì điểm này chính là tâm của ma trận, *borderType* là phương pháp để ước lượng và căn chỉnh các điểm ảnh nếu qua phép lọc chúng bị vượt ra khỏi giới hạn của ảnh. Thông thường giá trị mặc định của nó là 4.

Bài tập: Dùng bộ lọc Blur để lọc ảnh sau và đánh giá kết quả!

(https://4.bp.blogspot.com/_5UXLw8fmChc/ShJDuS1CzWI/AAAAAAAAAYE/5xuwoAEDAlQ/s1600/lena_saltpepper.jpg)

b. Lọc Sobel:

Lọc sobel chính là cách tính xấp xỉ đạo hàm bậc nhất theo hướng x và y, nó cũng chính là cách tính gradient trong ảnh. Bộ lọc này thông thường được áp dụng cho mục đích tìm biên trong ảnh. Ma trận lọc theo các hướng x, y lần lượt như sau:

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Trong OpeCV, hàm lọc Sobel có dạng như sau:

cv::Sobel(const Mat& src, Mat& dst, int ddepth, int xorder, int yorder, int ksize, double scale, double delta, int borderType)

Trong đó: *src và dst* là ảnh gốc và ảnh qua phép lọc, *ddepth* là độ sâu của ảnh sau phép lọc, có thể là *CV_32F, CV_64F … . xoder* và *yoder* là các đạo hàm theo hướng x và y, để tính đạo hàm theo hướng nào ta đặt giá trị đó lên 1, ngược lại nếu giá trị bằng 0, hàm cài đặt sẽ bỏ qua không tính theo hướng đó, *Scale* và *delta* là hai thông số tùy chọn cho việc tính giá trị đạo hàm lưa giá trị vi sai vào ảnh sau phép lọc, chúng có giá trị mặc định là 1 và 0. *borderType* là tham số như trên.

Bài tập: Dùng bộ lọc Sobel để lọc ảnh đường giao thông và đánh giá kết quả!

c. Lọc Laplace:

Lọc Laplace là cách tính xấp xỉ đạo hàm bậc hai trong ảnh, nó có ý nghĩa quan trọng trong việc tìm biên ảnh và phân tích, ước lượng chuyển động của vật thể.

$$dst = \Delta src = \frac{\partial^2 src}{\partial x^2} + \frac{\partial^2 src}{\partial y^2}$$

Ma trận lọc có dạng như sau:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Trong OpenCV, bộ lọc Laplace có dạng như sau:

cv::Laplacian(const Mat& src, Mat& dst, int ddepth, int ksize=1, double scale=1, double delta=0, int borderType)

Các thông số này có ý nghĩa giống như các thông số trong bộ lọc Sobel, chỉ khác ở chỗ *ksize* là một giá trị *int* mặc định bằng 1 và khi đó ma trận lọc laplace trên được áp dụng.

Bài tập: Dùng bộ lọc Sobel để lọc ảnh đường giao thông và đánh giá kết quả!

Ngoài 3 bộ lọc trên, OpenCV còn cài đặt khá nhiều bộ lọc khác như lọc trung vị (*medianBlur*), lọc Gause (*gaussianBlur*), *pyrDown, pyrUp* … Tuy nhiên, ta hoàn toàn có thể cài đặt bộ lọc cho riêng mình thông qua hàm *cv::filter2D*. Nguyên mẫu hàm này như sau:

filter2D(const Mat& src, Mat& dst, int ddepth, const Mat& kernel, Point anchor, double delta, int borderType).

Trong đó, *src* và *dst* là ảnh gốc và ảnh thu được qua phép lọc, *kernel* là ma trân lọc. Thông số *anchor* để chỉ ra tâm của ma trận, *delta* điều chỉnh độ sáng của ảnh sau phép lọc (ảnh sau phép lọc được cộng với *delta* và *borderType* là kiểu xác định những pixel nằm ngoài vùng ảnh.

Hàm *cv::filter2D* thực chất là hàm tính toán nhân chập giữa ảnh gốc và ma trận lọc để cho ra ảnh cuối sau phép lọc. Như vậy qua trên ta thấy rằng để tiến hành việc lọc ảnh ta chỉ cần định nghĩa một ma trận lọc *kernel.* Có rất nhiều nghiên cứu về toán học đã định nghĩa các ma trận này, do đó việc áp dụng vào lọc ảnh sẽ là khá đơn giản.

Bài tập: Viết chương trình lọc ảnh số với ma trận lọc $M = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$

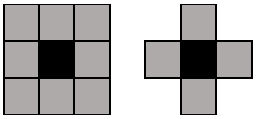
6. Các phép toán hình thái học trong ảnh

Các phép toán hình thái học là những phép toán liên quan tới cấu trúc hình học (hay topo) của các đối tượng trong ảnh. Các phép toán hình thái học tiêu biểu bao gồm phép giãn nở (dialation) phép co (erosion), phép mở (opening) và phép đóng (closing).

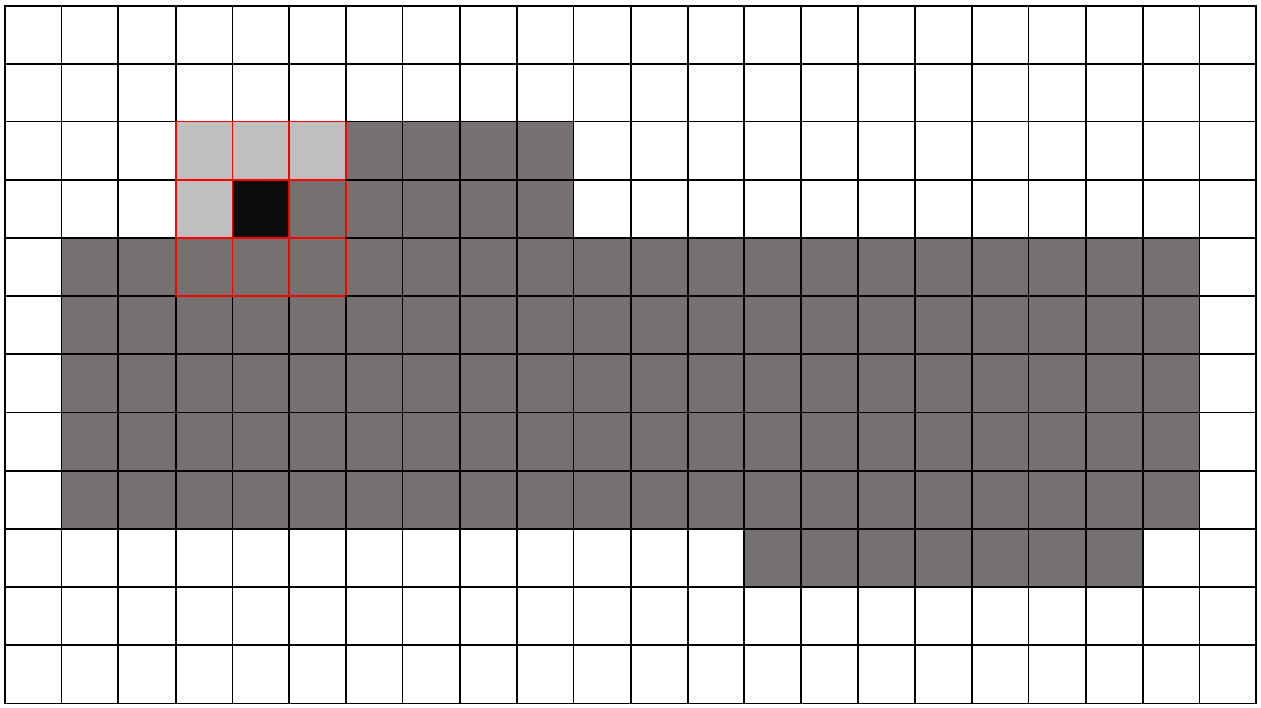
a. Phép toán giãn nở (dilation)

Phép toán giãn nở được định nghĩa $A \otimes B = \cup B_x$ với x ⊂ A trong đó, A là đối tượng trong ảnh, B là một cấu trúc phần tử ảnh. Phép toán này có tác dụng làm cho đối tượng ban đầu trong ảnh tăng lên về kích thước (giãn nở ra)

Cấu trúc phần tử ảnh (image structuring element) là một hình khối được định nghĩa sẵn nhằm tương tác với ảnh xem nó có thỏa mãn một số tính chất nào đó không, một số cấu trúc phần tử hay gặp là cấu trúc theo khối hình vuông và hình chữ thập.



Ta hãy xét một ảnh với đối tượng trong ảnh được biểu diễn bằng màu nền nâu, sau đó dùng cấu trúc phần tử hình vuông (màu đỏ) để làm giãn nở ảnh, kết quả là ảnh được giãn nở ra và phần giãn nở ra ta đánh dấu là dấu x.



Ứng dụng của phép giãn nở là làm cho đối tượng trong ảnh được tăng lên về kích thước, các lỗ nhỏ trong ảnh được lấp đầy, nối liền đường biên ảnh đối với những đoạn rời nhỏ.

b. Phép toán co (erosion)

Phép toán co trong ảnh được định nghĩa $A \ominus B = \{x|(B)_x \subseteq A\}$ trong đó A là đối tượng trong ảnh, B là cấu trúc phần tử ảnh. Ta cũng sẽ xét một ví dụ như trên với phép co trong ảnh. Đối tượng trong ảnh được biểu diễn bởi màu xám, cấu trúc phần tử ảnh là khối có viền màu đỏ, x là điểm sau phép thỏa mãn phép co ảnh, 0 là điểm không thỏa mãn.

Ta thấy rằng sau phép toán này đối tượng trong ảnh bị co lại, chính vì vậy mà nó được ứng dụng trong việc giảm kích thước của đối tượng, tách rời các đối tượng gần nhau và làm mảnh, tìm xương đối tượng.

Phép toán mở (opening) và đóng (closing) là sự kết hợ của phép co (erosion) và giãn (dialation) như trên, chúng được định nghĩa như sau:

Phép toán mở : $A \circ B = (A \ominus B) \oplus B$

Phép toán đóng: $A \cdot B = (A \oplus B) \ominus B$

Phép toán mở được ứng dụng trong việc loại bỏ các phần lồi lõm và làm cho đường bao đối tượng trong ảnh trở lên mượt mà hơn.

Phép toán đóng được ứng dụng trong việc làm trơn đường bao đối tượng, lấp đầy các khoảng trống trên biên và loại bỏ những hố nhỏ (một số pixel đứng thành cụm độc lập).

Trong OpenCV, các phép toán hình thái học trong ảnh được cài đặt trong hàm:

cv::morphologyEx, riêng phép giãn nở và phép co có thể gọi trực tiếp từ hàm *cv::dilate* và *cv::erode*

morphologyEx(*const Mat& src*, *Mat& dst*, *int op*, *const Mat& element*, *Point anchor*,

int iterations, *int borderType*, *const Scalar& borderValue*)

Trong đó, *src, dst* là ảnh đầu vào và ảnh sau phép xử lý hình thái học, *op* là kiểu lựa chọn phép hình thái học, chẳng hạn như phép giản nở là MORPH_DILATE, phép đóng là MORPH_OPEN …. *element* là cấu trúc phần tử ảnh, có ba cấu trúc cơ bản là theo khối hình vuông, hình chữ thập và hình elip. Để tạo ra các cấu trúc này ta có thể tự định nghĩa một ma trận với các hình khối tương ứng hoặc sử dụng hàm *getStructuringElement*, hàm này có cấu trúc như sau: *getStructuringElement(int shape, Size ksize, Point anchor),* với *shape* là kiểu hình khối (một trong 3 hình khối trên), *ksize* là kích thước của hình khối và là khích thước của hai số nguyên lẻ, *anchor* là điểm neo và thông thường nhận giá trị là *((ksize.width – 1)/2, (ksize.height – 1)/2).* Thông số tiếp theo *anchor* cũng có ý nghĩa tương tự, *iterations* là số lần lặp lại của phép toán hình thái và hai thông số cuối cùng là về giới hạn biên của những điểm ảnh nằm ngoài kích thước ảnh trong quá trình tính toán, nó có ý nghĩa như trong nội dung trước.

Phép toán về giãn nở và co có thể được gọi từ hàm *cv::morphologyEx* thông qua hai đối số *op* là *MORPH_DILATE* và *MORPH_ERODE* hoặc chúng có thể được gọi trực tiếp từ hàm *cv::dilate* và *cv::erode*, Cấu trúc của hai hàm này là tương tự nhau và các tham số hoàn toàn giống với tham số trong hàm *morphologyEx*:

dilate(const Mat& src, Mat& dst, const Mat& element, Point anchor=Point(-1, -1),
int iterations, int borderType, const Scalar& borderValue)

Một điều cần chú ý là trái với cách diễn đạt về các phép hình thái như trên, OpenCV có cách cài đặt ngược lại giữa đối tượng và nền ảnh, nghĩa là trong phép giản nở (dilate), phần được giản nở là nền của ảnh (background) chứ không phải các đối tượng vật thể trong ảnh, điều đó cũng có nghĩa rằng phép giản nở sẽ làm co lại các đối tượng vật thể trong ảnh và phép co (erode) sẽ làm co nền của ảnh đồng thời giản nở các đối tượng vật thể trong ảnh.

7. Tìm biên ảnh dựa trên bộ lọc Canny

Bộ lọc Canny là sự kết hợp của nhiều bước khác nhau để tìm và tối ưu đường biên, kết quả là cho ra một đường biên khá mảnh và chính xác. Quá trình tìm biên sử dụng phương pháp canny có thể được thực hiện qua 4 bước sau:

Bước 1: Loại bớt nhiễu trong ảnh.

Người ta loại nhiễu trong ảnh, làm cho ảnh mờ đi bằng cách nhân chập ảnh với một bộ lọc Gause, chẳng hạn bộ lọc Gaus 5x5 với hệ số σ = 1.4:

$$M = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Bước 2: Tính toán giá trị gradient trong ảnh

Vì đường biên trong ảnh là nơi phân cách giữa các đối tượng khác nhau, nên tại đó gradient của nó sẽ biến đổi mạnh mẽ nhất. Để tính toán gradient trong ảnh, ta có thể sử dụng bộ lọc Sobel, hoặc trực tiếp nhập chập ma trận ảnh với các mặt nạ theo hướng *x* và *y* chẳng hạn:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad ; \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Sau đó tính độ lớn gradient trong ảnh:

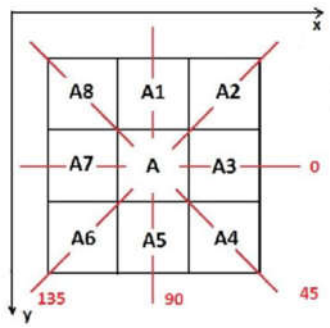
$$G = \sqrt{G_x^2 + G_y^2} \; v\text{à } \theta = \arctan(\frac{G_y}{G_x})$$

Trong đó, G_x, G_y chính là đạo hàm theo hướng X, Y của điểm ảnh ta đang xét. Góc θ sẽ được làm tròn theo các hướng thẳng đứng, nằm ngang và hướng chéo, nghĩa là nó sẽ được làm tròn để nhận các giá trị 0, 45, 90 và 135 độ.

Bước 3: Loại bỏ các giá trị không phải là cực đại

Bước này sẽ tìm ra những điểm ảnh có khả năng là biên ảnh nhất bằng cách loại bỏ đi những giá trị không phải là cực đại trong bước tìm gradient ảnh ở trên. Ta thấy rằng, với giá trị của góc θ ở trên thì biên của đối tượng có thể tuân theo bốn hướng, và ta có bốn khả năng sau:

4. Nếu $\theta = 0_0$, khi đó, điểm A sẽ được xem là một điểm trên biên nếu độ lớn gradient tại A lớn hơn độ lớn gradient của các điểm tại A_3, A_7.
5. Nếu $\theta = 45_0$, khi đó, điểm A sẽ được xem là một điểm trên biên nếu độ lớn gradient tại A lớn hơn độ lớn gradient của các điểm tại A_4, A_8
6. Nếu $\theta = 90_0$, khi đó, điểm A sẽ được xem là một điểm trên biên nếu độ lớn gradient tại A lớn hơn độ lớn gradient của các điểm tại A_1, A_5.
7. Nếu $\theta = 135_0$, khi đó, điểm A sẽ được xem là một điểm trên biên nếu độ lớn gradient tại A lớn hơn độ lớn gradient của các điểm tại A_2, A_6



Bước 4: Chọn ra biên của đối tượng trong ảnh
Sau bước trên, ta thu được tập các điểm tương ứng trên đường biên khá mỏng. Vì những điểm có giá trị gradient lớn bao giờ cũng có xác suất là biên thật sự hơn những điểm có giá trị gradient bé, đo đó để xác định chính xác hơn nữa biên của các đối tượng, ta sử dụng các ngưỡng. Theo đó, bộ lọc canny sẽ sử dụng một ngưỡng trên (upper threshold) và một ngưỡng dưới (lower threshold), nếu gradient tại một điểm trong ảnh có giá trị lớn hơn ngưỡng trên thì ta xác nhận đó là một điểm biên trong ảnh, nếu giá trị này bé hơn ngưỡng dưới thì đó không phải điểm biên. Trong trường hợp giá trị gradient nằm giữa ngưỡng trên và ngưỡng dưới thì nó chỉ được tính là điểm trên biên khi các điểm liên kết bên cạnh của nó có giá trị gradient lớn hơn ngưỡng trên.
Tìm biên ảnh bằng bộ lọc canny trong OpenCV:
OpenCV cung cấp một hàm cho ta xác định biên trong ảnh của các đối tượng. Nguyên mẫu của hàm này như sau:

canny(Mat src, Mat dst, int lower_threshold, int upper_threshold, int kernel_size)

Trong đó, *src* là ảnh cần phát hiện biên (là ảnh xám), *dst* là ảnh chứa biên được phát hiện, *lower_threshold, upper_threshold* là ngưỡng dưới, ngưỡng trên như đã nói bên trên, *kernel_size* là kích thước của mặt nạ dùng cho bước thứ hai để tính toán gradient của các điểm trong ảnh.
Trước khi tìm biên bằng phương pháp canny ta đã làm trơn một ảnh xám bằng bộ lọc *GaussianBlur* (việc làm này có ý rất lớn trong việc giảm thiểu các tiểu tiết không mong muốn trong đường biên sau này). Về nguyên tắc, bộ lọc GaussianBlur cũng là một bộ lọc số như đã giới thiệu ở bài trước về lọc số trong ảnh, cấu trúc của hàm GaussianBlur như sau:

cv::GaussianBlur(cv::InputArray src, cv::OutputArray dst, cv::Size ksize, double sigmaX, double sigmaY = 0, int borderType = 4)

trong đó src và dst sẽ là các mảng dữ liệu đầu vào và kết quả. Một ảnh cũng có thể coi là một mảng dữ liệu, đo đó *src* và *dst* chính là ảnh đầu và ảnh thu được sau khi biến đổi, *ksize* là kích thước của ma trận lọc, sigmaX, *sigmaY* là độ lệch chuẩn theo hướng x và y. Nếu *sigmaY = 0*, độ lệch chuẩn theo hướng y sẽ được lấy theo *sigmaX*. *BorderType* là cách nội suy biên đối với những điểm ảnh tràn ra ngoài kích thước của ảnh.

8. Chuyển đổi Hough, Phát hiện đường thẳng, đường tròn trong ảnh

Chuyển đổi Hough (Hough transformation) là một phương pháp được dùng nhiều trong phân tích và xử lý ảnh, mục đích chính của phương pháp này là tìm ra những hình dáng đặc trưng trong ảnh bằng cách chuyển đổi không gian ảnh ban đầu sang một không gian của các tham số nhằm đơn giản quá trình tính toán, trong bài này ta xét chuyển đổi Hough cho đường thẳng và đường tròn.

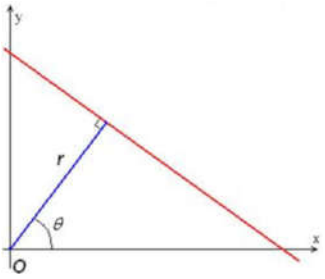
a. Chuyển đổi Hough cho đường thẳng:

Ta đã biết rằng, một đường thẳng trong không gian hai chiều có thể được biểu diễn dưới dạng y = kx + m và cặp hệ số góc k, giá trị m có thể được chọn để làm đặc trưng cho một đường thẳng. Tuy nhiên, cách biểu diễn theo cặp (k, m) khó thỏa mãn với những đường thẳng thẳng đứng khi mà hệ số góc là một số vô cùng. Để tránh trường hợp này, ta sẽ biểu diễn đường thẳng trong hệ tọa độ cực.
Phương trình đường thẳng trong hệ tọa độ cực có dạng như sau:

$$r = xcos(\theta) + ysin(\theta)$$

Trong đó, r là khoảng cách từ gốc tọa độ O tới đường thẳng, θ là góc cực.
Như vậy, với mỗi điểm (x_0, y_0) ta có một họ các đường thẳng đi qua thõa mãn phương trình

$$r_0 = x_0cos(\theta) + y_0sin(\theta)$$



Phương trình này biểu diễn một đường cong, như vậy trong một tấm ảnh có n điểm (n pixel) ta sẽ có n các đường cong. Nếu đường cong của các điểm khác nhau giao nhau, thì các điểm này cùng thuộc về một đường thẳng. Bằng cách tính các giao điểm này, ta sẽ xác định được đường thẳng, đó là nội dung ý tưởng của thuật toán Hough cho đường thẳng.

b. Chuyển đổi Hough cho đường tròn:

Chuyển đổi Hough cho đường tròn cũng tương tự như với đường thẳng, phương trình đường tròn được xác định bởi:

$$\begin{cases} x = u + Rcos\theta \\ y = v + Rsin\theta \end{cases}$$

Trong đó, (u,v) là tâm đường tròn, R là bán kính đường tròn, θ là góc có giá trị từ 0 tới 360 độ. Một đường tròn sẽ hoàn toàn được xác định nếu ta biết được bộ ba thông số (u,v,R). Từ phương trình trên ta có thể chuyển đổi tương đương:

$$\begin{cases} u = x - Rcos\theta \\ v = y - Rsin\theta \end{cases}$$

Ta xét với trường hợp đã biết trước giá trị của *R*. Khi đó, với mỗi điểm ảnh (*x,y*) ta sẽ xác định được một giá trị (*u,v*) và lưu nó vào một mảng. Tâm của đường tròn sẽ là giá trị xuất hiện trong mảng với tần suất lớn nhất. Trong trường hợp *R* chưa biết, ta tăng giá trị của *R* từ một ngưỡng min tới ngưỡng max nào đó và tiến hành như với trường hợp đã biết trước giá trị *R*

Tìm đường thẳng, đường tròn trong ảnh.

Để tìm đường thẳng trong ảnh, thư viện OpenCV có hỗ trợ hàm cv::HoughLines và cv::HoughLinesP, hai hàm này về cơ bản đều thực hiện một thuật toán, nhưng đưa ra kết quả ở hai dạng khác nhau.

Dạng 1: đưa ra kết quả của đường thẳng là một cặp giá trị *(r, θ)*

cv::HoughLines(src, lines, rho, theta, threshold, param)

Trong đó, *src* là ảnh nhị phân chứa biên của các đối tượng cần được phát hiện đường thẳng (trong thực tế ta có thể sử dụng là một ảnh xám), *lines* là vector chứa kết quả đầu ra của *(r, θ)* đã được phát hiện, rho là độ phân giải của r (tính theo pixel, thực chất đó là bước tăng nhỏ nhất của r để tính toán), *theta* là độ phân giải của góc *θ* (có giá trị từ 0 tới 360 độ), *threshold* là giá trị nhỏ nhất của tổng các giao điểm của các đường cong để xác định đường thẳng, *param* là một thông số mặc định của OpenCV chưa sử dụng đến (nó có giá trị bằng 0)

Dạng 2: đưa ra kết quả là tọa độ điểm đầu và điểm cuối của đường thẳng

cv::HoughLinesP(src, lines, rho, theta, threshold, minLenght, maxGap)

Trong đó, các thông số *src, rho, theta, threshold* giống như dạng 1, *lines* là vector chứa tọa độ điểm đầu và tọa độ điểm cuối của đường thẳng phát hiện được (x_1, y_1, x_2, y_2), *minLength* là độ dài nhỏ nhất để có thể xem nó là một đường thẳng (tính theo đơn vị pixel), *maxGap* là khoảng cách lớn nhất của hai điểm cạnh nhau để xác định chúng có thuộc về một đường thẳng hay không (tính theo đơn vị pixel) Đối với đường tròn, ta cũng áp dụng hàm tương tự: *cv::HougCircles()*, nguyên mẫu của hàm như sau:

cv::HougCircles(src, circles, method, dp, min_dist, param1, param2, min_radius, max_radius)

Trong đó, *src* có thể là một ảnh nhị phân chứa biên của đối tượng hoặc ảnh xám (chương trình sẽ tự động dò biên bằng phương pháp canny), *circles* là vector chứa kết quả phát hiện đường tròn với 3 tham số (*a, b, R*), method là phương pháp phát hiện đương tròn (hiện tại phương pháp trong OpenCV là *CV_HOUGH_GRADIENT*), *dp* là tỉ số nghịch đảo của độ phân giải (áp dụng trong phương pháp hiện tại), *min_dist* là khoảng cách nhỏ nhất giữa hai tâm đường tròn phát hiện được, *param1, param2* là các thông số ngưỡng trên và ngưỡng dưới phục vụ cho việc phát hiện biên bằng phương pháp canny, *min_radius và max_radius* là giới hạn nhỏ nhất, lơn nhất của bán kính đường tròn ta cần phát hiện(Nếu ta không biến bán kính, để mặc định hai giá trị này bằng không thì chương trình sẽ lần tăng giá trị bán kính một cách tự động để tìm ra tất cả các đường tròn trong ảnh).

Bài tập: Viết chương trình đọc vào 1 ảnh màu, sau đó được chuyển đổi qua ảnh xám *gray* và được làm trơn đi nhờ hàm *GaussianBlur*. Sử dụng hàm Canny để tìm ra các điểm biên, sử dụng *HoughLines* và *HoughCircles* để tìm ra các đường thẳng, đường tròn trong ảnh.