ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

*****

BÁO CÁO LẬP TRÌNH ỨNG DỤNG

Đề tài:

Xây dựng Chương trình Giải quyết bài toán Phát hiện tiền "Bẩn" trong Ảnh tiền (Fitness Classification) sử dụng C/C++ và OpenCV

Giảng viên hướng dẫn: (Thầy) Hoàng Văn Xiêm

Thành viên:

Lê Trung Kiên	210216	K66-ĐACLC2
Vũ Cao Thạch	21021633	K66-ĐACLC2

Lớp: 2223II_ELT2014_21

Phân công công việc:

Thành viên	Công việc
Lê Trung Kiên	Tìm hiểu và xây dựng code, tài liệu
Vũ Cao Thạch	Tìm hiểu và xây dựng code, tài liệu

Giới thiệu bài toán:

Về cơ bản, kỹ thuật phát hiện bất thường (Anomaly or Outlier Detection) dùng để nhấn mạnh vào việc quan sát các mục dữ liệu trong bộ dữ liệu để tìm ra các tập dữ liệu không khớp với mẫu dự kiến. Bất thường ở đây có thể đề cập đến độ lệch, sự khác thường, các nhiễu và ngoại lệ.

Sự bất thường được xem là khá quan trọng vì nó có thể cung cấp một số thông tin cần thiết. Nó có thể là một dữ liệu khác biệt so với mức trung bình chung trong một tập dữ liệu. Điều này chỉ ra rằng một cái gì đó khác thường đã xảy ra và các nhà phân tích dữ liệu cần chú ý

II. Một số nghiên cứu liên quan

1. Contours

a. Khái niệm:

Contour là "tập các điểm-liên-tục tạo thành một đường cong (curve) (boundary), và không có khoảng hở trong đường cong đó, đặc điểm chung trong một contour là các các điểm có cùng /gần xấu xỉ một giá trị màu, hoặc cùng mật độ. Contour là một công cụ hữu ích được dùng để phân tích hình dạng đối tượng, phát hiện đối tượng và nhận dạng đối tượng".

b. Tìm contours trong ảnh:

Trong OpenCV, contour hoạt động trên ảnh đơn kênh, nhưng hiệu quả nhất là ảnh nhị phân. Việc tìm kiếm contours trong OpenCV khá đơn giản bằng hàm findContours(). Hàm này đưa ra danh sách các contours tìm được, với mỗi contour là một danh sách các tọa độ điểm.

C++:

```
void findContours(InputOutputArray image, OutputArrayOfArrays contours, int mode, int method, Point offset = Point())
```

Trong đó:

image: ảnh đầu vào (đơn kênh hoặc ảnh nhị phân)

contours: danh sách contours tìm được

mode: cách truy vấn các contours, thông thường mình đặt là CV_RETR_TREE

method: phương thức ước lượng số đỉnh của từng contour. Có 2 phương thức thường dùng nhất là CV_CHAIN_APPROX_NONE (liệt kê toàn bộ các đỉnh của đa giác contour) và CV_CHAIN_APPROX_SIMPLE (hạn chế số đỉnh phải lưu trữ).

2. Threshold(ngưỡng):

a. Khái niệm:

Trong opencv, ngưỡng là một số nằm trong đoạn từ 0 đến 255. Giá trị ngưỡng sẽ chia tách

giá trị độ xám của ảnh thành 2 miền riêng biệt. Miền thứ nhất là tập hợp các điểm ảnh có giá trị nhỏ hơn giá trị ngưỡng. Miền thứ hai là tập hợp các các điểm ảnh có giá trị lớn hơn hoặc bằng giá trị ngưỡng.

Đầu vào của một thuật toán phân ngưỡng trong opencv thường có input là ảnh nguồn (source image) và giá trị ngưỡng. Đầu ra là ảnh đích đã được phân ngưỡng (destination image).

b. Tạo threshold:
Nếu pixel có giá trị lớn hơn giá trị ngưỡng thì nó được gán 1 giá trị (thường là 1), ngược lại nhỏ hơn giá trị ngưỡng thì nó được gán 1 giá trị khác (thường là 0).

```
double threshold(Mat src, Mat dst, double thresh, double maxval, int type)
```

Hàm sử dụng là threshold , tham số đầu tiên là 1 ảnh xám, tham số thứ 2 là giá trị ngưỡng, tham số thứ 3 maxval là giá trị được gán nếu giá pixel lớn hơn giá trị ngưỡng, tham số thứ 4 là loại phân ngưỡng. Tùy theo các loại phân ngưỡng mà pixel được gán giá trị khác nhau:

THRESH_BINARY

Nếu giá trị pixel lớn hơn ngưỡng thì gán bằng maxval

Ngược lại bằng gán bằng 0

THRESH_BINARY_INV

Nếu giá trị pixel lớn hơn ngưỡng thì gán bằng 0

Ngược lại bằng gán bằng maxval

THRESH_TRUNC

Nếu giá trị pixel lớn hơn ngưỡng thì gán giá trị bằng ngưỡng

Ngược lại giữ nguyên giá trị

THRESH_TOZERO

Nếu giá trị pixel lớn hơn ngưỡng thì giữ nguyên giá trị

Ngược lại gán bằng 0

THRESH_TOZERO_INV

Nếu giá trị pixel lớn hơn ngưỡng thì gán giá trị bằng 0

Ngược lại giữ nguyên.

3. Các phép toán hình thái học trong OpenCV

Trong OpenCV, các phép toán hình thái học trong ảnh được cài đặt trong hàm cv::morphologyEx().

```
cv::morphologyEx(InputArray src, OutputArray dst, int op, InputArray kernel,
    Point anchor = Point(-1, -1), int iterations = 1,
    int borderType = BORDER_CONSTANT,
    const Scalar & borderValue = morphologyDefaultBorderValue());
```

Phân tích

src: Là ảnh ban đầu:

dst: Là ảnh sau khi xử lý hình thái học.

op: Là loại hình thái học, ví dụ: MORPH_ERODE. Bạn có thể tham kháo các loại phép toán hình thái học trong cv::MorphTypes.

kernel: Là phần tử cấu trúc. Bạn hoàn toàn có thể khởi tạo và sử dụng bằng cách sử dụng hàm getStructuringElement().

anchor: Là điểm neo của phần tử cấu trúc kernel. Giá trị mặc định là (-1, -1).

iterations: Số lần lặp đi lặp lại các phép toán hình thái học lên ảnh. Ví dụ: Bạn càng để phép toán giãn nở ảnh (Dialation) thì ảnh càng giãn nở nhiều.

borderType-borderValue: Là giới hạn biên của những điểm ảnh (Pixel) nằm ngoài kích thước của ảnh trong quá trình tính toán.

+Hàm getStructuringElement():

```
cv::getStructuringElement(int shape, Size ksize, Point anchor = Point(-1, -1));
```

Phân tích

shape: Là hình dạng của phần tử cấu trúc. Ví dụ: MORPH_RECT. Bạn có thể tham khảo các loại hình dạng của phần tử cấu trúc trong cv::MorphShapes.

ksize: Là kích thước của ma tận phần tử cấu trúc.

anchor: Là điểm neo của phần tử cấu trúc. Giá trị mặc định là (-1, -1).

Ví Dụ:

```cpp
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

using namespace cv;

int main() {

    Mat imageSrc = imread("C:/Users/vctha/Desktop/jpeg/Original.jpg", IMREAD_GRAYSCALE);
    Mat imageDst;
    Mat imageBinary;

    threshold(imageSrc, imageBinary, 220, 255, THRESH_BINARY);
    cv::Mat kernel = cv::getStructuringElement(MORPH_CROSS, Size(5, 5));
    cv::morphologyEx(imageBinary, imageDst, MORPH_ERODE, kernel);

    imshow("imageBinary", imageBinary);
    imshow("imageDst", imageDst);

    waitKey(0);
    return 0;
}
```

III. Đề xuất thuật toán, mô hình, đánh giá kết quả

1.Bài toán tìm vết bẩn

*__Mô tả bài toán__ : Đây là bài toán tìm kiếm vết bẩn xuất hiện trên 1 tờ tiền bất kì.

*Ý tưởng thuật toán: So sánh từng mảng màu của tờ tiền gốc với tờ tiền bẩn, sử dụng threshold để chọn ngưỡng của 2 ảnh rồi trừ cho nhau, lưu kết quả vào 1 ảnh tmp.

*Cài đặt:

a. Khởi tạo dữ liệu

```
#include "Two_Same_Images.h"

void Same_Img_Processing() {
    Mat img = imread("1.jpg", IMREAD_GRAYSCALE);
    Mat cop = imread("4.jpg", IMREAD_GRAYSCALE);
    ll h = img.rows, w = img.cols;
    Mat tmp = cop.clone();
```

b.Tạo ngưỡng cho 2 ảnh:

```
threshold(img, img, 89, 255, THRESH_BINARY);
threshold(cop, cop, 50, 255, THRESH_BINARY);
```

c. Thực hiện phép trừ 2 ảnh và lưu vào tmp:

```
for (ll i = 0; i < h; i++) {
    for (ll j = 0; j < w; j++) {
        tmp.at<uchar>(i, j) = abs(cop.at<uchar>(i, j) - img.at<uchar>(i, j));
    }
}
```

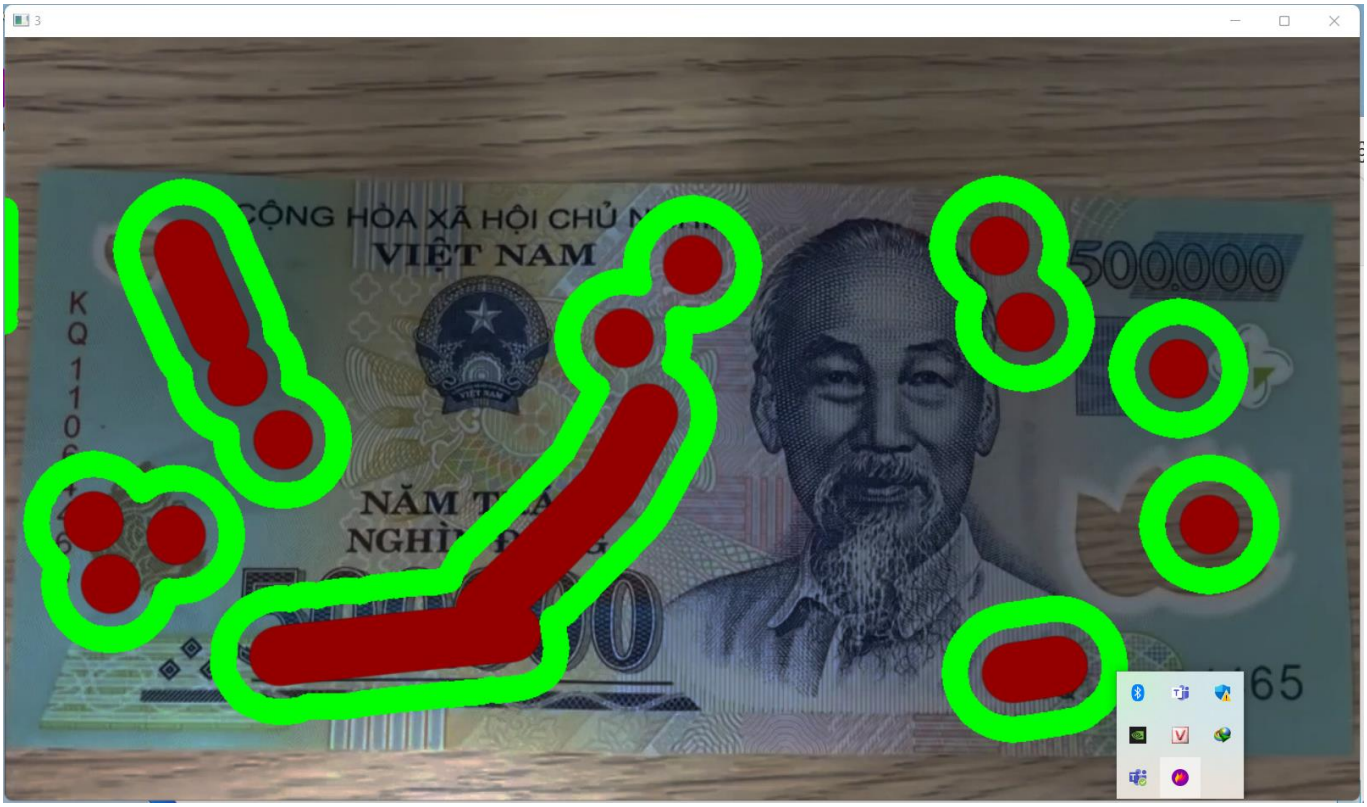
d. Thực hiện lọc:

```
for (ll i = 1; i < h - 1; i++) {// lọc đơn chấm
    for (ll j = 1; j < w - 1; j++) {
        if (tmp.at<uchar>(i - 1, j - 1) == 0 && tmp.at<uchar>(i - 1, j) == 0 && tmp.at<uchar>(i - 1, j + 1) == 0 &&
            tmp.at<uchar>(i, j - 1) == 0 && tmp.at<uchar>(i, j) == 255 && tmp.at<uchar>(i, j + 1) == 0 &&
            tmp.at<uchar>(i + 1, j - 1) == 0 && tmp.at<uchar>(i + 1, j) == 0 && tmp.at<uchar>(i + 1, j + 1) == 0)
            tmp.at<uchar>(i, j) = 0;
    }
}
```

e. Đánh dấu, vẽ đường bao quanh vết bẩn được xác định:

```
Mat white_img = img.clone();
for (ll i = 0; i < h; i++) {
    for (ll j = 0; j < w; j++) {
        white_img.at<uchar>(i, j) = 255;
    }
}
Mat obj = imread("7.jpg", IMREAD_COLOR);
Mat matElement = getStructuringElement(MORPH_RECT, Size(3, 3));//đánh dấu vết bẩn
morphologyEx(tmp, tmp, MORPH_ERODE, matElement);

vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(tmp, contours, hierarchy, RETR_TREE, CHAIN_APPROX_NONE);
drawContours(white_img, contours, -1, Scalar(0, 255, 0), 21);

tmp = white_img.clone();
tmp = Display_Canny(tmp);// lọc cạnh vết bẩn
//imshow("2.5", tmp);
matElement = getStructuringElement(MORPH_RECT, Size(1, 1));
morphologyEx(tmp, tmp, MORPH_OPEN, matElement);
vector<vector<Point>> contours1;
vector<Vec4i> v;
findContours(tmp, contours1, v, RETR_TREE, CHAIN_APPROX_NONE);
drawContours(obj, contours1, -1, Scalar(0, 255, 0), 7);// vẽ đường bao quanh vết bẩn
imshow("3", obj);
```

f. Kết quả sau khi chạy xong:



2. Bài toán tìm vết bẩn với tờ tiền có kích thước khác (to hoặc nhỏ hơn):

*__Mô tả bài toán__ : Đây là bài toán tìm kiếm vết bẩn xuất hiện trên 1 tờ tiền so với tờ tiền gốc ban đầu, tờ tiền có kích thước nhỏ hơn so với ảnh gốc.

*__Ý tưởng thuật toán__: Tương tự bài toán 1, lần này chúng ta chỉ chỉnh ảnh gốc về với kích thước của ảnh tờ tiền bẩn.

*Cài đặt:

a. Khởi tạo dữ liệu và chỉnh sửa kích thước của ảnh gốc:

```cpp
#include "Different_Images.h"

void Diff_Img_Processing() {
    Mat img = imread("1.jpg", IMREAD_COLOR);
    Mat cop = imread("7.jpg", IMREAD_COLOR);
    resize(img, img, Size(cop.cols, cop.rows), INTER_LINEAR);
    cvtColor(img, img, COLOR_BGR2GRAY);
    cvtColor(cop, cop, COLOR_BGR2GRAY);
    //imshow("img", img);
    ll h = img.rows, w = img.cols;
    Mat tmp = cop.clone();
```

b.Tạo ngưỡng cho 2 ảnh:

```cpp
threshold(img, img, 89, 255, THRESH_BINARY);
threshold(cop, cop, 50, 255, THRESH_BINARY);
```

c. Thực hiện phép trừ 2 ảnh và lưu vào tmp:

```cpp
for (ll i = 0; i < h; i++) {
    for (ll j = 0; j < w; j++) {
        tmp.at<uchar>(i, j) = abs(cop.at<uchar>(i, j) - img.at<uchar>(i, j));
    }
}
```

d. Thực hiện lọc:

```cpp
for (ll i = 1; i < h - 1; i++) {// lọc đơn chấm
    for (ll j = 1; j < w - 1; j++) {
        if (tmp.at<uchar>(i - 1, j - 1) == 0 && tmp.at<uchar>(i - 1, j) == 0 && tmp.at<uchar>(i - 1, j + 1) == 0 &&
            tmp.at<uchar>(i, j - 1) == 0 && tmp.at<uchar>(i, j) == 255 && tmp.at<uchar>(i, j + 1) == 0 &&
            tmp.at<uchar>(i + 1, j - 1) == 0 && tmp.at<uchar>(i + 1, j) == 0 && tmp.at<uchar>(i + 1, j + 1) == 0)
            tmp.at<uchar>(i, j) = 0;
    }
}
```

e. Đánh dấu, vẽ đường bao quanh vết bẩn được xác định:

```
Mat white_img = img.clone();
for (ll i = 0; i < h; i++) {
    for (ll j = 0; j < w; j++) {
        white_img.at<uchar>(i, j) = 255;
    }
}
Mat obj = imread("7.jpg", IMREAD_COLOR);
Mat matElement = getStructuringElement(MORPH_RECT, Size(3, 3));//đánh dấu vết bẩn
morphologyEx(tmp, tmp, MORPH_ERODE, matElement);

vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(tmp, contours, hierarchy, RETR_TREE, CHAIN_APPROX_NONE);
drawContours(white_img, contours, -1, Scalar(0, 255, 0), 21);

tmp = white_img.clone();
tmp = Display_Canny(tmp);// lọc cạnh vết bẩn
//imshow("2.5", tmp);
matElement = getStructuringElement(MORPH_RECT, Size(1, 1));
morphologyEx(tmp, tmp, MORPH_OPEN, matElement);
vector<vector<Point>> contours1;
vector<Vec4i> v;
findContours(tmp, contours1, v, RETR_TREE, CHAIN_APPROX_NONE);
drawContours(obj, contours1, -1, Scalar(0, 255, 0), 7);// vẽ đường bao quanh vết bẩn
imshow("3", obj);
```

f. Kết quả sau khi chạy xong:

3. Bài toán tìm vết bẩn với tờ tiền bị xoay và chứa vết bẩn trong đó:

*__Mô tả bài toán__ : Đây là bài toán tìm kiếm vết bẩn xuất hiện trên 1 tờ tiền so với tờ tiền gốc ban đầu, tờ tiền bị xoay so với ảnh gốc.

*Ý tưởng thuật toán:

a. Tạo tmp để clone ảnh gốc vào tmp.

b. Tạo vòng lặp biến k chạy từ 0 đến 360 (k thể hiện giá trị góc từ 0 đến 360), xoay tmp lần lượt theo k lần.

c. Tạo biến d = 0, tiếp theo đếm số điểm ảnh trong tmp giống trong ảnh bẩn, dùng biến d để đếm lại

d. Tạo biến maxx là biến lưu lại số lần d xuất hiện nhiều nhất ở bước 4.

e. Thực hiện lại vòng lặp như bước b và tiếp tục thực hiện lại bước c.

f. Nếu d bằng số lần xuất hiện maxx lớn nhất, xoay ảnh gốc góc k, lưu biến k, dừng vòng lặp.

=> k là góc tìm được để xoay ảnh đúng.

g. Thực hiện phần còn lại như bài toán 1.

*Cài đặt:

a. Khởi tạo dữ liệu:

```
void Rotated_Img_Processing() {
    Mat img = imread("C:/Users/vctha/Desktop/jpeg/Original.jpg", IMREAD_GRAYSCALE);
    Mat cop = imread("C:/Users/vctha/Desktop/jpeg/Rotated.jpg", IMREAD_GRAYSCALE);
    //ghep hinh vuong
    Mat sq = imread("C:/Users/vctha/Desktop/jpeg/black_img.jpg", IMREAD_GRAYSCALE);// khung hình vuông màu đen
    ll h = img.rows, w = img.cols;
```

b. Tạo 1 hàm xoay ảnh:

```
Mat rotate(Mat src, double angle)   //rotate function returning mat object with parametres imagefile and angle
{
    Mat dst;      //Mat object for output image file
    Point2f pt(src.cols / 2., src.rows / 2.);          //point from where to rotate
    Mat r = getRotationMatrix2D(pt, angle, 1.0);      //Mat object for storing after rotation
    warpAffine(src, dst, r, Size(src.cols, src.rows));  ///applie an affine transforation to image.
    return dst;         //returning Mat object for output image file
}
```

c. Thực hiện thuật toán tìm góc để xoay ảnh:

```
ll angle, d = 0, maxx = INT_MIN;
for (ll k = 0; k <= 360; k++) {// Thuật toán tìm góc
    Mat tmp = img.clone();
    tmp = rotate(tmp, k);
    d = 0;
    for (ll i = 0; i < 720; i++) {
        for (ll j = 0; j < 720; j++) {
            if (tmp.at<uchar>(i, j) == cop.at<uchar>(i, j)) d++;
        }
    }
    maxx = max(maxx, d);
}
for (ll k = 0; k <= 360; k++) {
    Mat tmp = img.clone();
    tmp = rotate(tmp, k);
    d = 0;
    for (ll i = 0; i < 720; i++) {
        for (ll j = 0; j < 720; j++) {
            if (tmp.at<uchar>(i, j) == cop.at<uchar>(i, j)) d++;
        }
    }
    if (d == maxx) {
        img = rotate(img, k);
        angle = k;// góc cần tìm đây
        break;
    }
}
```

d.Tạo ngưỡng cho 2 ảnh:

```
threshold(img, img, 89, 255, THRESH_BINARY);
threshold(cop, cop, 50, 255, THRESH_BINARY);
```

e. Thực hiện phép trừ 2 ảnh và lưu vào tmp:

```
for (ll i = 0; i < 720; i++) {// trừ 2 ảnh
    for (ll j = 0; j < 720; j++) {
        tmp.at<uchar>(i, j) = abs(cop.at<uchar>(i, j) - img.at<uchar>(i, j));
    }
}
```

f. Thực hiện lọc:

```
for (ll i = 1; i < 720 - 1; i++) {// Xử lí vết chấm
    for (ll j = 1; j < 720 - 1; j++) {
        if (tmp.at<uchar>(i - 1, j - 1) == 0 && tmp.at<uchar>(i - 1, j) == 0 && tmp.at<uchar>(i - 1, j + 1) == 0 &&
            tmp.at<uchar>(i, j - 1) == 0 && tmp.at<uchar>(i, j) == 255 && tmp.at<uchar>(i, j + 1) == 0 &&
            tmp.at<uchar>(i + 1, j - 1) == 0 && tmp.at<uchar>(i + 1, j) == 0 && tmp.at<uchar>(i + 1, j + 1) == 0)
            tmp.at<uchar>(i, j) = 0;
    }
}
```

g. Thực hiện ghép các ảnh vào khung đen lớn (đẹp hơn):

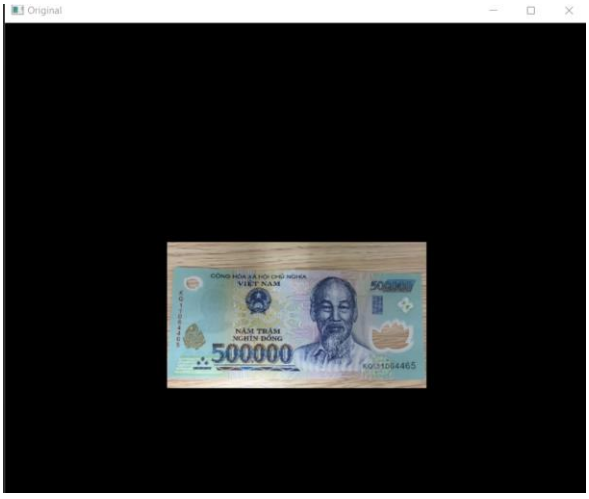
```
Mat org = imread("C:/Users/vctha/Desktop/jpeg/1.jpg", IMREAD_COLOR);//ghép ảnh gốc vào khung
h = org.rows, w = org.cols;
resize(org, org, Size(w / 4, h / 4), INTER_LINEAR);
h /= 4;
w /= 4;
sq = imread("C:/Users/vctha/Desktop/jpeg/black_img.jpg", IMREAD_COLOR);
for (ll i = (720 - h) / 2; i < (720 - h) / 2 + h; i++) {
    for (ll j = (720 - w) / 2; j < (720 - w) / 2 + w; j++) {
        ll temp = org.at<Vec3b>(i - (720 - h) / 2, j - (720 - w) / 2)[0];
        sq.at<Vec3b>(i, j)[0] = temp;
        temp = org.at<Vec3b>(i - (720 - h) / 2, j - (720 - w) / 2)[1];
        sq.at<Vec3b>(i, j)[1] = temp;
        temp = org.at<Vec3b>(i - (720 - h) / 2, j - (720 - w) / 2)[2];
        sq.at<Vec3b>(i, j)[2] = temp;
    }
}
imshow("Original", sq);
Mat rot = imread("C:/Users/vctha/Desktop/jpeg/4.jpg", IMREAD_COLOR);//ghép ảnh bị xoay vào khung
h = rot.rows, w = rot.cols;
resize(rot, rot, Size(w / 4, h / 4), INTER_LINEAR);
h /= 4;
w /= 4;
sq = imread("C:/Users/vctha/Desktop/jpeg/black_img.jpg", IMREAD_COLOR);
for (ll i = (720 - h) / 2; i < (720 - h) / 2 + h; i++) {
    for (ll j = (720 - w) / 2; j < (720 - w) / 2 + w; j++) {
        ll temp = rot.at<Vec3b>(i - (720 - h) / 2, j - (720 - w) / 2)[0];
        sq.at<Vec3b>(i, j)[0] = temp;
        temp = rot.at<Vec3b>(i - (720 - h) / 2, j - (720 - w) / 2)[1];
        sq.at<Vec3b>(i, j)[1] = temp;
        temp = rot.at<Vec3b>(i - (720 - h) / 2, j - (720 - w) / 2)[2];
        sq.at<Vec3b>(i, j)[2] = temp;
    }
}
imshow("Rotated", rotate(sq, 30));


Mat obj = imread("C:/Users/vctha/Desktop/jpeg/4.jpg", IMREAD_COLOR);//ghép ảnh bị xoay vào khung lần nữa
h = obj.rows, w = obj.cols;
resize(obj, obj, Size(w / 4, h / 4), INTER_LINEAR);
h /= 4;
w /= 4;
sq = imread("C:/Users/vctha/Desktop/jpeg/black_img.jpg", IMREAD_COLOR);
for (ll i = (720 - h) / 2; i < (720 - h) / 2 + h; i++) {
    for (ll j = (720 - w) / 2; j < (720 - w) / 2 + w; j++) {
        ll temp = obj.at<Vec3b>(i - (720 - h) / 2, j - (720 - w) / 2)[0];
        sq.at<Vec3b>(i, j)[0] = temp;
        temp = obj.at<Vec3b>(i - (720 - h) / 2, j - (720 - w) / 2)[1];
        sq.at<Vec3b>(i, j)[1] = temp;
        temp = obj.at<Vec3b>(i - (720 - h) / 2, j - (720 - w) / 2)[2];
        sq.at<Vec3b>(i, j)[2] = temp;
    }
}
```

h. Đánh dấu, vẽ đường bao quanh vết bẩn được xác định:

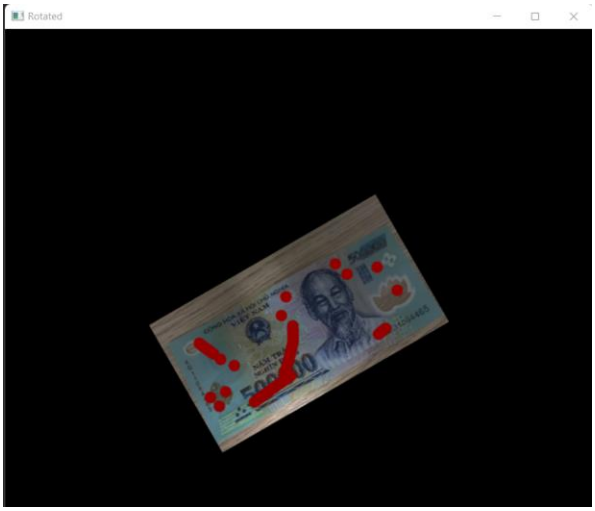
```
sq = rotate(sq, angle);
Mat white_img = img.clone();//tạo ảnh trắng
for (ll i = 0; i < 720; i++) {
    for (ll j = 0; j < 720; j++) {
        white_img.at<uchar>(i, j) = 255;
    }
}
tmp = Display_Serial(white_img, tmp);//lấy viền vết bẩn
tmp = Display_Canny(tmp);//lọc cho viền mịn hơn
//imshow("tmp", tmp);
Mat matElement = getStructuringElement(MORPH_RECT, Size(1, 1));// công đoạn khoanh vùng vết bẩn
morphologyEx(tmp, tmp, MORPH_OPEN, matElement);
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(tmp, contours, hierarchy, RETR_TREE, CHAIN_APPROX_NONE);
drawContours(sq, contours, -1, Scalar(0, 255, 0), 6);
sq = rotate(sq, -angle);
imshow("My result", sq);
```

i. Kết quả:

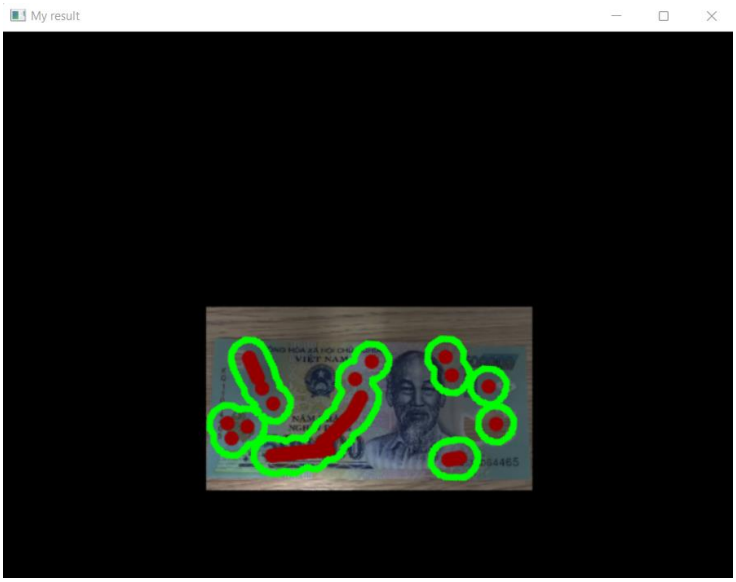
Ảnh gốc:



Ảnh bẩn:



Ảnh xác định vết bẩn:



4. Bài toán xác định vết bẩn với 2 tờ tiền có serie khác nhau:

*__Mô tả bài toán__ : Đây là bài toán tìm kiếm vết bẩn xuất hiện trên 1 tờ tiền so với tờ tiền gốc ban đầu, tờ tiền bẩn có số serie khác hoàn toàn so với ảnh gốc.

Ý tưởng thuật toán:

a. Tạo biến tmp làm clone cho ảnh gốc.

b. Chạy vòng lặp, gán biến x cho giá trị pixel trong ảnh gốc.

c. Sử dụng bảng băm cnt1 để đếm số lần xuất hiện của x.

d. Chuyển pixel của tmp lên 255.

e. Chạy vòng lặp, nếu bảng băm cnt1 không phát hiện được giá trị pixel trong ảnh bẩn, ta gán giá trị của ảnh bẩn cho biến tmp.

f. Chạy vòng lặp, nếu giá trị pixel của tmp không bằng 5 hoặc 255 hoặc 254, ta gán các giá trị đó bằng 255 (đưa về màu trắng).

g. Chạy thêm vòng lặp để lọc các vết chấm.

h. Dựa vào thuật toán loang, chúng ta sẽ loang ra các chỗ có vết bẩn và đánh dấu chúng, sử dụng 2 vòng lặp.

i. Thực hiện đánh dấu bằng contour như các bài toán trước.

***Cài đặt:**

a. Khởi tạo dữ liệu:

```
id Serial_Diff_Processing() {
    Mat img = imread("C:/Users/vctha/Desktop/jpeg/1.jpg", IMREAD_COLOR);
    Mat cop = imread("C:/Users/vctha/Desktop/jpeg/6.jpg", IMREAD_COLOR);
    imshow("img", img);
    imshow("cop", cop);
    Mat v = cop.clone();
    cvtColor(img, img, COLOR_BGR2GRAY);
    cvtColor(cop, cop, COLOR_BGR2GRAY);
    Mat tmp = img.clone();
    ll h = img.rows, w = img.cols, max1 = INT_MIN, max2 = INT_MIN;
```

b. Thực hiện dùng bảng băm để giải thuật:

```
for (ll i = 0; i < h; i++) {
    for (ll j = 0; j < w; j++) {
        ll x = img.at<uchar>(i, j);
        cnt1[x]++;
        tmp.at<uchar>(i, j) = 255;
    }
}
```

c. Tìm giá trị của cop trong bảng băm:

```
cop = Equalize_Hist_Red(cop);
for (ll i = 0; i < h; i++) {
    for (ll j = 0; j < w; j++) {
        if (cnt1[cop.at <uchar>(i, j)] == 0) tmp.at<uchar>(i, j) = cop.at <uchar>(i, j);
    }
}
```

d. Đánh dấu các vị trí được thêm vào tmp trong bảng băm cnt:

```
for (ll i = 0; i < h; i++) {
    for (ll j = 0; j < w; j++) {
        cnt[tmp.at<uchar>(i, j)]++;
    }
}
for (ll i = 0; i < 256; i++) cout << cnt[i] << " ";
```

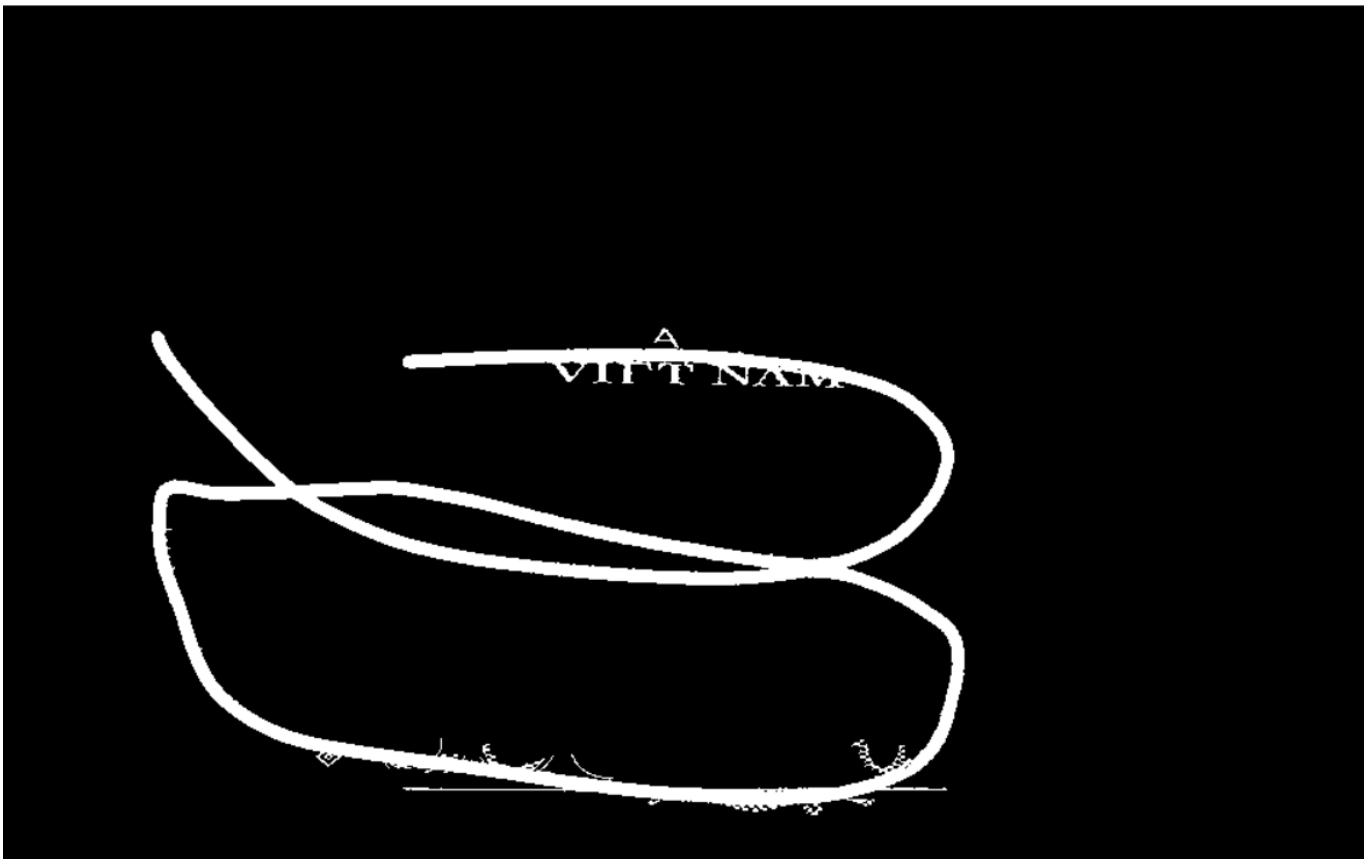
Ra được các mốc và giá trị của chúng như sau:

```
3407 3789 3065 3803 3440 3262 0 0 2942 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3730 894162
```

Dễ thấy các mốc có giá trị khác 0 đại diện cho số lần xuất hiện của giá trị pixel ở ảnh bản trong ảnh tmp. Lần lượt lấy vị trí của các giá trị khác 0, ta có giá trị thứ 5 cho ra ảnh gần với kết quả mong muốn nhất. Cài đặt thuật toán dưới và cho ra hình ảnh như sau:

```
or (ll i = 0; i < h; i++) {
   for (ll j = 0; j < w; j++) {
      if (tmp.at<uchar>(i, j) != 5 && tmp.at<uchar>(i, j) != 255 && tmp.at<uchar>(i, j) != 254) tmp.at<uchar>(i, j) = 255;
   }
```

tmp



e. Sử dụng contours để đánh dấu một ít lên vết bẩn rồi áp dụng thuật toán loang để đưa dấu tràn khắp vết bẩn trên tờ tiền bẩn.

```
Mat matElement = getStructuringElement(MORPH_RECT, Size(1, 1));
morphologyEx(tmp, tmp, MORPH_CLOSE, matElement);
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(tmp, contours, hierarchy, RETR_TREE, CHAIN_APPROX_NONE);
drawContours(cop, contours, -1, Scalar(0, 255, 0), 3);
//tmp = Display_Canny(tmp);
threshold(tmp, tmp, 127, 255, THRESH_BINARY_INV);
matElement = getStructuringElement(MORPH_RECT, Size(1, 3));
morphologyEx(tmp, tmp, MORPH_ERODE, matElement);
vector<vector<Point>> contours1;
vector<Vec4i> hierarchy1;
findContours(tmp, contours1, hierarchy1, RETR_TREE, CHAIN_APPROX_NONE);
drawContours(v, contours1, -1, Scalar(0, 255, 0), 3);
Mat var = imread("C:/Users/vctha/Desktop/jpeg/6.jpg", IMREAD_GRAYSCALE);
threshold(var, var, 69, 255, THRESH_BINARY);
for (ll i = 0; i < h; i++) {
    for (ll j = 0; j < w; j++) {
        if (tmp.at<uchar>(i, j) != 0) {
            if (var.at<uchar>(i, j) == var.at<uchar>(i - 1, j - 1)) tmp.at<uchar>(i - 1, j - 1) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i - 1, j)) tmp.at<uchar>(i - 1, j) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i - 1, j + 1)) tmp.at<uchar>(i - 1, j + 1) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i, j - 1)) tmp.at<uchar>(i, j - 1) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i, j + 1)) tmp.at<uchar>(i, j + 1) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i + 1, j - 1)) tmp.at<uchar>(i + 1, j - 1) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i + 1, j)) tmp.at<uchar>(i + 1, j) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i + 1, j + 1)) tmp.at<uchar>(i + 1, j + 1) = 255;
        }
```

```
for (ll i = h - 1; i >= 0; i--) {
    for (ll j = w - 1; j >= 0; j--) {
        if (tmp.at<uchar>(i, j) != 0) {
            if (var.at<uchar>(i, j) == var.at<uchar>(i - 1, j - 1)) tmp.at<uchar>(i - 1, j - 1) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i - 1, j)) tmp.at<uchar>(i - 1, j) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i - 1, j + 1)) tmp.at<uchar>(i - 1, j + 1) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i, j - 1)) tmp.at<uchar>(i, j - 1) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i, j + 1)) tmp.at<uchar>(i, j + 1) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i + 1, j - 1)) tmp.at<uchar>(i + 1, j - 1) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i + 1, j)) tmp.at<uchar>(i + 1, j) = 255;
            if (var.at<uchar>(i, j) == var.at<uchar>(i + 1, j + 1)) tmp.at<uchar>(i + 1, j + 1) = 255;
        }

    }
}
matElement = getStructuringElement(MORPH_RECT, Size(1, 1));
morphologyEx(tmp, tmp, MORPH_ERODE, matElement);
vector<vector<Point>> contours2;
vector<Vec4i> hierarchy2;
findContours(tmp, contours2, hierarchy2, RETR_TREE, CHAIN_APPROX_NONE);
drawContours(v, contours2, -1, Scalar(0, 255, 0), 3);
imshow("tmp", tmp);
imshow("v", v);
```

Kết quả ảnh thu được như sau:

5. **Bài toán xác định vết bẩn với 2 tờ tiền. Tờ thứ 2 bị rách/ hư hỏng:**

***Mô tả bài toán** : Đây là bài toán tìm kiếm vết bẩn xuất hiện trên 1 tờ tiền so với tờ tiền gốc ban đầu, tờ tiền bẩn bị hư hỏng 1 phần so với ảnh gốc.

Ý tưởng thuật toán:

a. Tạo bảng băm fre. Sử dụng bảng băm fre đếm số lần xuất hiện từng pixel của ảnh gốc.

b. Tách background, duyệt và gán màu trắng cho đến khi gặp viền tờ tiền.

c. Lọc các giá trị pixel ít xuất hiện hoặc không có trong ảnh gốc

d. Chọn ra những pixel xuất hiện nhiều nhất trong tmp

***Cài đặt:**

a. Khởi tạo dữ liệu:

```
ll fre[256];

void Torn_Money_Processing() {
    Mat img = imread("To-tien-50-nghin-HD.jpg", IMREAD_GRAYSCALE);
    Mat cop = imread("50k_rach1.jpg", IMREAD_GRAYSCALE);
    //img = Equalize_Hist_Red(img);
    //cop = Equalize_Hist_Red(cop);
    //threshold(cop, cop, 67, 255, THRESH_BINARY);
    Mat tmp = cop.clone();
```

b. Thực hiện thuật toán tách background tìm vết bẩn:

```
for (ll i = 0; i < img.rows; i++) {// dem so lan xuat hien tung pixel cua img
    for (ll j = 0; j < img.cols; j++) {
        fre[img.at<uchar>(i, j)] += 1;
    }
}
////////tach background
for (ll i = 0; i < cop.rows; i++) {
    for (ll j = 0; j < cop.cols; j++) {
        if (fre[cop.at<uchar>(i, j)] < 1000) cop.at<uchar>(i, j) = 255;
        else break;
    }
}
for (ll i = cop.rows - 1; i >= 0; i--) {
    for (ll j = cop.cols - 1; j >= 0; j--) {
        if (fre[cop.at<uchar>(i, j)] < 1000) cop.at<uchar>(i, j) = 255;
        else break;
    }
}//////

////loc ra nhung pixel ma it xuat hien hoac khong co trong img
for (ll i = 0; i < cop.rows; i++) {
    for (ll j = 0; j < cop.cols; j++) {
        if (fre[cop.at<uchar>(i, j)] == 0 || fre[cop.at<uchar>(i, j)] < 50) tmp.at<uchar>(i, j) = cop.at<uchar>(i, j);
        else {
            tmp.at<uchar>(i, j) = 255;
        }
    }
}
/////////

//////chon ra nhung pixel xuat hien nhieu nhat trong tmp
memset(fre, 0, sizeof(fre));
for (ll i = 0; i < cop.rows; i++) {
    for (ll j = 0; j < cop.cols; j++) {
        fre[tmp.at<uchar>(i, j)]++;
    }
}
for (ll i = 0; i < tmp.rows; i++) {
    for (ll j = 0; j < tmp.cols; j++) {
        if (fre[tmp.at<uchar>(i, j)] < 700) tmp.at<uchar>(i, j) = 255;
    }
}
```

c. Đánh dấu, vẽ đường bao quanh vết bẩn được xác định:

```
threshold(tmp, tmp, 150, 255, THRESH_BINARY);
tmp = Display_Canny(tmp);

Mat res = imread("50k_rach1.jpg", IMREAD_COLOR);// bat dau highlight len anh goc

Mat matElement = getStructuringElement(MORPH_RECT, Size(1, 1));
morphologyEx(tmp, tmp, MORPH_ERODE, matElement);
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(tmp, contours, hierarchy, RETR_TREE, CHAIN_APPROX_NONE);
drawContours(res, contours, -1, Scalar(0, 255, 0), 5);
///////

img = imread("To-tien-50-nghin-HD.jpg", IMREAD_COLOR);
cop = imread("50k_rach1.jpg", IMREAD_COLOR);
imshow("Original_Money_Image", img);
imshow("Torn_Money_Image", cop);
imshow("My result", res);
```

d. Kết quả nhận được: