

**ng d ạng x ử lý ảnh trong th ực th ực v i th
vi n OpenCV C/C++**

Nguyễn Văn Long
long.06clc@gmail.com

M u

Xử lý ảnh và th giác máy là lĩnh vực mà ngày nay có phát triển và ứng dụng rất rộng rãi trong nhiều lĩnh vực khác nhau như vào sản xuất công nghiệp, các thuỷ toán và công trình nghiên cứu khác nhau của nhiều nhà khoa học trên thế giới.

Việt Nam, các ngành kỹ thuật như kỹ thuật điện tử, kỹ thuật ô tô, kỹ thuật khai thác mỏ, kỹ thuật hàng không, kỹ thuật biển, kỹ thuật hàng không vũ trụ, kỹ thuật công nghệ thông tin, kỹ thuật viễn thông... Tuy nhiên nhìn một cách khách quan thì số lượng các ngành kỹ thuật khai thác mỏ là quá ít ỏi, lĩnh vực này vẫn còn phát triển chậm trong tương lai nếu không có quan tâm đến cách nghiêm túc.

Xuất phát từ nhu cầu kỹ thuật môn học kỹ thuật chung mà không chỉ vào ban chuyên ngành, kỹ thuật công nghệ thông tin, kỹ thuật viễn thông Việt là không nhiều, nhưng quá trình nghiên cứu nghiêm túc, kinh nghiệm thực tế tác giả đã cố gắng cho ra cuốn sách *nguyên lý ảnh trong thực tiễn viễn thông OpenCV*.

Cuốn sách nhằm mục đích giới thiệu, không xa vào những công thức toán học trừu tượng, phân tích toán học, sinh viên nắm bắt môn học một cách chung chung mà không i vào bài toán, không hiểu rõ lý do tại sao. Vì vậy không ai muốn đọc sách *nguyên lý ảnh trong thực tiễn viễn thông OpenCV*. Các chương trong cuốn sách này đều kèm ví dụ minh họa mô phỏng và viết bằng ngôn ngữ C++ và giúp cách viết OpenCV, mà thường viết mã nguồn mà có ảnh giá là một nhánh vật lý áp dụng các công nghệ trong thời gian thực.

Cuốn sách được chia thành bốn phần, phần đầu giới thiệu về OpenCV, phần thứ hai nói về mảng, phần thứ ba nói về chisel, phần thứ tư nói về MFC và phần cuối cùng nói về mảng, phần thứ năm nói về bài toán nhận diện biển số xe...

Cuốn sách không chỉ là tài liệu tham khảo bổ ích trong quá trình học tập của các bạn sinh viên, quá trình làm luận văn, án... mà còn là công cụ hỗ trợ cho việc triển khai các ứng dụng thương mại và doanh nghiệp và những ứng dụng quan tâm tinh tế.

Cuối cùng dù đã dành nhiều tâm huyết hoàn thành cuốn sách nhưng chưa chia sẻ cho nhiều người, mong các bạn góp ý và phản hồi. Xin gửi lời chúc mừng và cảm ơn sâu sắc đến tác giả.

H ạng d ẫn s ố ng sách

Cuốn sách cung cấp một số bài toán nghiên cứu và quá trình làm việc cụ thể của tác giả, ví dụ như trong sách bạn có thể thấy qua những bài tập cốt yếu chính, sau đó có thể tìm thêm tài liệu nâng cao hơn và có thể tham khảo thêm vào mục 참고 프로그램, source code kèm.

Thư viện OpenCV cung cấp trong sách là bản OpenCV 2.4.3, với các bản OpenCV khác thì bạn có thể tùy chỉnh một chút tuy nhiên vẫn chưa có sẵn là tương ứng gì nhau. Ngôn ngữ lập trình cho các ví dụ là C/C++, IDE sử dụng là Visual Studio 2010. Tuy nhiên có thể chia thành trong cuốn sách này có một tách biệt phần lý chính ra vào một file *.cpp nào nên ta có thể lấy nó áp dụng vào các trình điều khiển khác.

Có 10 chương chính bao quát một số khía cạnh của lĩnh vực xử lý hình ảnh khá chi tiết và giải thích kỹ, 3 project của tác giả mô tả chung chung hơn. Do đó bạn cầnучathànhquenvới thư viện OpenCV nên cùnghợp lý theo thời gian để hiểu rõ.

Trong cuốn sách có nhiều ví dụ liên quan tới thuật lập trình không do phỏng vấn tác giả mà có thể nói qua về một số khía cạnh, trên thực tế có nhiều cách khác nhau giải quyết cùng một công việc, ví dụ như lập trình bằng cách rõ ràng có thể tham khảo thêm tài liệu của người khác hoặc giải quyết theo hướng mà bạn cảm thấy là thuận tiện nhất.

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Mục lục

Chương I. Làm quen với thư viện OpenCV

| | |
|---|---|
| 1. Giới thiệu về thư viện OpenCV | 5 |
| 2. Phiên bản OpenCV 1 hay OpenCV 2 | 5 |
| 3. Hướng dẫn sử dụng OpenCV trên Window | 6 |

Chương II. Các phép xử lý ảnh trong OpenCV

| | |
|--|----|
| 1. Chương trình ưu tiên | 12 |
| 2. Không gian màu, chuyển đổi không gian màu | 13 |
| 3. Điều chỉnh sáng, tăng phím | 17 |
| 4. Nhận diện phân, nhận phân hóa vùng ảnh | 19 |
| 5. Histogram, cân bằng histogram | 23 |
| 6. Phóng to, thu nhỏ, xoay ảnh | 27 |
| 7. Lọc trong ảnh | 30 |
| 8. Các phép toán hình thái học trong ảnh | 37 |
| 9. Tìm biên ảnh với bài Canny | 43 |
| 10. Chuyển đổi Hough, Phát hiện đường tròn trong ảnh | 46 |

Chương III. Lập trình xử lý ảnh với giao diện MFC

| | |
|--|----|
| 1. Giới thiệu MFC | 51 |
| 2. Khởi tạo project MFC | 51 |
| 3. Làm việc với các điều khiển (Control) | 54 |
| 4. Chuyển đổi các kiểu dữ liệu trong MFC | 59 |
| 5. Chương trình tính và hiển thị lên giao diện MFC | 61 |

Chương IV. Mô tả ứng dụng trong thực tế

| | |
|--|----|
| 1. My Photo Editor, phần mềm chỉnh sửa ảnh | 64 |
| 2. Nhận diện biển xe | 73 |
| 3. MyCam, mô tả hiển thị ảnh trong video | 90 |

Chương I. Làm quen với thư viện OpenCV

1. Giới thiệu về thư viện OpenCV

OpenCV (Open Source Computer Vision) là một thư viện mã nguồn mở về giải máy vision có 500 hàm và hơn 2500 các thuật toán để xử lý ảnh, và các vấn đề liên quan đến giải máy. OpenCV cung cấp một cách tiếp cận, sử dụng các thành phần của các dòng chip như... để thực hiện các phép tính toán trong thời gian thực, nghĩa là có thể áp dụng cho nó có thể nhanh cho các ứng dụng thông thường. OpenCV là thư viện cung cấp khả năng làm việc trên nhiều nền tảng khác nhau (cross-platform), nghĩa là nó có thể chạy trên hệ điều hành Window, Linux, Mac, iOS... Vì thế, OpenCV tuân theo các quy định về sử dụng phần mềm mã nguồn mở BSD do tổ chức có thể sử dụng miễn phí cho mục đích phi thương mại или не для коммерческих целей.

Đầu tiên, OpenCV được khởi động vào năm 1999, năm 2000 nó được giới thiệu trong một hội nghị của IEEE về các vấn đề trong giải máy và nhận được, tuy nhiên trong OpenCV 1.0 mãi đến tháng 11 năm 2006 mới chính thức công bố và năm 2008 bản 1.1 (pre-release) mới được ra mắt. Tháng 10 năm 2009, bản OpenCV thứ hai ra mắt (tháng này là phiên bản 2.x), phiên bản này có giao diện C++ (khác với phiên bản trước có giao diện C) và có khá nhiều điểm khác biệt so với phiên bản trước.

Thư viện OpenCV ban đầu có sự hỗ trợ Intel, sau đó có sự hỗ trợ Willow Garage, một phòng thí nghiệm chuyên nghiên cứu và công nghệ robot. Cho đến nay, OpenCV vẫn là thư viện miễn phí, có phát triển bởi một nhóm không lợi nhuận (non-profit foundation) và có sự đóng góp từ các cá nhân.

2. Phiên bản OpenCV 1 hay OpenCV 2?

Cho đến nay, trải qua hơn 6 năm từ lúc phiên bản OpenCV đầu tiên được công bố, đã có lần lượt nhiều phiên bản OpenCV ra mắt, tuy nhiên có thể chia thành hai bản chính dựa trên những điểm khác biệt nhỏ như sau: phiên bản OpenCV thứ nhất (hay còn gọi là phiên bản OpenCV 1.x) và phiên bản OpenCV thứ hai (hay còn gọi là phiên bản OpenCV 2.x). Sau đây ta sẽ chỉ ra một số điểm khác biệt của hai phiên bản này.

- OpenCV 1.x (bao gồm bản 1.0 và bản pre-release 1.1) được xây dựng trên giao diện C, có trúc cấu trúc dữ liệu trên cấu trúc class `IplImage`, trong khi đó OpenCV 2.x được xây dựng trên giao diện C++, có trúc cấu trúc dữ liệu trên cấu trúc class `cv::Mat`.
- Trong OpenCV 1.x, người dùng phải hoàn toàn quản lý bộ nhớ của các biến, nghĩa là khi muốn truy cập một mảng dữ liệu, ta phải luôn chú ý giữ phỏng nó khi không còn sử dụng nữa (trong khi đó trong OpenCV 2.x việc quản lý bộ nhớ được dễ dàng hơn nhờ các hàm hỗ trợ các lớp riêng trong OpenCV 2.x). Điều này khi mà tiếp tục không còn cần sử dụng nữa.

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

- Vì có rất nhiều thay đổi cùng một cách mà không trong OpenCV 2.x là đã thay đổi hoàn toàn so với OpenCV 1.x, một phần là giao diện C++ có phần định nghĩa tên so với C, một phần là các hàm trong OpenCV 2.x đã có tính chất khác nhau bao gồm thời gian không cần thiết và thời gian giao diện ngắn hơn. Chẳng hạn ta hãy xét ví dụ về việc phát hiện đường tròn trong hình ảnh đã vào thuật toán Hough, các bước cần làm là load hình ảnh, chuyển sang thành phần, tìm biên và trên bước canny và phát hiện đường tròn đã trên thuật toán Hough. OpenCV 1.x thực hiện như sau:

```
// Phát hiện đường tròn trong ảnh OpenCV 1.x
IplImage* src = cvLoadImage("image.jpg");
IplImage* gray = cvCreateImage(cvGetSize(src), IPL_DEPTH_8U, 1);
cvCvtColor(src, gray, CV_BGR2GRAY);
cvCanny(gray, gray, 10, 30, 3);
CvMemStorage* storage = cvCreateMemStorage(0);
CvSeq* circles = cvHoughCircles(gray, storage, CV_HOUGH_GRADIENT, 1,
50, 100, 50);
```

Trong khi đó, OpenCV 2.x thực hiện như sau:

```
// Phát hiện đường tròn trong ảnh OpenCV 1.x
Mat src = imread("image.jpg");
Mat gray;
CvtColor(src, gray, CV_BGR2GRAY);
Canny(gray, gray, 10, 30, 3);
Vector<Vec3f> circles;
HoughCircles(gray, circles, CV_HOUGH_GRADIENT, 1, 50, 100, 50);
```

Ta thấy rằng lệnh `gray` trong OpenCV 2.x không cần phải khởi tạo,而是 `storage` (không có ý nghĩa về mặt sử dụng) cũng không cần phải khởi tạo (và do đó không cần gán giá trị).

- Thay đổi OpenCV 1.x tuy chỉ am hiểu lõi lý và thuật toán, tuy nhiên nó vẫn dùng số khai. Thay đổi OpenCV 2.x là có bối cảnh khá nhiều thay đổi, thuật toán và cách tiếp cận khác nhau có biệt trong các khía cạnh phát hiện (detection), nhận dạng (partten recognition) và theo dõi (tracking). Hiện nay, tuy có giao diện C++ như OpenCV 2.x vẫn dùng phần giao diện C++ thích với các phiên bản của OpenCV 1.x ...

Tóm tắt có thể thấy rằng phiên bản 2.x là có nhiều cải tiến so với phiên bản 1.x, tuy nhiên trong một số trường hợp có các thay đổi nhưng khi mà trình điều khiển thuần C++ không còn giá trị. Trong cuốn sách này, các nội dung cài đặt, thuật toán, ứng dụng ... chia thành cho OpenCV phiên bản 2.x trên một hướng iu hành Window.

3. Hướng dẫn sử dụng thư viện OpenCV trên Window

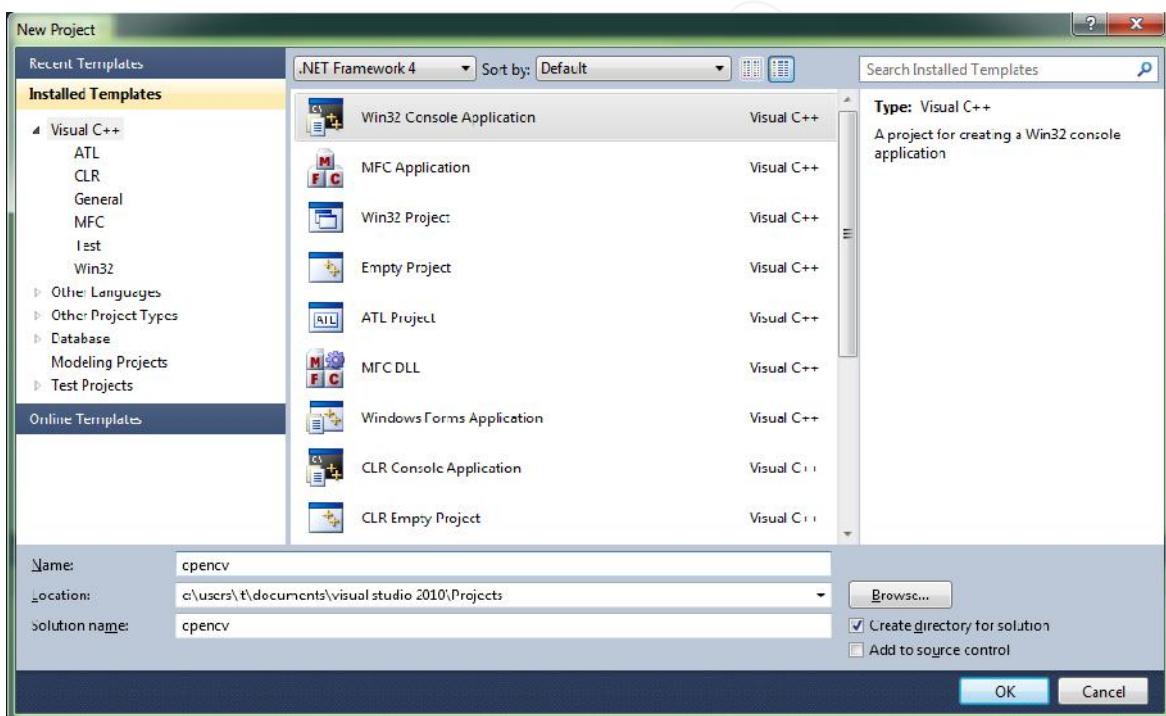
Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Trước hết ta cần download thư viện OpenCV về máy tính, thường là luôn download bản mới nhất tại <http://sourceforge.net/projects/opencvlibrary/>. Chẳng hạn để build sẵn phù hợp với hành lang dùng, bản OpenCV có sẵn trong cuốn sách này là bản 2.4.3 và lần update cuối cùng là vào ngày 25 tháng 12 năm 2012. Sau khi download về máy, tiến hành cài đặt bình thường, там можно установить в папку C:\opencv. Tiếp theo ta sẽ tiến hành tùy chỉnh có thể làm việc với OpenCV qua hai IDE thông dụng là Microsoft Visual Studio và Eclipse CDT.

Trên Microsoft Visual Studio

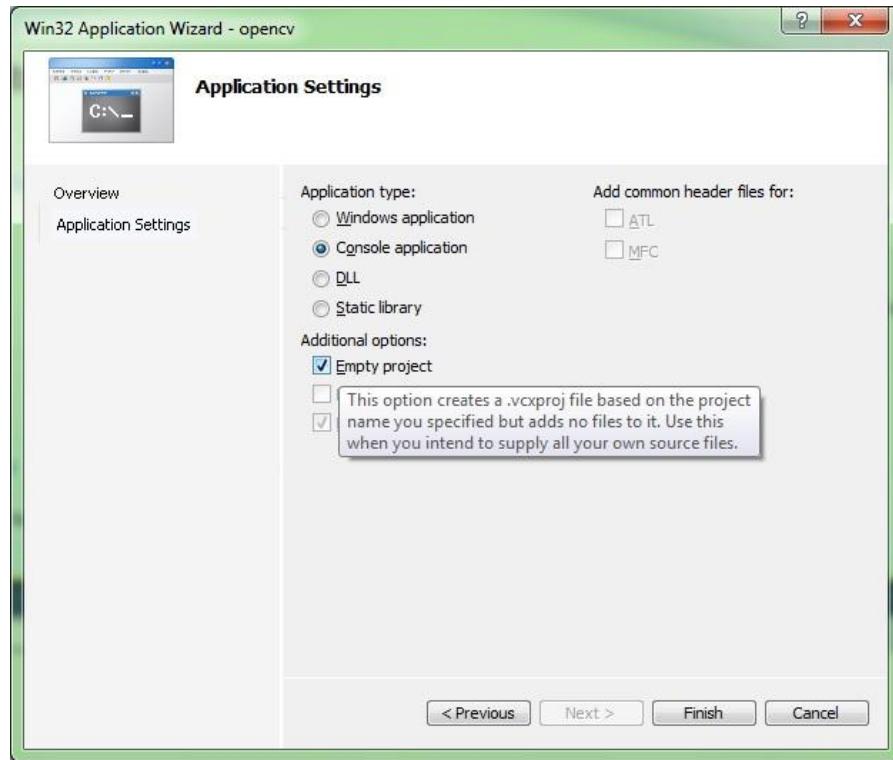
Phiên bản Visual studio sử dụng đây là phiên bản Visual Studio 2010, các phiên bản trước ta hoàn toàn có thể cài hình cách tiếp theo.

Tạo một project mới: New > Project, trong cửa sổ New Project chọn Visual C++, Win32 console application. tên project là opencv



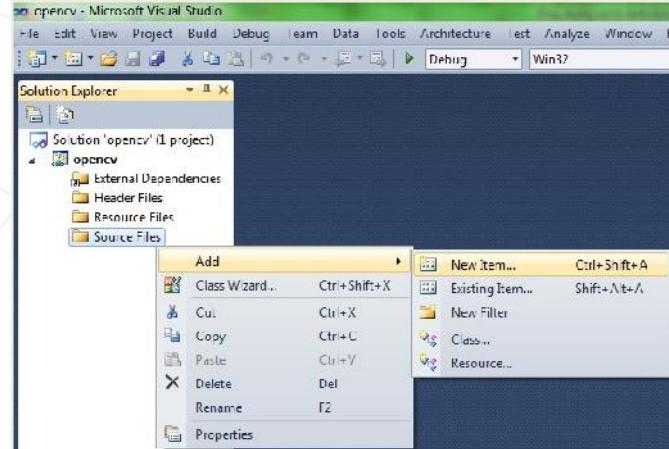
Chọn OK, sau đó nhấp Next, hộp thoại tiếp theo xuất hiện, hộp thoại này ta chọn *Application type* là *Console application* và *Additional option* là *Empty project*, nhấn Finish kết thúc quá trình khởi tạo.

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



Project mới sẽ tạo ra là project hoàn toàn trống, ta phải thêm vào ít nhất một file nguồn và chương trình có thể chạy, trong *Solution Explorer* ta click chuột phải vào *Source Files*, chọn *Add -> New Item...*. Khi đó *Add New Item* hiện ra, ta chọn kiểu cần thêm vào là *C++ File(.cpp)* và tên cho file thêm vào, giả sử là *FirstApp.cpp*. Vậy giờ trong file này ta có thể thêm vào các *#include* và ghi hàm *main()* để chạy chương trình.

Chương trình có thể chạy được và có thể viết với OpenCV ta có tùy chỉnh thêm một số thuộc tính của project như sau:



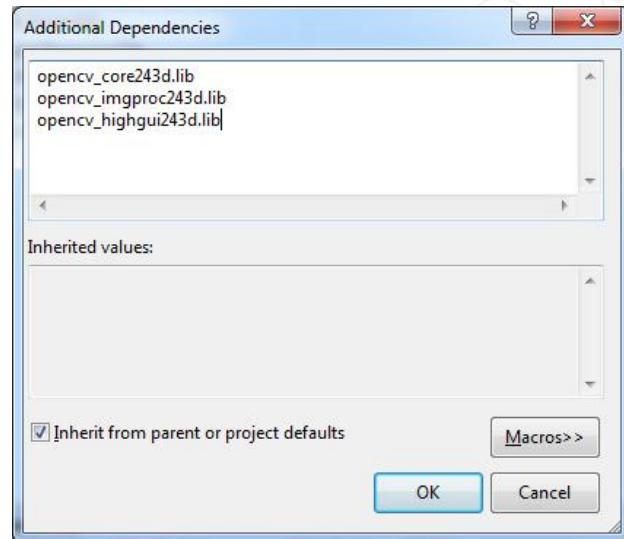
Vào Project -> Properties (hoặc nhấn phím Alt + F7) để mở Properties. Khi đó *opencv Property Pages* hiện ra, trong mục *Configuration Properties* chọn *VC++ Directories*, trong ô bên phải, ta tìm mục *Include Directories* và *Library Directories*. Ta sẽ chọn đường dẫn hai mục này là các đường dẫn ngang của viễn OpenCV.

Mục *Include Directories*, ta tùy chỉnh ô bên phải là *C:\opencv\build\include*. Mục *Library Directories* trong mục *C:\opencv\build\x86\vc10\lib* nếu là ta sử dụng hệ điều hành 32bit hoặc *C:\opencv\build\x64\vc10\lib* cho hệ điều hành 64bit.

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Tiếp theo, trong hộp thoại *opencv Property Pages -> Configuration Properties -> Linker* chọn *Input*, trong tab *Additional Dependencies* thêm vào các giá trị cho mục *Additional Dependencies* là *opencv_core243d.lib, opencv_imgproc243d.lib, opencv_highgui243d.lib*.

Chú ý là các lib thêm vào sẽ không ngang với các header ta khai báo trong chương trình, và tùy thuộc vào mục đích sử dụng mà ta có thể thêm vào các lib các nhau, giống như khi sử dụng tới các hàm về video, khi đó ta thêm header `#include <opencv2/video/video.hpp>` thì trong phần *Additional Dependencies* ta phải khai báo thêm *opencv_video243d.lib*. Chỗ này cần chú ý các file trên thường ta copy sang folder *debug*, ta có thể thêm các lib không có chữ “*d*” vào i như *opencv_core243.lib ...* trong *release*. Tuy nhiên khi sang còn hỗ trợ và cần những điều kiện như sau để debug.



Cụ thể, khi đã chép xong tất cả các file, nó có thể chạy được trên các file **.dll*. Cách này gần nhất là ta copy các file **.dll* sang folder (như *opencv_core243d.dll, opencv_imgproc243d.dll*) vào thư mục chứa file chạy của chương trình (file **.exe*). Các file **.dll* này nằm trong mục *C:\opencv\build\x86\bin* và win 32 bit hoặc *C:\opencv\build\x64\bin* và win 64 bit. Với các phiên bản OpenCV cũ hơn, ta cần copy luôn file *tbb_debug.dll* (trong *debug*) hoặc *tbb.dll* (trong *release*) vào thư mục chứa file **.exe*. *tbb.dll* (*Thread building block*) là file khá quan trọng, thiếu nó chương trình sẽ báo lỗi.

Sau khi đã hoàn tất việc chép tất cả các header, library và link tới các library tương ứng, ta có thể include các header của opencv vào chương trình và có thể gọi các hàm làm việc của OpenCV.

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace std;
using namespace cv;

void main()
{
    ...
}
```

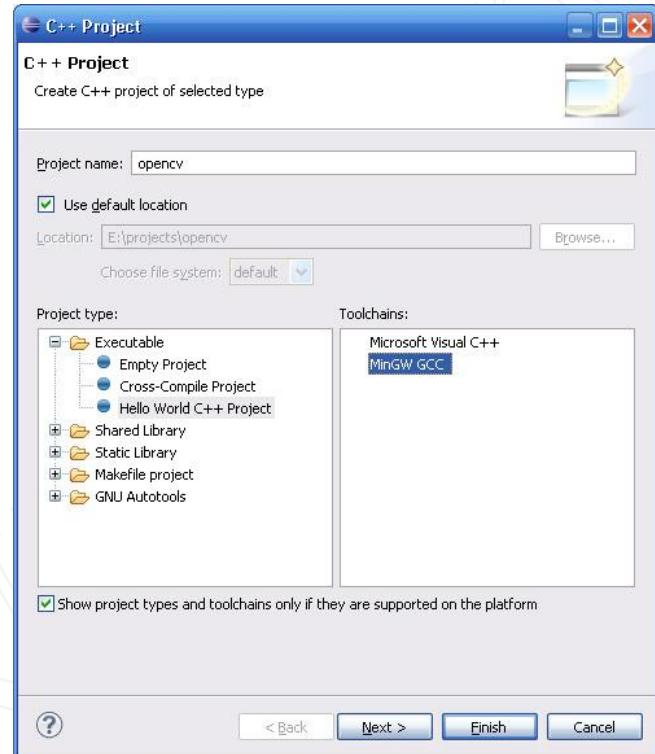
Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Với Eclipse CDT

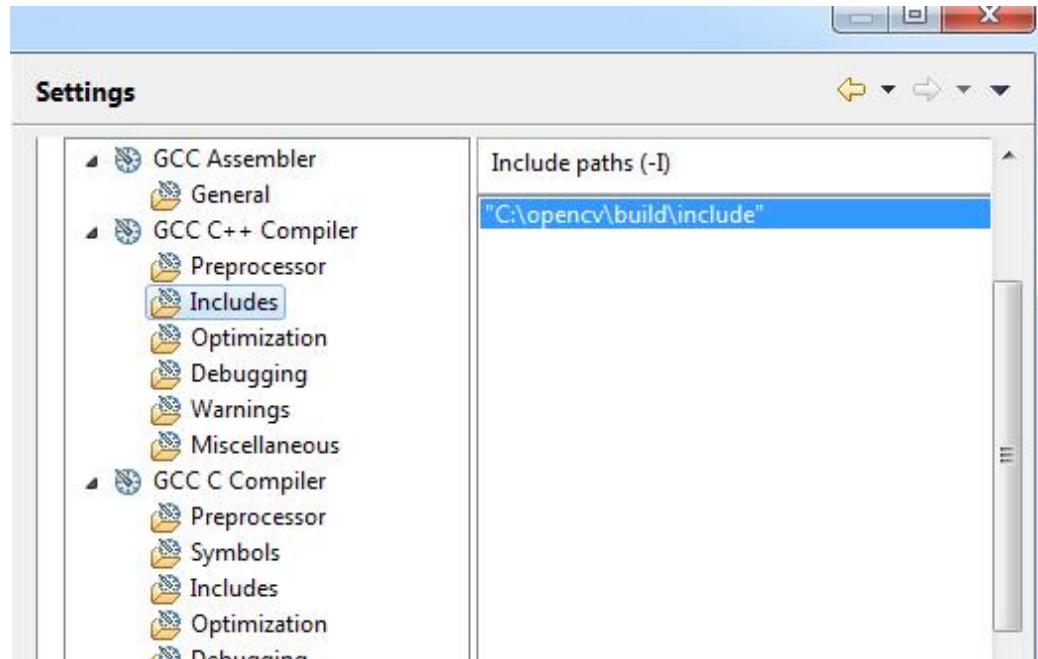
Khi khởi động Eclipse, Tạo mới C++ Project, chọn C++ Project xu thi n, trong hộp thoại Project name là opencv, Project type là Hello World C++ Project (Có thể chọn là Empty Project), Toolchains là MinGW GCC, Chọn Finish và ta có một Project mới. Vậy giờ tùy chỉnh cho project này họ tên là opencv.

Trong cách tạo Project ->Properties, cách Properties hiển thị ra. Trong cách Properties chọn C/C++ Build->Settings. Trong tab Tool Settings.

Phân GCC C++ Compiler chọn Include path để định vị thư mục Include của OpenCV là C:\opencv\build\include. Trong phân MinGW C++ Linker chọn Library và chọn các mục sau: click vào đường Library search path (-L) và đặt tên là mingw\lib và Windows 32 bit hoặc C:\opencv\build\x64\mingw\lib và Windows 64 bit. Tiếp đó click vào mục "cung cấp" thêm Library(-I) vào, các library cần thêm lần lượt là: opencv_core243, opencv_highgui243, opencv_imgproc243 ... nói chung là tùy vào nhu cầu sử dụng có thể thêm một số thư viện khác vào.



Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



Ta cần copy các *.dll sang vào thư mục cùng file chạy *.exe a chéng trình chéng trình có thể chạy thành công.

Chương II. Các phép xử lý hình và ứng dụng cơ bản

1. Chương trình đầu tiên

Trong bài này ta sẽ tìm hiểu một ví dụ về chương trình *Hello world* load và hiển thị ra màn hình. Chương trình như sau:

```
#include "stdafx.h"
#include <iostream>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\core\core.hpp>

using namespace std;
using namespace cv;

int main()
{
    cout<<"Chuong trinh dau tien"<<endl;
    Mat img = imread("vietnam.jpg", CV_LOAD_IMAGE_COLOR);
    namedWindow("Viet Nam", CV_WINDOW_AUTOSIZE);
    imshow("Viet Nam", img);
    waitKey(0);
    return 0;
}
```

Nhà nói trên, trong Opencv với giao diện C++, tất cả các cú pháp đều mới, mà trong đó có câu lệnh `cv::Mat`. Hàm `imread` sẽ nhận vào và trả về biến `img`. Nguyên tắc của hàm này như sau: `cv::Mat imread(const std::string &filename, int flags)` trong đó, `filename` là đường dẫn đến file ảnh, nếu file ảnh không nằm trong thư mục làm việc hiện hành thì ta phải chỉ ra đường dẫn bằng cách như `D:\Anh\vietnam.jpg` hoặc `D://Anh//Vietnam.jpg`. `Flags` là tham số loại hình mà ta muốn load vào, có thể là `load` hoặc `load nhau` (nhưng ta chỉ `CV_LOAD_IMAGE_COLOR`, nếu là nhám xám thì ta `CV_LOAD_IMAGE_GRAYSCALE`...).

Sau khi đã load thành công, muốn hiển thị lên màn hình ta phải trả về cách, hàm `namedWindow(const std::string &winname, int flags)` sẽ trả về cách với tiêu chuẩn là một chuỗi string `winname`. Tham số `flags` sẽ trả về các thuộc tính như sau: `CV_WINDOW_AUTOSIZE` để kích thước cửa sổ tự động theo kích thước của ảnh, `CV_WINDOW_FULLSCREEN` để kích thước cửa sổ khít với màn hình máy tính ...

Cuối cùng, hàm `imshow(const std::string &winname, cv::InputArray Mat)` sẽ hiển thị ra cách đã trả về.

Hàm `waitKey(int delay)` sẽ chờ đến khi có một phím bất kỳ vào trong khoảng thời gian là `delay`. Chú ý là nếu không có hàm này thì chương trình sau khi chạy sẽ không dừng lại màn hình và kết thúc luôn, ta dùng hàm này mục đích là để màn hình lặp

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

trong m t kho ng th i gian b ng tham s *delay* (tính theo n v millisecond). N u mu n d ng màn hình l i mãi mãi ta t tham s *delay* b ng 0.

Và sau ây là k t qu ch ng trình chạy:



2. Không gian màu, chuyển đổi giữa các không gian màu

Không gian màu là m t mô hình toán h c dùng mô t các màu s c trong th c t c bi u di n d i d ng s h c. Trên th c t có r t nh i u không gian màu khác nhau c mô hình s d ng vào nh ng m c ích khác nhau. Trong bài này ta s tìm hi u qua v ba không gian màu c b n hay c nh c t i và ng d ng nhi u, ó là h không gian màu RGB, HSV và CMYK.

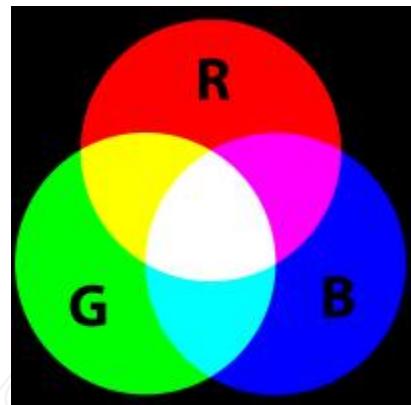
Không gian màu RGB

RGB là không gian màu r t ph bi n c dùng trong h a máy tính và thi t b k thu t s khác. Ý t ng chính c a không gian màu này là s k t h p c a 3 màu s c c b n : màu (R, Red), xanh l c (G, Green) và xanh l (B, Blue) mô t t t c các màu s c khác.

N u nh m t nh s c mã hóa b ng 24bit, ngh a là 8bit cho kênh R, 8bit cho kênh G, 8bit cho kênh B, thì m kênh này màu này s nh n giá tr t 0-255. V i m i giá tr khác nhau c a các kênh màu k t h p v i nhau ta s c m t màu khác nhau, nh v y ta s có t ng c ng $255 \times 255 \times 255 = 1.66$ tri u màu s c.

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Ví dụ: màu en là sự kết hợp của các kênh màu (R, G, B) với giá trị riêng (0, 0, 0) màu trắng có giá trị (255, 255, 255), màu vàng có giá trị (255, 255, 0), màu tím có giá trị (64, 0, 128) ... Nếu ta dùng 16bit mã hóa mỗi kênh màu (48bit cho toàn bộ 3 kênh màu) thì dải màu sẽ trải rộng lên tới $3^*2^{16} = \dots$ Một số con số như sau:



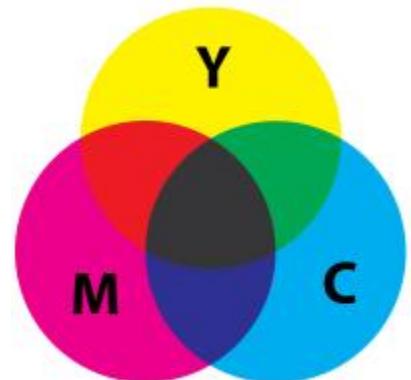
Không gian màu CMYK.

CMYK là không gian màu được sử dụng phổ biến trong ngành công nghiệp in. Ý tưởng cơ bản của nó không gian này là dùng 4 màu sắc cơ bản để pha chế ra màu in. Trên thực tế, người ta dùng 3 màu là C=Cyan: xanh lá cây, M=Magenta: hồng xám, và Y=Yellow: vàng bù đắp các màu sắc khác nhau. Nếu lấy màu hàng xám có giá trị vàng sẽ ra màu đen, màu xanh lá cây và xanh lá cây cho xanh lam ... Sử dụng 3 màu trên sẽ cho ra màu en, tuy nhiên màu en đây không phải là en tuy nhiên có thể pha nâu en, nên trong ngành in, người ta thêm vào màu en in nhạt chiết tạo màu en thay vì phai kỹ thuật 3 màu sắc trên. Và nhay về ta có hình màu CMYK. ch

ây là ký hiệu màu en (Black), có nghĩa là chỉ dùng bù đắp màu Blue nên người ta lấy cái tên K bù đắp màu en?

Nguyên lý làm việc của màu này như sau: Trên một mảng ảnh, khi mỗi màu này được in lên sẽ hòa lẫn nhau thành phần màu trắng. 3 màu C, M, Y khác nhau in theo nhau sẽ khác nhau sẽ hòa lẫn nhau thành phần óm tắt cách khác nhau và cuối cùng cho ra màu sắc chính xác. Khi in màu en, thay vì phai kỹ thuật 3 màu vàng sẽ dùng màu en in lên. Nguyên lý này khác với nguyên lý làm việc của màu RGB ch

RGB là sử dụng các thành phần màu, còn hình CMYK là sử dụng các thành phần màu.



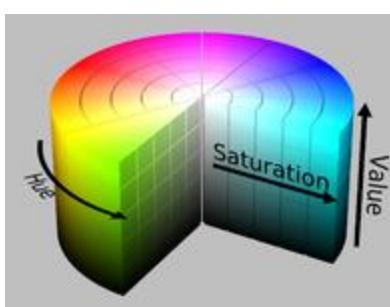
Không gian màu HSV.

HSV và HSL là không gian màu được sử dụng nhiều trong việc chỉnh sửa ảnh, phân tích hình và mô phỏng các giác quan của máy tính. Hình không gian này đưa vào 3 thông số sau mô tả màu sắc: H = Hue: màu sắc, S = Saturation: mức độ hòa, V = value: giá trị cường độ sáng.

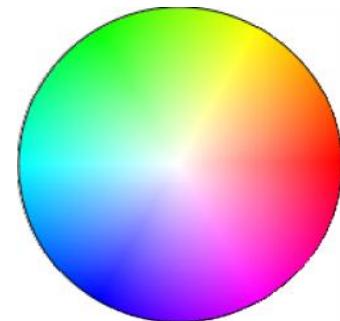
Không gian màu này thường được sử dụng để diễn đạt hình ảnh hoặc hình ảnh. Theo đó,

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

i theo vòng tròn từ 0 - 360 là trang bị biến màu sắc (Hue). Trong này bao gồm màu đỏ tươi (red primary) và màu xanh lá cây tươi (green primary) nằm trong khoảng 0-120, từ 120 - 240 là màu xanh lá cây tím xanh lá (green primary - blue primary). Từ 240 - 360 là tím đậm tím.



Không gian màu HSV



Hình tròn biểu diễn màu sắc (HUE)

Chuyển đổi giữa các không gian màu.

Chuyển đổi RGB sang CMYK và ngược lại.

Nhà đã nói trên, thành phần K là thành phần phông dùng để in cho những hình ảnh có màu đen trong hình CMYK, do vậy chuyển đổi không gian màu từ RGB sang CMYK trước khi ta chuyển RGB sang CMY sau đó tìm thành phần K còn lại. Công thức chuyển đổi RGB sang CMY:

$$(C', M', Y') = ((255 - R), (255 - G), (255 - B)).$$

Vì tính giá trị của K là làm tăng khác vì nó liên quan tới nhà sản xuất công nghệ in, tuy nhiên về mặt lý thuyết có thể chia phần nhạt ròng K = min {C'/255, M'/255, Y'/255}, với 0 <= K <= 100.

Nếu K = 100, thì C = M = Y = 0 (trong hình ảnh màu đen)

Nếu 0 < K < 100: C = (C'/255 - K) * 100 / (100 - K), M = (M'/255 - K) * 100 / (100 - K), Y = (Y'/255 - K) * 100 / (100 - K) và K = K. Trong đó, C, M, Y, K là làm tròn tới 1 chữ số nguyên.

Chuyển đổi RGB sang HSV và ngược lại

Giả sử ta có một hình ảnh màu có giá trị trong hình RGB là (R, G, B). ta chuyển sang không gian HSV như sau:

$$\text{t} M = \text{Max}(R, G, B), m = \text{Min}(R, G, B) \text{ và } C = M - m.$$

Nếu M = R, H' = (G - B) / C mod 6. Nếu M = G, H' = (B - R) / C + 2. Nếu M = B, H' = (R - G) / C + 4. Và H = H' * 60. Trong trường hợp C = 0, H = 0⁰

$$V = M.$$

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

$S = C/V$. Trong trường hợp $V=0$, $S=0$.

Chuyển đổi HSV sang RGB ta làm như sau:

Giả sử ta có không gian màu HSV với $H = [0, 360]$, $S = [0, 1]$, $V = [0, 1]$. Khi đó, ta tính $C = VxS$, $H' = H/60$.

$X = C(1 - |H' \bmod 2 - 1|)$. Ta biểu diễn như sau:

$$(R1, G1, B1) = \begin{cases} (0, 0, 0) & \text{nếu } H \text{ chưa được xác định} \\ (C, X, 0) & \text{nếu } 0 \leq H' < 1 \\ (X, C, 0) & \text{nếu } 1 \leq H' < 2 \\ (0, C, X) & \text{nếu } 2 \leq H' < 3 \\ (0, X, C) & \text{nếu } 3 \leq H' < 4 \\ (X, 0, C) & \text{nếu } 4 \leq H' < 5 \\ (C, 0, X) & \text{nếu } 5 \leq H' < 6 \end{cases}$$

Chương trình chuyển đổi các không gian màu

Trong OpenCV, các không gian màu có thể chuyển đổi qua lệnh hàm cvtColor (convert color), nguyên mực hàm này như sau:

```
cv::cvtColor(cv::InputArray src, cv::OutputArray dst, int code)
```

Trong đó, src , dst là nhúng c và nh thu c sau khi chuyển đổi không gian màu. $code$ là mã chuyển đổi không gian màu. OpenCV có nghĩa là chuyển đổi giữa các không gian màu chung hàn nh $code = CV_BGR2GRAY$ chuyển đổi không gian màu RGB sang nhám, $code = CV_HSV2BGR$ chuyển đổi không gian màu HSV sang không gian màu RGB ...

```
#include "stdafx.h"
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>

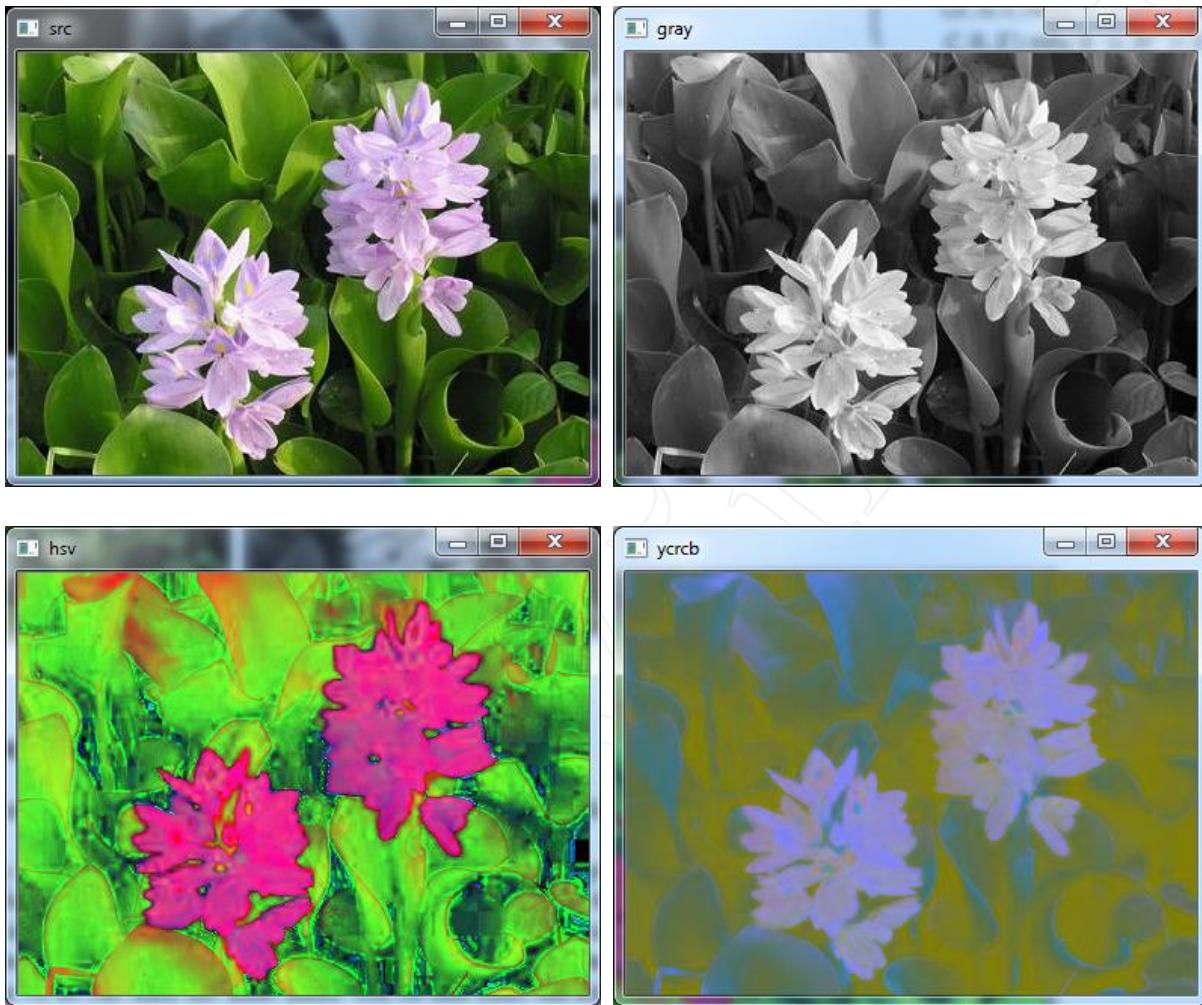
using namespace cv;

void main()
{
    Mat src = imread("LucBinh.jpg", CV_LOAD_IMAGE_COLOR);
    Mat gray, hsv, ycrcb;
    cvtColor(src, gray, CV_BGR2GRAY);
    cvtColor(src, hsv, CV_BGR2HSV);
    cvtColor(src, ycrcb, CV_BGR2YCrCb);
    imshow("src", src);
    imshow("gray", gray);
    imshow("hsv", hsv);
    imshow("ycrcb", ycrcb);
    waitKey(0);
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

}

Sau đây là hình ảnh khi chuyển các khung màu trên



3. Ứng dụng sang và tăng phong nh

Trong bài này ta sẽ tìm hiểu về cấu trúc của một bức ảnh, cách tiếp cận và ứng dụng tinh thông tin.

Một hình ảnh có cấu trúc trên máy tính là một mảng ma trận các điểm ảnh (hay pixel). Trong OpenCV nó được biểu diễn dưới dạng cv::Mat. Ta xét một khung thông tin như thế, nó là một RGB. Vì vậy, mỗi pixel ảnh quan sát có thể là một thành phần màu R (Red), Green (Green) và Blue (Blue). Số thành phần này theo thông tin là R, G, B khác nhau sẽ tạo ra vô số các màu sắc khác nhau. Giả sử như mã hóa bằng 8 bit về từng kênh màu, khi đó giá trị của R, G, B sẽ nằm trong khoảng [0, 255]. Như vậy ta có thể biểu diễn từ $255 \times 255 \times 255 \sim 1.6$ triệu màu sắc ba màu cơ bản trên. Ta có thể xem cách biểu diễn trong OpenCV như dưới đây:

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

| | C t 0 | | | C t 1 | | | C t 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 2 | 2, 0 | 2, 0 | 2, 0 | 2, 1 | 2, 1 | 2, 1 | 2, m | 2, m | 2, m |
| Hàng n | n, 0 | n, 0 | n, 0 | N, 1 | n, 1 | n, 1 | n, m | n, m | n, m |

Nhìn vào hình ảnh có n hàng và m cột, m là chiều dài của nhánh (width) và n là chiều cao của nhánh (height). Mỗi pixel ở vị trí (i,j) trong hình ảnh có 3 kênh màu khác nhau trong nó. Truy xuất tông màu pixel như sau: $\text{pixel}[i][j]$

$\text{img}.at<\text{cv}::\text{Vec3b}>(i,j)[k]$

Trong đó, i, j là pixel hàng thứ i và cột thứ j , img là nhánh mà ta cần truy xuất tất cả các pixel của nó. $\text{cv}::\text{Vec3b}$ là kiểu vector uchar 3 thành phần, dùng để biểu thị 3 kênh màu tông màu. k là kênh màu thứ k , $k = 0, 1, 2$ tương ứng với 3 kênh màu B, G, R. Chú ý là trong OpenCV, hệ màu RGB có biến đổi theo thứ tự chính cái là BGR.

Sau đây ta áp dụng kỹ thuật trên làm tông, giảm sáng và tông phun cảm nhận màu, vì cách làm này có thể hoàn toàn tông tím và nhám, chênh khác biệt là nhau dung một kênh duy nhất biến đổi thành nhám.

Chương trình tông, giảm sáng và tông phun cảm nhận

Giả sử f là một hàm biến đổi cho mỗi tông nào đó, $f(x,y)$ là giá trị của pixel trong hình ảnh (x,y) . $t g(x,y) = f(x,y) + \epsilon$. Khi đó, nếu $\epsilon = 1$, thì ta nói tông $g(x,y)$ có tăng phun gần bằng 1 so với tông $f(x,y)$. Nếu $\epsilon = -1$ ta nói tông $g(x,y)$ đã thay đổi tông phun là . Đầu vào công thức trên ta có chương trình thay đổi sáng và tông phun cảm nhận sau:

```
#include "stdafx.h"
#include <iostream>
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>

using namespace std;
using namespace cv;

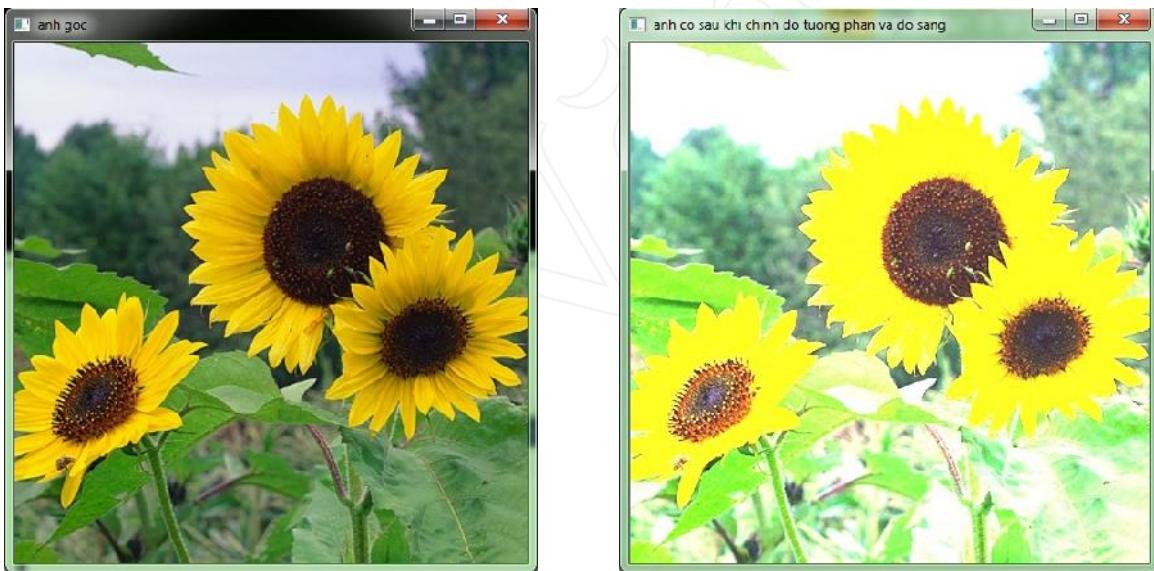
int main()
{
    cout<<"Chương trình điều chỉnh độ sáng và tông phun" << endl;
    Mat src = imread("hoa_huong_duong.jpg", 1);
    Mat dst = src.clone();
    double alpha = 2.0;
    int beta = 30;
    for(int i = 0; i < src.rows; i++)
        for(int j = 0; j < src.cols; j++)
            for(int k = 0; k < 3; k++)
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
        dst.at<Vec3b>(i,j)[k] = saturate_cast<uchar>(
            alpha*(src.at<Vec3b>(i,j)[k] ) + beta);
    imshow("anh goc", src);
    imshow("anh co sau khi chinh do tuong phan va do sang", dst);
    waitKey(0);
    return 0;
}
```

Trong chương trình trên, hàm `clone()` sẽ sao chép một nhánh hình ảnh vào nhánh (`dst = src.clone()`). Giá trị của các pixel $f(x,y)$ và $g(x,y)$ ở đây phải nằm trong khoảng $[0, 255]$, trong khi phép biến đổi $g(x,y) = f(x,y) + \alpha$ có thể khi đó giá trị của $g(x,y)$ vượt quá 255. Để tránh tình trạng tràn số học khi sử dụng không thích, ta dùng thêm hàm `saturate_cast<uchar>(type)`. Hàm này sẽ biến đổi giá trị α không phải `uchar` thành kiểu dữ liệu `uchar`.

Sau đây là kết quả chương trình với giá trị $\alpha = 2.0$ và $\beta = 30$:



4. Nhìn phân, nhộn phân hóa và tăng cường

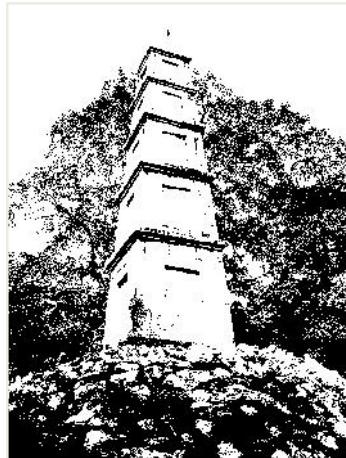
Nhìn phân là nhìn mà giá trị của các điểm ảnh chỉ có hai giá trị 0 hoặc 255 trong vùng với hai màu đen hoặc trắng. Nhộn phân hóa một nhánh là quá trình biến một nhánh xám thành nhìn phân. Giá trị $f(x,y)$ là giá trị của vùng sáng của một điểm ảnh (x,y) , T là ngưỡng nhìn phân. Khi đó, nhánh xám $f(x,y)$ chuyển thành nhìn phân nếu vào công thức $f'(x,y) = 0$ nếu $f(x,y) \leq T$ và $f'(x,y) = 255$ nếu $f(x,y) > T$.

Hình sau mô tả một nhìn phân và nhộn phân $T = 100$.

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



nh xám



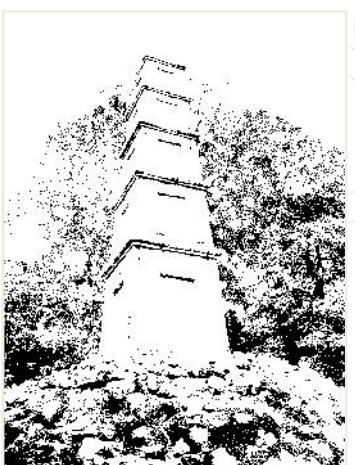
nh nh phân

Hàm chuyển nh phân hóa nh trong OpenCV là hàm *threshold()*. Nguyên m u hàm nh sau:

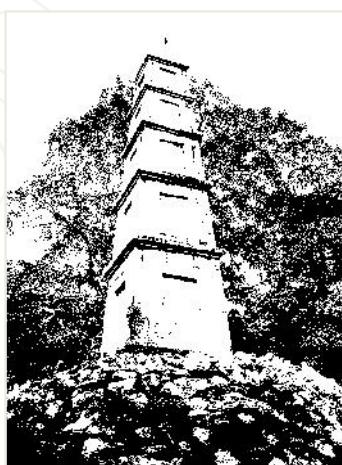
threshold(cv::InputArray src, cv::OutputArray dst, double thresh, int maxval, int type)

Trong ó, *src* là nh u vào m t kên màu (nh xám ...), *dst* là nh sau khi c nh phân hóa, *thresh* là ng ng nh phân, *maxval* là giá tr l n nh t trong nh (*maxval* = 255 i v i nh xám), *type* là ki u nh phân có th là CV_THRESH_BINARY, CV_THRESH_BINARY_INV, CV_THRESH_OTSU... l n l t là nh phân hóa thông th ng, nh phân hóa ng c và nh phân hóa theo thu t toán Otsu ...

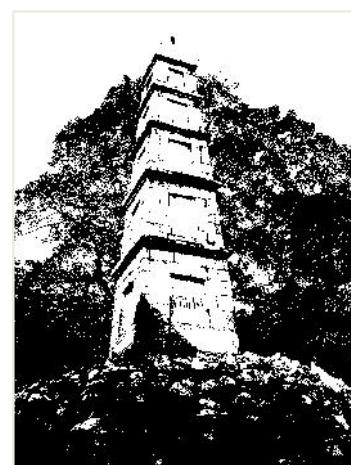
K t qu c a vi c nh phân hóa m t nh ph thu c vào ng ng T, có ngh a là v i m i ng ng T khác nhau thì ta có nh ng nh nh phân khác nhau. Hình sau mô t 3 nh nh phân t ng ng v i ng ng T = 50, T = 100 và T = 150.



T = 50



T = 100



T = 150

thu c m t nh nh phân t t mà không c n ph i quan tâm t i các i u ki n ánh sáng khác nhau (không c n quan tâm t i ng ng T), ng i ta dùng m t k thu t sao cho v i m i ng ng xám khác nhau ta luôn thu c m t nh nh phân t t. K thu t ó g i là k

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

thu t nh phân hóa v i ng ng (Dynamic threshold) hay nh phân thích nghi (Adaptive threshold)

Có nhi u ph ng pháp khác nhau th c hi n vi c này, tuy nhiên chúng u d a trên ý t ng chính là chia nh ra thành nh ng vùng nh , v i m i vùng áp d ng vi c nh phân cho vùng ó v i nh ng ng nh phân khác nhau.Các ng ng nh phân các vùng c tính toán d a trên 1 n m c xám c a chính các pixel trên vùng ó. Gi s ta tính toán ng ng cho m t vùng nào ó d a trên trung bình c a các pixel trong vùng ó (ta có th xem m t vùng là m t c a s). Ta xét quá trình nh phân v i ng ng trong m t vùng c a s 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 nh nh phân thu c trên là vùng nh c nh phân v i ng ng là trung bình c ng c a t t c các ô trong c a s $T = (55 + 10 + 100 + \dots)/25 = 65.6$.

Ch ng trình nh phân hóa v i ng ng nh sau

```
// Adaptive Threshold
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>

#include <iostream>

using namespace std;
using namespace cv;

int main()
{
    cout<<"Nhi phan anh voi nguong dong"<<endl;
    Mat src = imread("Thap_But.jpg", CV_LOAD_IMAGE_GRAYSCALE);
    Mat dst;
    adaptiveThreshold(src, dst, 255, CV_ADAPTIVE_THRESH_MEAN_C,
                      CV_THRESH_BINARY, 35, 5);
    imshow("Anh xam goc", src);
    imshow("Anh nhi phan voi nguong dong", dst);
    waitKey(0);
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
    return 1;  
}
```

Trong chương trình trên, hàm `threshold` hiển thị kết quả phân hóa với là hàm `adaptiveThreshold`, Nguyên mực của hàm như sau:

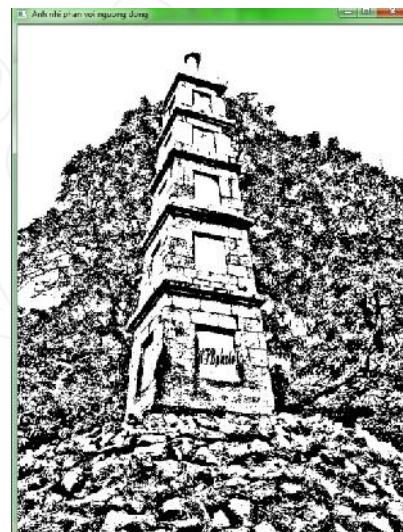
```
cv::adaptiveThreshold(cv::InputArray src, OutputArray dst, double maxValue,  
int adaptiveMethod, int thresholdType, int blockSize, double C)
```

Trong đó, `src` là nháy mực nhâp phán, `dst` là nháy kí thuât, `maxValue` là giá trị lớn nhất trong nháy mực (thông thường là 255), `adaptiveMethod` là cách thách thức phân vùng, `thresholdType` là cách tính giá trị nhâp phán trong tảng vùng cản nhâp phán, `blockSize` là kích thước các ô áp dụng cho việc tính toán nhâp phán, và `C` là môt thông số bù tr trong tảng hophil có tinh phan quá lỏng.

Kết quả khi chạy chương trình như sau:



nháy mực

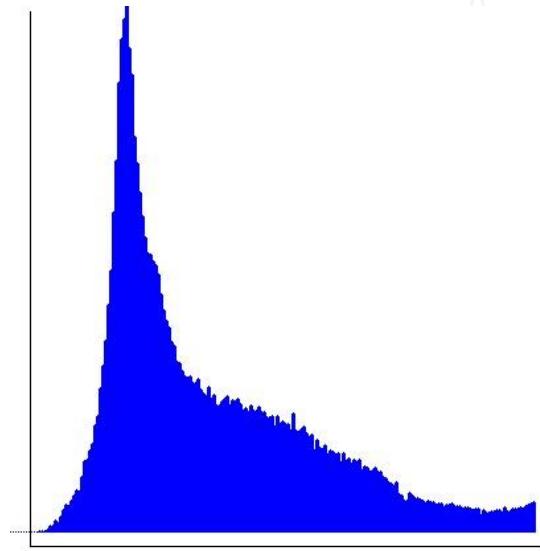


nháy nhâp phán

5. Histogram, cân bằng histogram trong nh

Histogram cát mảnh là một biểu đồ nói lên mối quan hệ giữa các giá trị của pixel ảnh (tín hiệu) và tần suất xuất hiện của chúng. Nhìn vào biểu đồ histogram ta có thể đoán được tính chất ánh sáng của hình ảnh nào.

Nếu một ảnh có histogram lệch về phía bên phải, ta nói nó có ánh sáng sáng. Nếu lệch về phía trái thì nó có ánh sáng xám. Hình bên mô tả histogram cát mảnh xám, nó này có histogram lệch về phía trái của biểu đồ và do đó nó này là khá tối. Ở đây có màu, ta có thể tính toán histogram cho từng kênh màu một. Sau đây là cách để trình tính và vẽ biểu đồ histogram cát mảnh màu.



```
#include <iostream>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

using namespace std;
using namespace cv;

int main()
{
    std::cout<<"Tạo histogram ảnh màu"<<std::endl;

    Mat src = imread("buoi.jpg");
    vector<Mat> img_rgb;
    Mat img_r, img_g, img_b;
    int w = 400, h = 400;
    int size_hist = 255;
    float range[] = {0, 255};
    const float* hist_range = {range};

    split(src, img_rgb);
    calcHist(&img_rgb[0], 1, 0, Mat(), img_b, 1, &size_hist, &hist_range, true, false);
    calcHist(&img_rgb[1], 1, 0, Mat(), img_g, 1, &size_hist, &hist_range, true, false);
    calcHist(&img_rgb[2], 1, 0, Mat(), img_r, 1, &size_hist, &hist_range, true, false);

    int bin = cvRound((double)w/size_hist);

    Mat disp_r(w, h, CV_8UC3, Scalar( 255,255,255 ) );
    Mat disp_g = disp_r.clone();
    Mat disp_b = disp_r.clone();

    normalize(img_b, img_r, 0, disp_b.rows, NORM_MINMAX, -1, Mat() );
    normalize(img_g, img_g, 0, disp_g.rows, NORM_MINMAX, -1, Mat() );
    normalize(img_r, img_b, 0, disp_r.rows, NORM_MINMAX, -1, Mat() );
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
for( int i = 1; i < 255; i++ )
{
    line(disp_r, Point(bin*(i), h), Point(bin*(i), h - cvRound(img_r.at<float>(i))),  

          Scalar(0, 0, 255), 2, 8, 0 );
    line(disp_g, Point(bin*(i), h), Point(bin*(i), h - cvRound(img_g.at<float>(i))),  

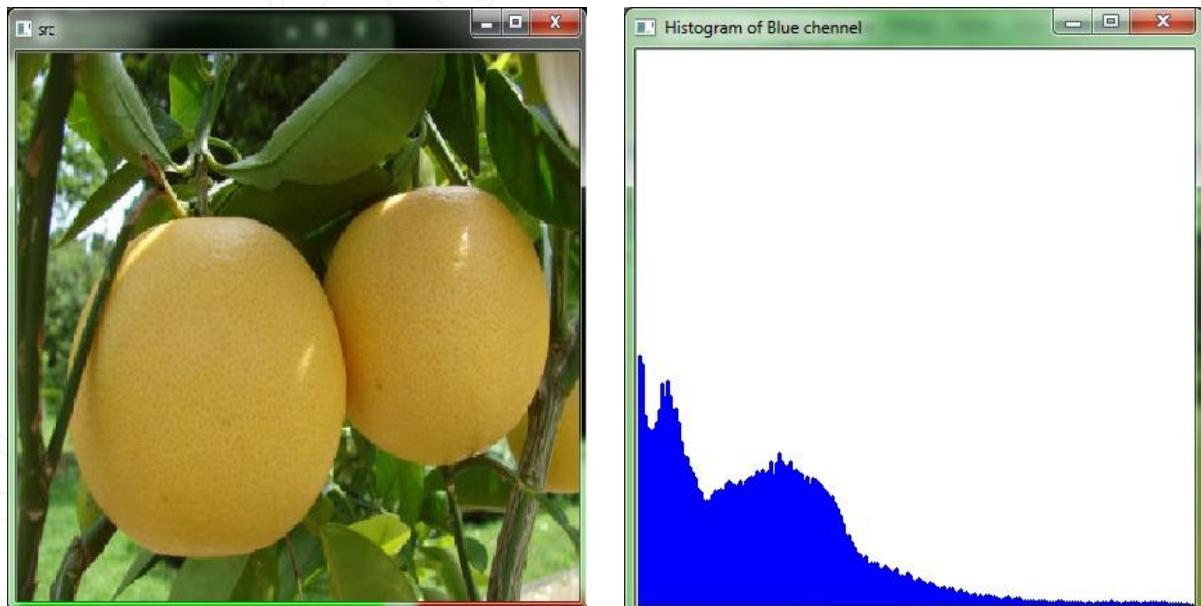
          Scalar( 0, 255, 0 ), 2, 8, 0 );
    line(disp_b, Point( bin*(i), h), Point(bin*(i), h - cvRound(img_b.at<float>(i))),  

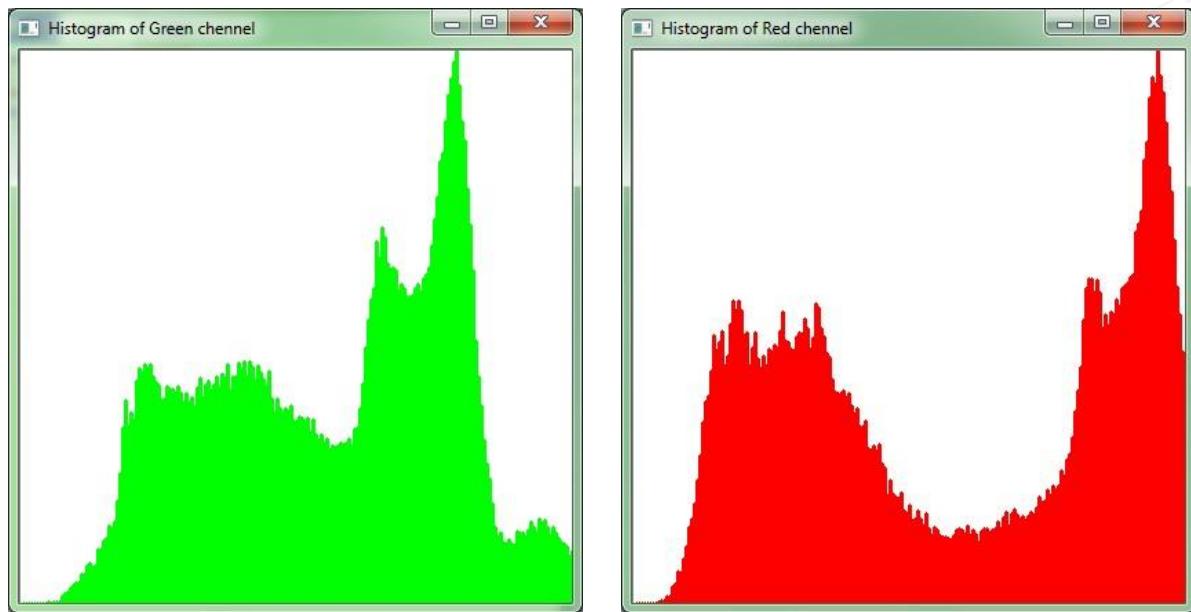
          Scalar(255, 0, 0), 2, 8, 0 );
}
namedWindow("src", 0);
imshow("src", src);
imshow("Histogram of Blue chennel", disp_b);
imshow("Histogram of Green chennel", disp_g);
imshow("Histogram of Red chennel", disp_r);

cv::waitKey(0);
return 1;
}
```

Trong chương trình trên, đầu tiên nhập ảnh vào là một hình ảnh, tính histogram của tám kênh màu mutiatsu phân tách nh này thành 3 kênh riêng rẽ theo thứ tự là Blue, Green và Red. Hàm `cv::split(const cv::Mat src, cv::Mat *mvbegin)` sẽ tách thành `src` thành các kênh màu tám ngang lùi trong `mvbegin`.

Hàm `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)` sẽ tính histogram của nhữn vào `images` và lùi kí quát tính toán vào mảng `hist`. Các thông số khác bao gồm: `nimages` là số lượng ảnh vào, `channels` là danh sách chỉ các kênh dùng để tính histogram, `mask` là một mảng tùy chỉnh, nếu không dung gì thì là `cv::Mat().` `dims` là chỉ số của histogram, bốn OpenCV hiện tại chỉ tính toán histogram với chỉ số lên tới 32. `histSize` là kích thước của histogram mỗi chiều, hai tham số cuối có thể thay đổi, sau đây là kết quả chương trình.





i v i nh xám, ta có th xem nh nó là m t kênh màu, do v y tính histogram c a nh xám ta c ng có th làm hoàn toàn t ng t b ng cách g i hàm *calcHist()*

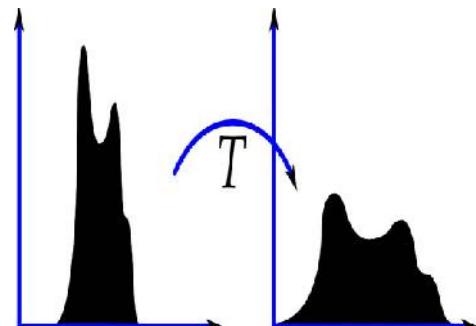
Cân b ng histogram

Cân b ng histogram (histogram equalization) là ph ng pháp làm cho bi u histogram c a nh c phân b m t cách ng u. ây là m t bi n i khá quan tr ng giáp nâng cao ch t l ng nh, thông th ng ây là b c ti n x lý c a m t nh u vào cho các b c ti p theo.

cân b ng histogram, ta dung hàm *equalizeHist(cv::InputArray src, cv::OutputArray dst)* trong ó, *src* là nh u vào m t kênh màu (nh xám ch n h n), *dst* là nh sau khi cân b ng. Ví d :

```
Mat src = imread("src.jpeg", CV_LOAD_IMAGE_GRAYSCALE); // Load anh xam  
imshow("Anh xam goc", src);  
Mat dst;  
equalizeHist(src, dst);  
imshow("anh xam sau khi can bang histogram", dst);
```

Ta có k t qu sau:



Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



cân bằng histogram là một kỹ thuật chuyển đổi màu RGB sang HSV, sau đó cân bằng thành phần kênh màu V (Value - độ sáng) và binning cell. Chương trình sau cân bằng histogram là như sau:

```
// Cân bằng histogram

#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>

#include <iostream>

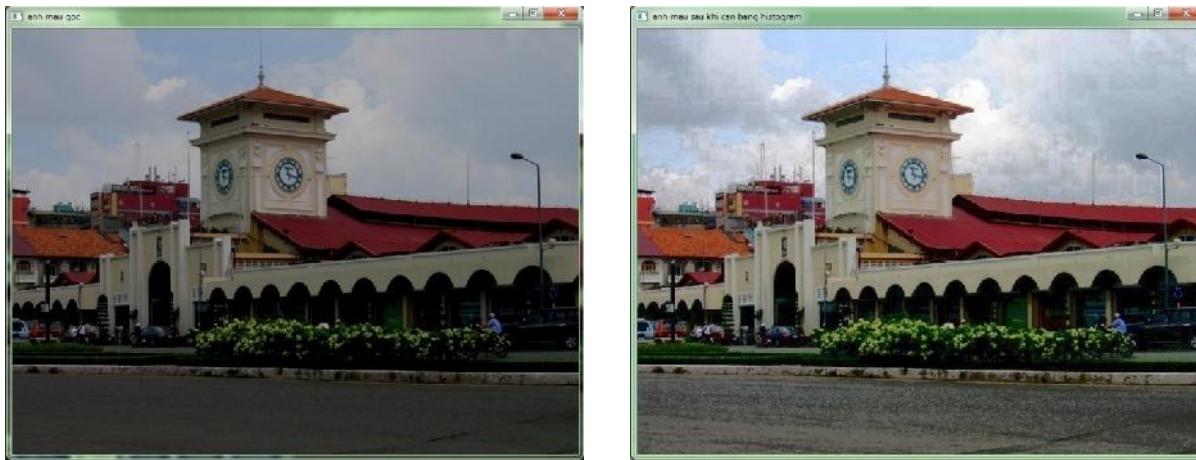
using namespace std;
using namespace cv;

int main()
{
    cout<<"Chương trình cân bằng histogram"<<endl;
    Mat src = imread("Cho_Ben_Thanh.jpg", CV_LOAD_IMAGE_COLOR);

    Mat hsv, disp;
    cvtColor(src, hsv, CV_BGR2HSV);
    vector<Mat> hsv_channels;
    // Tách hsv thành 3 kênh màu
    split(hsv, hsv_channels);
    // Cân bằng histogram kênh màu v (value)
    equalizeHist(hsv_channels[2], hsv_channels[2]);
    // Trộn ảnh
    merge(hsv_channels, hsv);
    // Chuyển đổi hsv sang rgb để hiển thị
    cvtColor(hsv, disp, CV_HSV2BGR);
    imshow("anh mau goc", src);
    imshow("anh mau sau khi can bang histogram", disp);
    waitKey(0);

}
```

Sau đây là kết quả cách chương trình trên



6. Phóng to, thu nhỏ và xoay ảnh

Những điều nói trên, những thao tác chia sẻ là một trong các tính năng, do đó có thể phóng to, thu nhỏ hay xoay một cách mà không cần các thuật toán tốn thời gian.

Tất cả đều sử dụng biến affine để quay và thay đổi hình ảnh, như các ma trận.

Bí quyết affine:

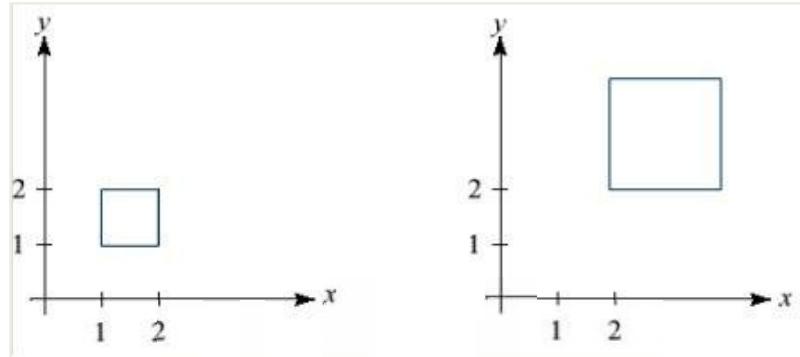
Giả sử ta có vector $p = [x, y]^T$ và ma trận M 2×2 . Phép biến đổi affine trong không gian hai chiều có thể xác định nghĩa $p' = Mp$ trong đó $p' = [x', y']^T$. Viết một cách tóm tắt minh ta có:

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

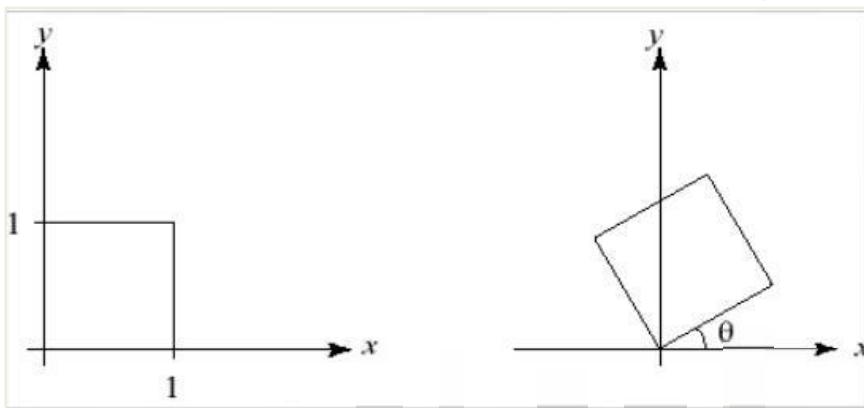
Hay $x' = \alpha x + \delta y, y' = \gamma x + \beta y$.

Xét ma trận $\begin{bmatrix} \alpha & \delta \\ \gamma & \beta \end{bmatrix}$. Nếu $\delta = \gamma = 0$, khi đó $x' = x$ và $y' = y$, phép biến đổi này làm thay đổi chỉ số ma trận. Nếu là trong trường hợp thu nhỏ. Hình sau mô tả phép biến đổi $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



Nếu ta nh ngha ma tr n $M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$ thi phép biến s quay p thành p' v i góc quay là .



Nếu ma tr n M c nh ngha thành $M = \begin{bmatrix} \alpha \cdot \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \beta \cdot \cos(\theta) \end{bmatrix}$ thi phép biến i s v a là phép biến i theo t 1 và quay.

Bây gi ta s xét ch ng trình phóng to, thu nh và quay anh.

```
// Chuong trinh phong to, thu nho va quay anh

#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>

using namespace cv;

int main()
{
    Mat src = imread("HoaSen.jpg");
    Mat dst = src.clone();

    double angle = 45.0;
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
double scale = 1.5;
Point2f center(src.cols/2, src.rows/2);

Mat mat_rot = getRotationMatrix2D(center, angle, scale);
warpAffine(src, dst, mat_rot, src.size());
imshow("Anh goc", src);
imshow("Anh sau phep bien doi", dst);
waitKey(0);

return 1;
}
```

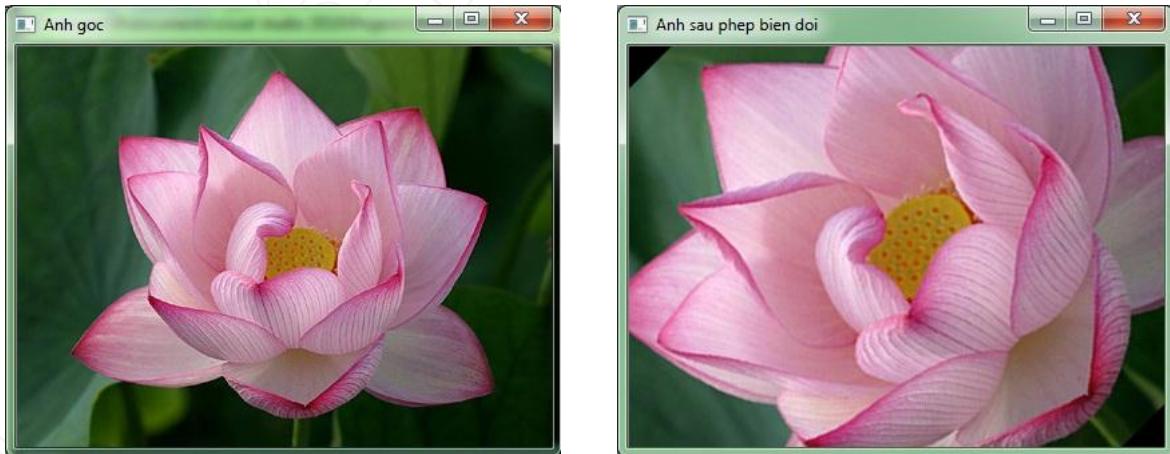
Trong chương trình trên, hàm `cv::getRotationMatrix2D(cv::Point center, double angle, double scale)` sẽ tạo ra ma trận với tâm quay `center`, góc quay `angle` và tỉ lệ `scale`. Ma trận này được tính toán trong Opencv là ma trận như sau:

$$M = \begin{bmatrix} \alpha & \beta & (1 - \alpha).center.x - \beta.center.y \\ -\beta & \alpha & \beta.center.x - (1 - \alpha).center.y \end{bmatrix}$$

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

Ta thấy rằng ma trận này là hoàn toàn tách rời với ma trận của phép biến đổi affine. Ý nói là trên, ngoài việc thành phần thứ 3 là thành phần giúp dịch chuyển tâm quay vào chính giữa các bước. Chú ý là có sự khác biệt một chút về cách tính trong nhau, hai trong nhau là góc trên bên trái làm gốc là (0,0) còn hai trong ta hay lấy điểm trung tâm bên trái làm gốc, do đó có sự khác nhau.

Kết quả cách chương trình trên với `scale = 1.5` và `góc quay = 45°` như hình sau



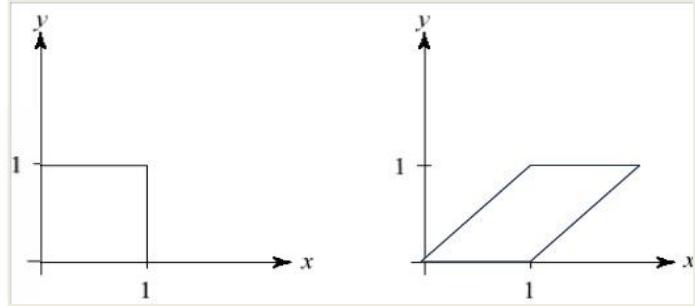
Ngoài hai phép biến đổi là tách rời quay và缩放, ta có thể thêm các biến đổi khác của phép biến đổi affine như phép trượt (shearing), hoặc phép phản chiếu (reflection) bằng

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

vì c^o nh ngh a lⁱ i ma trⁿ M. Ta th^u nh ngh a lⁱ i ma trⁿ M c^m t^c nh tr^t c^a ảnh gốc. Quay lại ma trận M nh^u trên, n^u ta^u nh ngh a⁼ = 1 c^on^g nh^u n^u m^u t^c giá tr^bất kⁱ, khi đó ta s^ec^o:

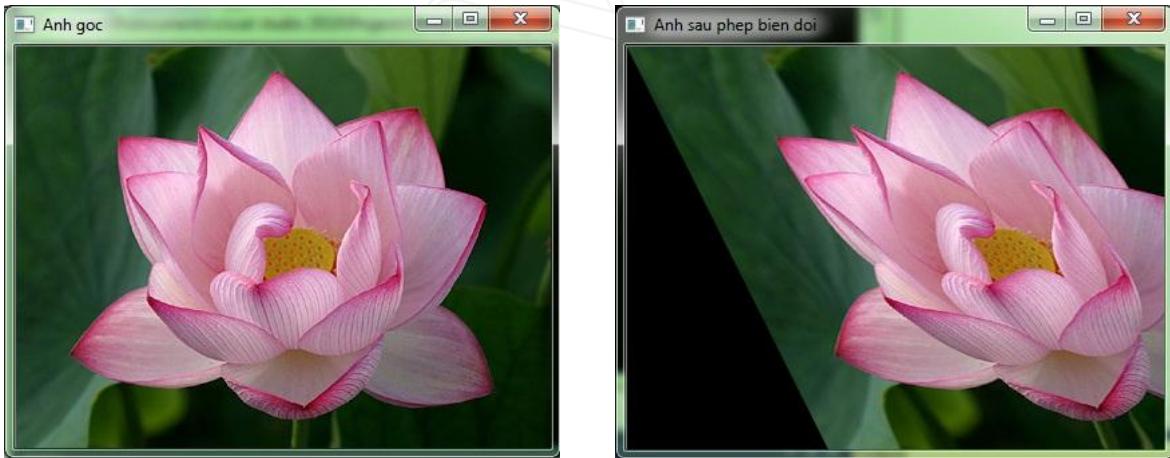
$$\begin{cases} x' = x + \delta y \\ y' = y \end{cases}$$
 ta s^ec^o định ngh a ma
trⁿ M = $\begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ đ^e tr^t t^c nh
ban^u thành^u nh m^u i v['] i h['] s['] tr^röt^r
= 0.5, ch^u ý là thành phⁿ thⁿ ba $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$
nh ngh a ma trⁿ trong Opencv s[']

th^u hiⁿ d^d d^d ch^u chuyⁿ, gi^{ng} nh^u trong ví^d trên ta chuyⁿ tâm quay v^u tâm c^a b^c
nh ch^u ng^h n^u. Ta s^e làm gi^{ng} h^u t^c ví^d trên, ch^u thay ma trⁿ M là ma trⁿ ta t^c nh
ngh a



```
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());
```

và ta có k^t qu^u :



7. L^cs^s trongⁿh

L^cs^s trongⁿh có ý ngh a quan tr^u ng trong vi^c t^c o ra các hi^u ng trongⁿh, m^t s^s
hi^u ng^h s^s d^d ng^h các b^l c^{nh} làm^m nh(Blur), làm^t rⁿ nh(Smooth) ...
Nguyên^t c^c chung c^a ph^u ng^h pháp^l c^{nh} là cho^u nh^u nhân ch^p vⁱ m^t ma trⁿ l^c,

$$I_{dst} = M^* I_{src}$$

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

I_{src} , I_{dst} là nh g c và nh sau khi th c hi n phép l c nh b ng cách nhân v i ma tr n l c M . Ma tr n M ôi khi còn g i là m t n (mask), nhân (kernel). V i m i phép l c ta có nh ng ma tr n l c M khác nhau, không có quy nh c th nào cho vi c xác nh M , tuy nhiên ma tr n này có m t s c i m nh sau:

- Kích th c c a ma tr n th ng là m t s 1 ch ng h n 3x3, 5x5 ... Khi ó, tâm c a ma tr n s n m giao c a hai ng chéo và là i m áp t l ên nh mà ta c n tính nhân ch p.
- T ng các ph n t trong ma tr n thông th ng b ng 1. N u t ng này l n h n 1, nh qua phép l c s có sáng l n h n nh ban u. Ng c l i nh thu c s t i h n nh ban u.

Ví d v ma tr n l c (ma tr.n1.c Sobel theo h ng x, y và ma tr n Gausian Blur)

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

Công th ic cụ thể cho việc lọc ảnh như sau:

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

Trong ó, ta ang tính phép nhân ch p cho i m nh có t a (x,y) và vì ta l y tâm c a ma tr n l c là i m g c n ên u ch y t $-n$ (i m bên trái) và v ch y t $-n$ (i m phía trên) n n, v i n = (kích th c m t n - 1)/2. d hi u h n ta xét m t v í d v vi c làm tr n nh s d ng m t ma tr n l c nh sau:

$$\begin{array}{|c|c|c|} \hline 1/9 & 1/9 & 1/9 \\ \hline \end{array} * \begin{array}{|c|c|c|c|c|} \hline 100 & 100 & 100 & 100 & 100 \\ \hline 100 & 200 & 205 & 203 & 100 \\ \hline 100 & 195 & 200 & 200 & 100 \\ \hline 100 & 200 & 205 & 195 & 100 \\ \hline 100 & 100 & 100 & 100 & 100 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|} \hline 100 & 100 & 100 & 100 & 100 \\ \hline 100 & 144 & 205 & 203 & 100 \\ \hline 100 & 195 & 200 & 200 & 100 \\ \hline 100 & 200 & 205 & 195 & 100 \\ \hline 100 & 100 & 100 & 100 & 100 \\ \hline \end{array}$$

nhân ch p
pixel (2,2)

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

$$\begin{array}{|c|c|c|} \hline 1/9 & 1/9 & 1/9 \\ \hline \end{array} * \begin{array}{|c|c|c|c|c|} \hline 100 & 100 & 100 & 100 & 100 \\ \hline 100 & 200 & 205 & 203 & 100 \\ \hline 100 & 195 & 200 & 200 & 100 \\ \hline 100 & 200 & 205 & 195 & 100 \\ \hline 100 & 100 & 100 & 100 & 100 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|} \hline 100 & 100 & 100 & 100 & 100 \\ \hline 100 & 144 & 167 & 203 & 100 \\ \hline 100 & 195 & 200 & 200 & 100 \\ \hline 100 & 200 & 205 & 195 & 100 \\ \hline 100 & 100 & 100 & 100 & 100 \\ \hline \end{array}$$

nhân chập
pixel (2,3)

.....

Và k t qu cu i cùng ta có:

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

nhân chập

Ta thấy rõ, nh ban u v i là nh có t ng ph n khá l n (các giá tr 1 n pixel chênh l ch l n: 100, 200, ...), sau phép l c nh có t ng ph n gi n i hay b làm m i(lúc này chênh l ch giá tr gi a các pixel gi m i: 100, 144, 167 ...). Sau ây ta s xem xét m t s b 1 c trong OpenCV.

L c Blur:

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Matrice lọc ảnh này có dạng: $M = \frac{1}{rows*cols} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}$

Bước này có tác động làm trung bình, không nhiễu hình.

Hàm cài đặt trong OpenCV có dạng:

`cv::blur(const Mat& src, const Mat& dst, Size ksize, Point anchor, int borderType)`
trong đó, `src` và `dst` là nhúng ảnh và nhau phép lắc. `ksize` là kích thước ma trận lắc, `ksize = Size(rows, cols)`, `anchor` là tâm neo của ma trận lắc, nếu `anchor` là `(-1, -1)` thì tâm này chính là tâm của ma trận. `borderType` là phương pháp lắc và cách nhau các ảnh nhau qua phép lắc chúng bao gồm tra khung giới hạn của ảnh. Thông thường giá trị mặc định của nó là 4.

nh qua phép lắc blur



Lắc Sobel:

Lắc sobel chính là cách tính xấp xỉ cho hàm bắc nhớt theo hướng x và y, nó cũng chính là cách tính gradient trong ảnh. Bước này thông thường áp dụng cho mục đích tìm biên trong ảnh. Ma trận lọc theo các hướng x, y lần lượt 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 OpenCV, hàm cài đặt phép lắc này như sau:

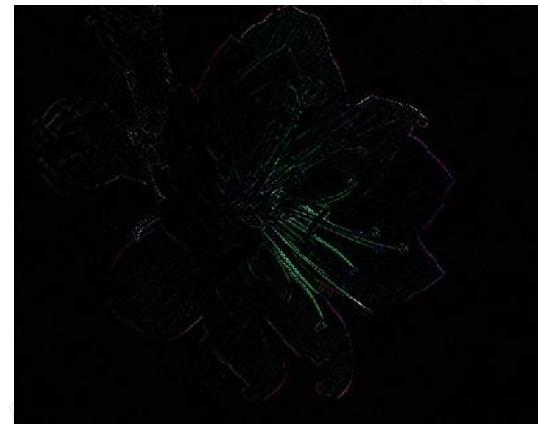
`cv::Sobel(const Mat& src, Mat& dst, int ddepth, int xorder, int yorder, int ksize, double scale, double delta, int borderType)`

Trong đó, `src` và `dst` là nhúng ảnh và nhau qua phép lắc. `ddepth` là sâu của ảnh sau phép lắc, có thể là `CV_32F`, `CV_64F`... . `xorder` và `yorder` là các hàm theo

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

hình ảnh x và y, tính toán hàm theo hình nào ta sẽ giá trị lên 1, nếu `cv::Sobel` giá trị bằng 0, hàm cài đặt sẽ qua không tính theo hình ó. `Scale` và `delta` là hai thông số tùy chỉnh cho việc tính giá trị của hàm 1 là giá trị sai vào nhau phép lắc, chúng có giá trị mặc định là 1 và 0. `borderType` là tham số nhau trên.

nh qua phép lắc Sobel:



Lắc Laplace:

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

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

Matrice này có định nghĩa sau:

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

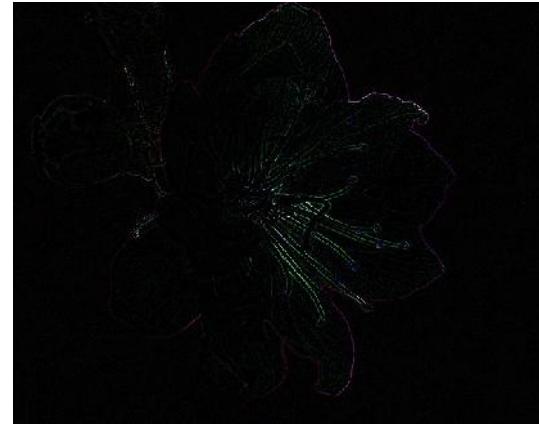
Trong OpenCV, bộ lọc này được cài đặt qua hàm:

```
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 gì như các thông số trong bộ lọc Sobel, chia khác chỉ `ksize` là một giá trị integer có nhau 1 và khi đó ma trận lọc laplace trên sẽ áp dụng.

nh qua phép lắc Laplace:

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



Ngoài 3 b 1 c trên, OpenCV còn cài t khá nhi u b 1 c khác nh l c trung v (*medianBlur*), 1 c Gause (*gaussianBlur*), *pyrDown*, *pyrUp* ... Tuy nhiên, ta hoàn toàn có th cài t b 1 c cho riêng mình thông qua hàm *cv::filter2D*. Nguyên m u hàm này nh sau:

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

Trong ó, *src* và *dst* là nh g c và nh thu c qua phép l c, *kernel* là ma trán l c. Thông s *anchor* ch ra tâm c a ma tr n, *delta* i u ch nh sáng c a nh sau phép l c (nh sau phép l c c c ng v i *delta* và *borderType* là ki u xác nh nh ng pixel n m ngoài vùng nh.

Hàm *cv::filter2D* th c ch t là hàm tính toán nhân ch p gi a nh g c và ma tr n l c cho ra nh cu i sau phép l c. Nh v y qua trên ta th y r ng ti n hành vi c l c nh ta ch c n nh ngh a m t ma tr n l c *kernel*. Có r t nhi u nghiên c u v toán h c ã nh ngh a các ma tr n này, do ó vi c áp d ng vào l c nh s là khá n gi n.

Ch ng trình demo ph ng pháp l c nh.

Ch ng trình sau s th c hi n một số b 1 c nh do chính ta nh ngh a, ch ng h n ta có ma tr n l c $M = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$, ma tr n này là ma tr n s làm n i b t nh gi ng nh nh kh c 3D, ta s áp d ng ma tr n này l c m t nh.

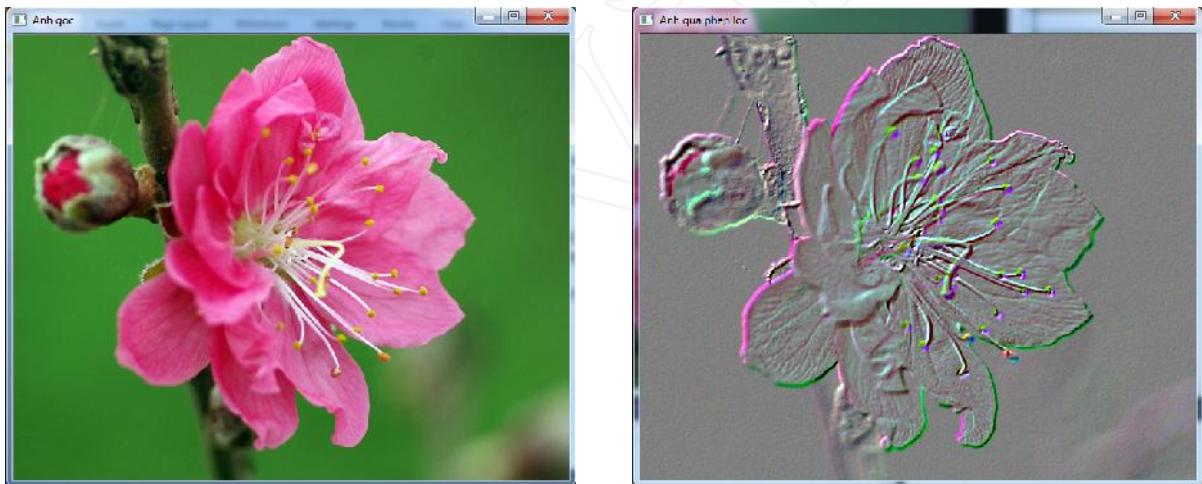
```
#include "stdafx.h"
#include <opencv2\core\core.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <opencv2\highgui\highgui.hpp>
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
using namespace cv;

void main()
{
    Mat src = imread("HoaDao.jpg", CV_LOAD_IMAGE_COLOR);
    Mat dst;
    double m[3][3] = { -1, -1, 0,
                      -1, 0, 1,
                      0, 1, 1
                    };
    Mat M = cv::Mat(3,3,CV_64F, m, Point(-1,-1), 128.0);
    cv::filter2D(src, dst, src.depth(), M);
    imshow("Anh goc", src);
    imshow("Anh qua phap loc", dst);
    cv::waitKey(0);
}
```

Kết quả cách nhau:

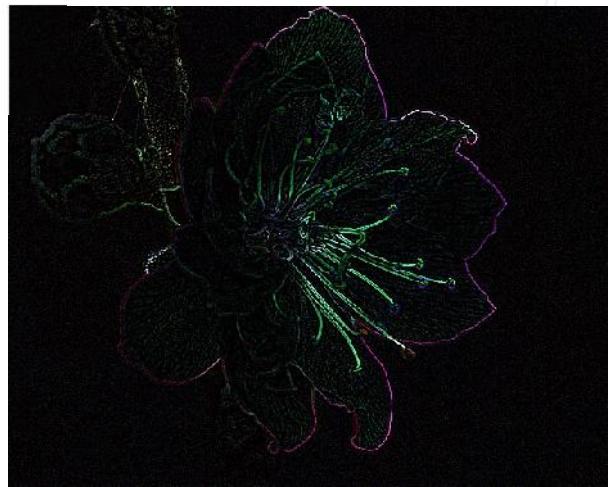


Một số bước sau có thể hữu ích cho việc tạo ra các hình ảnh mới:

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -7 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \text{delta} = 0$$



$$M = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, delta = 0$$



$$M = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, delta = 0$$



8. Các phép toán hình thái học trong ảnh

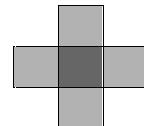
Các phép toán hình thái học là những phép toán liên quan tới cấu trúc hình học (hay topo) của các đối tượng trong ảnh. Các phép toán hình thái học tiêu biểu bao gồm phép giãn nở (dilation), phép co (erosion), phép mở (opening) và phép đóng (closing).

Phép toán giãn nở (dilation)

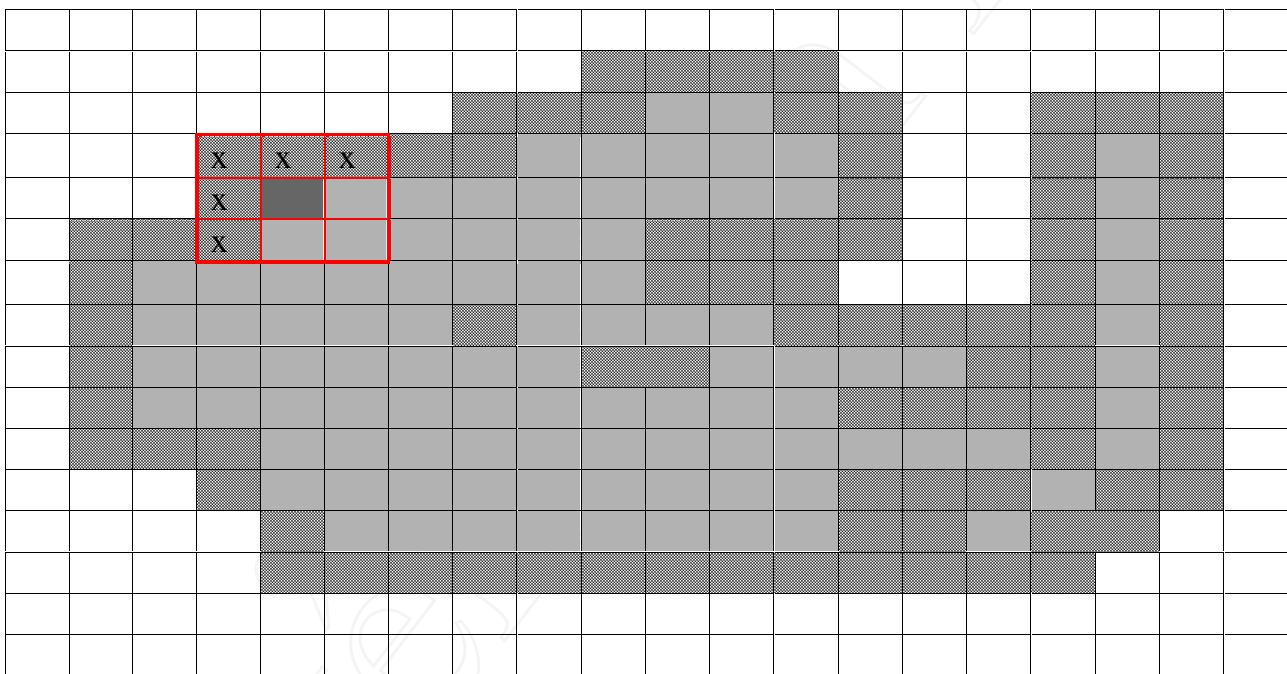
Phép toán giãn nở có nghĩa là $A \oplus B$ với $x \in A$ trong đó, A là đối tượng, B là mảnh cấu trúc phẳng ảnh. Phép toán này có tác dụng làm cho đối tượng ban đầu trong ảnh tăng lên về kích thước (giảm nồng độ).

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Cuối trúc phím tinh (image structuring element) là một hình khung có nhau sảnh mảng tác với nhau xem nó có thể a mãn mảng tính chất nào ó không, mảng sảnh cuối trúc phím tinh hay giao phím là cuối trúc theo khung hình vuông và hình chéo p



Ta hãy xét một ảnh vội iết ng trong nh ảnh biến đổi bằng màu nâu, sau ó dùng cuối trúc phím tinh hình vuông (màu nâu) làm giao nhau, kề qua là nh ảnh giao nhau và phím giao nhau ta ảnh dưới là ảnh x.



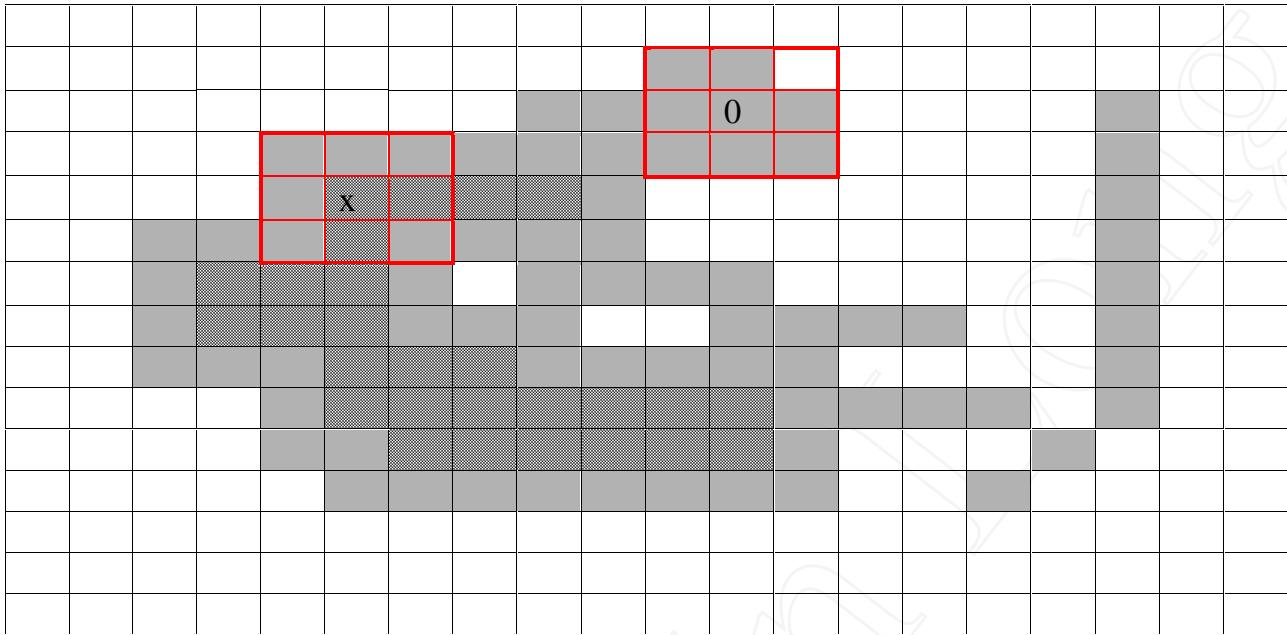
ng đồng cùa phép giãn nở là làm cho iết ng trong nh ảnh tăng lên và kích thước, các lỗ nh trong nh ảnh sẽ bị đóng biên ảnh biến hình o n rình ...

Phép toán co (erosion)

Phép toán co trong ảnh có nghĩa $A \ominus B = \{x | (B)_x \subseteq A\}$ trong ó A là iết ng trong ảnh, B là cuối trúc phím tinh. Ta cũng sẽ xét một ví dụ như trên với phép co trong ảnh. iết ng trong ảnh biến đổi bằng màu xám, cuối trúc phím tinh là khung có viền màu nâu, x là ảnh sau phép thay đổi mãn phép co nhau, 0 là ảnh không thay đổi.



Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



Ta thấy rằng sau phép toán này kết quả trong hình ảnh có lỗ, chính vì vậy mà nó có thể dùng trong việc giảm kích thước của các kết quả, tách rời các kết quả gần nhau và làm mờ nh, tìm kiếm kết quả.

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

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

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

Phép toán mở có thể dùng trong việc loại bỏ các phân lõm và làm cho khung bao kết quả trong hình trở lên mà không thay đổi.

Phép toán đóng có thể dùng trong việc làm tròn khung bao kết quả, lấp đầy các khoảng trống trên biên và loại bỏ những hố nhỏ (một số pixel bị thành cát lấp).

Trong OpenCV, các phép toán hình thái học trong hình ảnh cần thiết trong hàm `cv::morphologyEx`, riêng phép giãn nở và phép co có thể gộp thành 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& borderColor)`

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Trong ó, src, dst là nh u vào và nh sau phép x lý hình thái h c. op là ki u l a ch n phép hình thái h c, ch ng h n nh phép gi n n là MORPH_DILATE, phép óng là MORPH_OPEN element là c u trúc ph n t nh, có ba c u trúc c b n là theo kh i hình vuông, hình ch th p và hình elip. t o ra các c u trúc này ta có th t nh ngh a m t ma tr n v i các hình kh i t ng ng ho c s d ng hàm *getStructuringElement*, hàm này có c u trúc nh sau: *getStructuringElement(int shape, Size ksize, Point anchor)*, v i *shape* là ki u hình kh i (m t trong 3 hình kh i trên), *ksize* là kích th c c a hình kh i và là khích th c c a hai s nguyên l , *anchor* là i m neo và thông th ng nh n giá tr là $((ksize.width - 1)/2, (ksize.height - 1)/2)$. Thông s ti p theo *anchor* c ng có ý ngh a t ng t . *iterations* là s 1 n l p l i c a phép toán hình thái và hai thông s cu i cùng là v gi i h n biên c a nh ng i m nh n m ngoài kích th c nh trong quá trình tính toán, nó có ý ngh a nh các ví d trong bài tr c.

Phép toán v gi n n và co có th c g i t hàm *cv::morphologyEx* thông qua hai i s op là MORPH_DILATE và MORPH_ERODE ho c chúng có th c g i tr c ti p t hàm *cv::dilate* và *cv::erode*, C u trúc c a hai hàm này là t ng t nhau và các tham s hoàn toàn gi ng v i tham s trong hàm *morphologyEx*:

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

M t i u c n chú ý là trái v i cách di n t v các phép hình thái nh trên, OpenCV có cách cài t ng c l i gi a i t ng và n n nh, ngh a là trong phép gi n n (dilate), ph n c gi n n là n n c a nh (background) ch khong ph i các i t ng v t th trong nh, i u óc ng có ngh a r ng phép gi n n s làm co l i các i t ng v t th trong nh và phép co (erode) s làm co n n c a nh ng th i gi n n các i t ng v t th trong nh.

Sau ây ta s xét m t ví d v vi c s d ng các phép toán hình thái trong nh trong vi c phát hi n bi n s xe ô tô và cách ly các kí t trong bi n s .

Gi s ta có m t nh ch a bi n s , vì bi n s có n n tr ng và kí t màu en, nên tr c tiên ta nh phân nh ó, sau ó tìm ng bao c a các i t ng (contour) t ó xác nh c xem nh ng ng bao nào có kh n ng là bi n s xe nh t d a trên t l các c nh chi u dài, r ng c a hình ch nh t bao quanh ng bao ó (d nhiên ây là m t cách t i p c n r t n g i n!). Tuy nhiên, ta ch xác nh c ng bao quanh i t ng khi t p các i m n m trên biên c a nó t o thành m t vùng kín. Trong nhi u tr ng h p c a bi n s , vi c b m t m t vài i m nh trong quá trình nh phân trên biên là chuy n th ng xuyên x y ra và do ó s khó xác nh c m t ng bao quanh bi n s . kh c ph c

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

tình trạng này, ta sẽ làm gì là nén ảnh biên (tách các khung ảnh).
erode (làm mờ) làm giảm các ảnh trên biên trả lên lần lượt.

```
#include "stdafx.h"
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

using namespace cv;
using namespace std;

void main()
{
    Mat src1 = imread("BienSo.jpg", CV_LOAD_IMAGE_COLOR);
    Mat src2 = src1.clone(); // copy ảnh
    Mat gray, binary;
    cvtColor(src1, gray, CV_BGR2GRAY);
    threshold(gray, binary, 100, 255, CV_THRESH_BINARY);
    imshow("Anh nhi phan goc", binary);
    Mat morpho;
    Mat element = getStructuringElement(MORPH_CROSS, Size(3,3),
                                         Point(1,1));
    erode(binary, morpho, element, Point(-1,-1), 3);
    imshow("Anh sau khi thuc hien phep gian no", morpho);

    vector<vector<Point>> contours1;
    findContours(binary, contours1, CV_RETR_LIST, CV_CHAIN_APPROX_NONE );
    for(size_t i = 0; i < contours1.size(); i++)
    {
        Rect r = boundingRect(contours1[i]);
        if(r.width/(double)r.height > 3.5f && r.width/(double)r.height < 4.5f)
            rectangle(src1, r, Scalar(0, 0, 255), 2, 8, 0);
        else
            rectangle(src1, r, Scalar(0, 255, 0), 1, 8, 0);
    }
    imshow("Ket qua phat hien truoc phep gian no", src1);

    vector<vector<Point>> contours2;
    findContours(morpho, contours2, CV_RETR_LIST, CV_CHAIN_APPROX_NONE );
    for(size_t i = 0; i < contours2.size(); i++)
    {
        Rect r = boundingRect(contours2[i]);
        if(r.width/(double)r.height > 3.5f && r.width/(double)r.height < 4.5f)
            rectangle(src2, r, Scalar(0, 0, 255), 2, 8, 0);
        else
            rectangle(src2, r, Scalar(0, 255, 0), 1, 8, 0);
    }
}
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
imshow("Kết quả phát hiện sau khi phép gian nó", src2);  
waitKey(0);  
}
```

Trong chương trình trên, ta lần lượt thay đổi cách tìm biên với hai cách phân, một cách nhô nhô phân thông thường và một cách nhô nhô phân ánh sáng làm giảm thông qua hàm *erode*. Hàm *findContours* sẽ tìm khung bao quanh các ký tự riêng biệt có nhô nhô hóa và lumen bao này vào một vector là các điểm nằm trên khung bao đó, hàm *boundingRect* sẽ tìm ra một hình chữ nhật bao gồm tất cả các khung bao đã tìm ra từ hàm *findContours*. Để vào thời điểm này ta có thể xem xét liệu khung bao mà ta tìm được có phải là một vùng của biển số hay không, nếu phải ta sẽ hình chung t này thông qua hàm *rectangle* với một màu khác (*Scalar(0, 0, 255)*: màu đỏ) và nét mờ hơn. Kế tiếp chúng ta sẽ trình bày sau



Tất yếu rằng, ở một trong những cách lần nhau sẽ không xác định biển số (sau cùng là cần phải) nên ta tìm cách để hình bao khép kín quanh biển số, tuy nhiên ta cũng nhận thấy rằng khi các ký tự vẫn còn trong nhau sẽ bị tách ra, các ký tự sẽ có xu hướng dính vào nhau và việc tách các ký tự là khó khăn, chúng ta có thể giải quyết vấn đề này bằng cách áp dụng kỹ thuật chia nhỏ và cài đặt tách ra các ký tự.

9. Tìm biên nhấp nháy trên bối cảnh Canny

Bối cảnh Canny là sự kết hợp của nhiều bước khác nhau để tìm và tách riêng biên, kết quả là cho ra một đường biên khá chính xác. Quá trình tìm biên sử dụng phương pháp canny có thể chia thành 4 bước sau:

Bước 1: Loại bỏ nhiễu trong ảnh.

Ngoài ta loại bỏ nhiễu trong ảnh, làm cho ảnh mờ đi bằng cách nhân với một kernel Gausse, có kích thước 5x5 với h = s = 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 biên trong ảnh là nơi phân cách giữa các vật thể khác nhau, nên tách nó bằng gradient của nó sẽ biến đổi mạnh mẽ. Để tính toán gradient trong ảnh, ta có thể sử dụng bối cảnh Sobel, hoặc trực tiếp nhập vào ma trận nhân với các маски theo hướng x và y như sau:

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

Sau đó tính toán gradient trong ảnh:

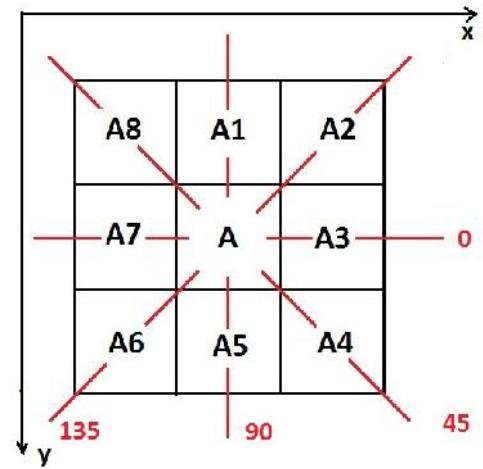
$$G = \sqrt{G_x^2 + G_y^2} \text{ và } \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Trong đó, G_x, G_y chính là hàm theo hướng X, Y của ảnh mà ta đang xét. Góc sẽ làm tròn theo các hướng thẳng đứng, nằm ngang và hướng chéo, nghĩa là nó sẽ làm tròn về những các giá trị 0, 45, 90 và 135°.

Bước 3: Loại bỏ các giá trị không phải là các vật thể

Bước này sẽ tìm ra những ảnh nhỏ có không phải là biên nhỏ nhặt bằng cách loại bỏ những giá trị không phải là các vật thể trong bước tìm gradient nhỏ trên. Ta thấy rằng, với giá trị của góc trên thì biên của vật thể có thể tuân theo bài toán, và ta có thể khử nhiễu sau:

4. $\text{Nu} = 0^0$, khi đó, ảnh A sẽ được xem là mảnh i m trên biên nếu 1 n gradient t i A l nh n 1 n gradient c a các i m t i A₃, A₇.
5. $\text{Nu} = 45^0$, khi đó, ảnh A sẽ được xem là mảnh i m trên biên nếu 1 n gradient t i A l nh n 1 n gradient c a các i m t i A₄, A₈
6. $\text{Nu} = 90^0$, khi đó, ảnh A sẽ được xem là mảnh i m trên biên nếu 1 n gradient t i A l nh n 1 n gradient c a các i m t i A₁, A₅.
7. $\text{Nu} = 135^0$, khi đó, ảnh A sẽ được xem là mảnh i m trên biên nếu 1 n gradient t i A l nh n 1 n gradient c a các i m t i A₂, A₆



Bài 4: Chuyển ra biên của một ảnh trong nh

Sau bài trước, ta thu được t p các i m t ng ng trên nh biên khá m ng. Vì nh ng i m có giá tr gradient l nh bao gi c ng có xác su t là biên th t s h n nh ng i m có giá tr gradient bé, o ó xác nh chính xác h n n a biên c a các i t ng, ta s d ng các ng ng. Theo đó, b 1 c canny s s d ng m t ng ng trên (upper threshold) và m t ng ng d i (lower threshold), n u gradient t i m t i m trong nh có giá tr l nh n ng ng trên thì ta xác nh n ó là m t i m biên trong nh, n u giá tr này bé h n ng ng d i thì ó không ph i i m biên. Trong tr ng h p giá tr gradient n m gi a ng ng trên và ng ng d i thì nó ch c tính là i m trên biên khi các i m liên k t bên c nh c a nó có giá tr gradient l nh n ng ng trên.

Tìm biên nh b ng b l c canny trong OpenCV:

OpenCV cung cấp m t hàm cho ta xác nh biên trong nh c a các i t ng. Nguyên m u c a hàm này nh sau :

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

Trong ó, `src` là nh c n phát hi n biên (là nh xám), `dst` là nh ch a biên c phát hi n, `lower_threshold`, `upper_threshold` là ng ng d i, ng ng trên nh á nói bên trên, `kernel_size` là kích th c c a m t n dùng cho b c th hai tính toán gradient c a các i m trong nh.

Ch ng trình demo tìm biên trong nh d a trên b 1 c canny:

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
#include "stdafx.h"
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace cv;

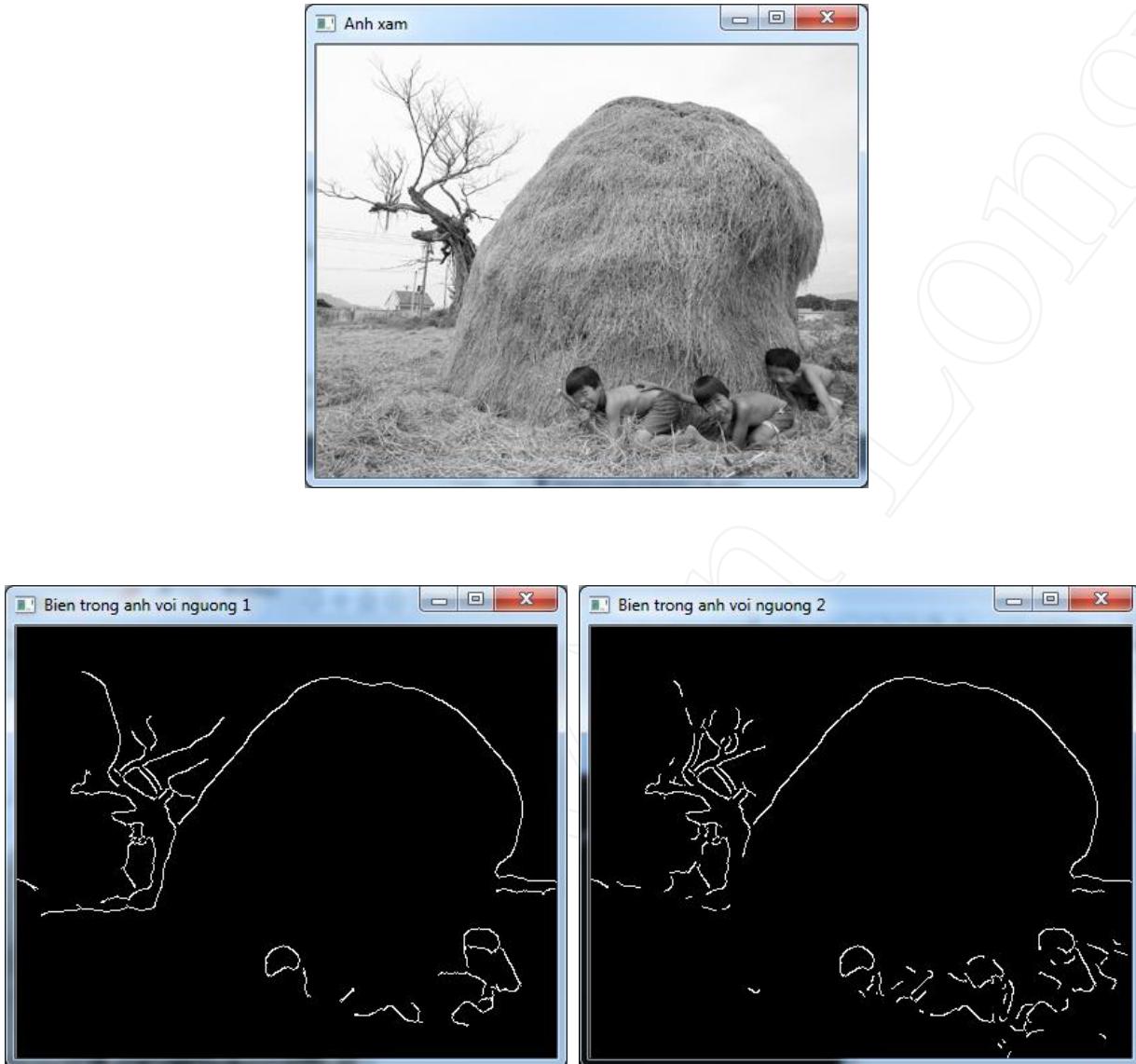
void main()
{
    Mat gray = cv::imread("TuoiTho.jpg", CV_LOAD_IMAGE_GRAYSCALE);
    Mat dst1, dst2;
    imshow("Anh xam", gray);
    GaussianBlur(gray, gray, Size(9,9), 2);
    double t1 = 30, t2 = 200;
    Canny(gray, dst1, t1, t2, 3, false);
    t1 = 100; t2 = 120;
    Canny(gray, dst2, t1, t2, 3, false);
    imshow("Bien trong anh voi nguong 1", dst1);
    imshow("Bien trong anh voi nguong 2", dst2);
    waitKey(0);
}
```

Trong chương trình trên, trước khi tìm biên bằng phương pháp canny ta đã làm tròn mảng xám bằng bù lỗ *GaussianBlur* (vì có làm này có ý nghĩa trong việc giảm thiểu các tiếng ồn không mong muốn trong ảnh biên sau này). Vô nguyên tắc, bù lỗ *GaussianBlur* cũng là một bước sMOOTHING (giả thiệu bài toán với các khía cạnh 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 điều khiển vào và kết quả. Một nhược điểm có thể coi là mất thời gian để lặp đi lặp lại, do đó src và dst chính là nhau và nhau thuần túy sau khi biến đổi, *ksize* là kích thước của ma trận lọc, *sigmaX*, *sigmaY* là 1 chuỗi theo hàng x và y. Nếu *sigmaY = 0*, 1 chuỗi theo hàng y sẽ chỉ lặp theo *sigmaX*. *BorderType* là cách xử lý biên giới hình ảnh mờ tràn ra ngoài kích thước của ảnh.

Kết quả thuần túy sau các tham số *t1*, *t2* khác nhau, ta có những biên có chi tiết khác nhau như hình sau.

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



10. Chuyển đổi Hough, Phát hiện đường cong, ng tròn trong nh

Chuyển đổi Hough (Hough transformation) là một phương pháp được dùng nhiều trong phân tích và xử lý ảnh, mà có ích chính là phương pháp này là tìm ra những hình dáng có trong nhau bằng cách chuyển đổi không gian sang một không gian các tham số mà nêu gì n quá trình tính toán, trong bài này ta xét chuyển đổi Hough cho đường cong và ng tròn.

Chuyển đổi Hough cho đường thẳng.

Tại đây bài toán, mảng đường thẳng trong không gian hai chiều có thể được biểu diễn dưới dạng $y = kx + m$ và có phẳng gốc k , giá trị m có thể thay đổi làm trung cho một đường thẳng. Tuy nhiên, cách biểu diễn theo cặp (k, m) khó thám mãn với hình ảnh đường thẳng mà chỉ là một số vô cùng. Để tránh trường hợp này, ta sử dụng cách biểu diễn đường thẳng trong hệ tọa độ sau:

$$r = x\cos(\theta) + y\sin(\theta)$$

Trong đó, r là khoảng cách tia giao của đường thẳng, θ là góc của.

Như vậy, với mỗi điểm (x_0, y_0) ta có một đường thẳng đi qua thỏa mãn phương trình

$$r_0 = x_0\cos(\theta) + y_0\sin(\theta)$$

Phương trình này biểu diễn một đường cong, nhảy vị trong một tần số có nốt (n pixel) ta sẽ có n các đường cong. Nếu những cong của các điểm khác nhau giao nhau, thì các điểm này cùng thu về một đường thẳng.

Bằng cách tính các giao điểm này, ta xác định đường thẳng, nó là sử dụng ý tưởng của thuật toán Hough cho đường thẳng.

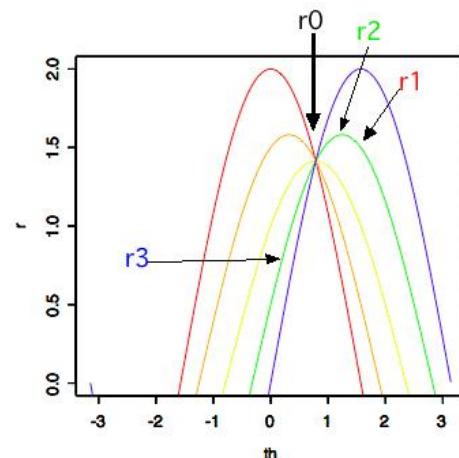
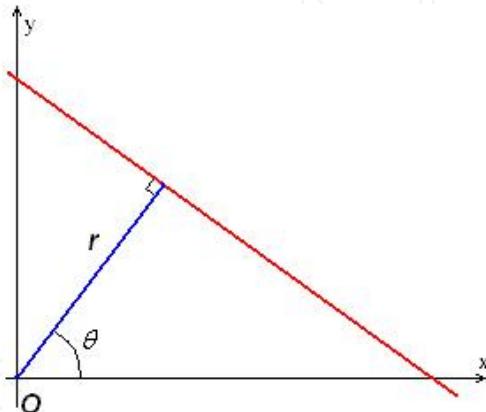
Chuyển đổi Hough cho đường tròn

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

$$\begin{cases} x = u + R\cos\theta \\ y = v + R\sin\theta \end{cases}$$

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

$$\begin{cases} u = x - R\cos\theta \\ v = y - R\sin\theta \end{cases}$$



Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Ta xét về trung tâm hình ảnh giá trị R . Khi đó, với mỗi điểm (x,y) ta xác định cung tròn có trung tâm (u,v) và lùi nó vào mặt màng. Tâm của cung tròn sẽ là giá trị xung quanh trong mặt màng với bán kính R chia thành t phần, tia trung bình giá trị R là $\frac{R}{t}$ và minh chứng rằng max nào đó và tiến hành nhảy vị trí hình ảnh R để tìm giá trị R .

Tìm đường thẳng, cung tròn trong nhau.

Tìm đường thẳng trong nhau, thay vì `cv::HoughLines` và `cv::HoughLinesP`, hai hàm này vẫn bao gồm thuật toán, nhưng khác nhau.

Đang 1: arracktquist cách tìm đường thẳng là một cách giá trị (r, θ)

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

Trong đó, `src` là nhánh phân chia biên của các điểm ảnh cần phát hiện đường (trong thực tế ta có thể sử dụng là một nhánh xám), `lines` là vector chứa các điểm (r, θ) của đường phát hiện, `rho` là phân giải pixel (tính theo pixel, thường là 1), `theta` là phân giải góc (có giá trị từ 0 đến 360°), `threshold` là giá trị nhỏ nhất của tần số giao điểm của các đường cong xác định đường, `param` là một thông số mà OpenCV chưa định rõ (nó có giá trị bằng 0).

Đang 2: arracktquist là tách riêng và tìm kiếm cách tìm đường

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

Trong đó, các thông số `src, rho, theta, threshold` giống như đang 1, `lines` là vector chứa tách riêng và tách riêng nhau có thể xem nó là một đường (tính theo nút pixel), `minLength` là dài nhất có thể xem nó là một đường (tính theo nút pixel), `maxGap` là khoảng cách lớn nhất giữa hai tách riêng nhau xác định chúng có thuộc mảng đường hay không (tính theo nút pixel).

Để tìm cung tròn, ta có thể áp dụng hàm `cv::HoughCircles()`, nguyên mực của hàm như sau:

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

Trong đó, `src` có thể là một nhánh phân chia biên của một hình ảnh (chạy trình sốt để dò biên bằng phương pháp canny), `circles` là vector chứa phát

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

hi n
ng tròn v i 3 tham s (a, b, R), method là ph
ng pháp phát hi n
ng tròn
(hi n t i ph
ng pháp trong OpenCV là *CV_HOUGH_GRADIENT*), *dp* là t s
ngh ch
o c a
phân gi i (áp d
ng trong ph
ng pháp hi n t i), *min_dist* là kho ng cách nh
nh t gi a hai tâm
ng tròn phát hi n
c, *param1*, *param2* là các thông s
ng ng
trên và ng
ng d
i ph c v
cho vi c phát hi n
biên b
ng ph
ng pháp canny,
min_radius và *max_radius* là gi i h n nh
nh t, l n nh t c a bán kính
ng tròn ta c n
phát hi n(N u ta không bi n bán kính, m c
nh hai giá tr này b
ng không thì ch
ng
trình s 1 n t ng giá tr bán kính m t cách t
ng
tìm ra t t c
các
ng tròn trong
nh).

Trong ó: ...

Ch
ng trình tìm
ng th
ng,
ng tròn trong
nh:

```
#include "stdafx.h"
#include <opencv2\core\core.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;

void main()
{
    Mat src = imread("DongHo.jpg", 1);
    Mat gray;
    cvtColor(src, gray, CV_BGR2GRAY);
    GaussianBlur(gray, gray, Size(9, 9), 2, 2 );

    // Tim duong thang
    Mat canny;
    Canny(gray, canny, 100, 200, 3, false);
    vector<Vec4i> lines;
    HoughLinesP(canny, lines, 1, CV_PI/180, 50, 60, 10);

    // Tim duong tron
    vector<Vec3f> circles;
    HoughCircles(gray, circles, CV_HOUGH_GRADIENT, 1, 100, 200, 100, 0, 0);

    // Ve duong thang, duong tron len anh
    for(int i = 0; i < lines.size(); i++ )
    {
        Vec4i l = lines[i];
        line(src, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 2);
    }
}
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
for(int i = 0; i < circles.size(); i++ )  
{  
    Point center(cvRound(circles[i][0]), cvRound(circles[i][1])); // Tâm  
    int radius = cvRound(circles[i][2]); // Ban kinh  
    circle(src, center, radius, Scalar(0,0,255), 2, 8, 0 );  
}  
  
imshow("Anh sau khi tim thay duong thang - Duong tron", src);  
waitKey(0);  
}
```

Trong chương trình trên, ta tiên nhât sẽ vào một biến `src`, sau đó sẽ chuyển qua ảnh xám `gray` và sẽ làm tròn ảnh bằng hàm `GaussianBlur`. Sau đó vì có phát hiện đường thẳng, ta sẽ phát hiện ra tất cả các điểm biên của đường, do vậy ta sử dụng hàm `canny` để nhận phân chia tất cả các điểm biên. Trong trường hợp phát hiện đường tròn, hàm `HoughCircles` sẽ tìm kiếm cho ra thông tin qua hai thông số (trong chương trình trên là 200 và 100) và do đó ta không cần phải tách riêng viền này. Kết quả thu được sẽ như hình sau.



Chương III. Lập trình xử lý ảnh với giao diện MFC

1. Giới thiệu về MFC

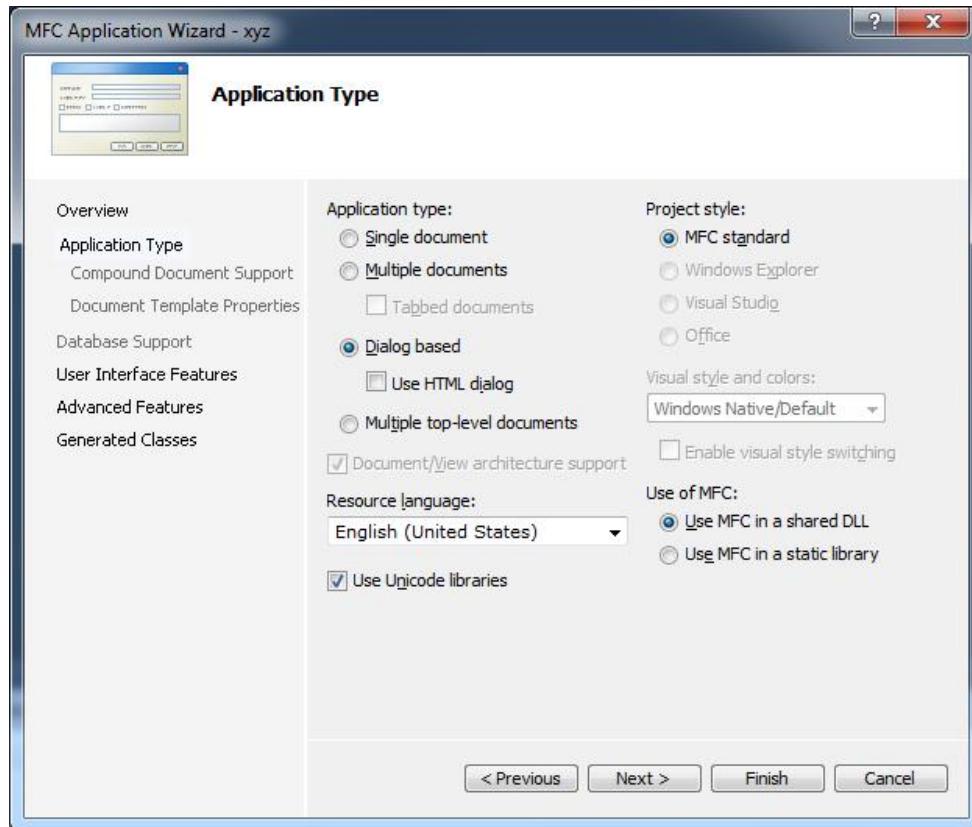
MFC (Microsoft Foundation Classes) là bộ thư viện được Microsoft phát triển phục vụ cho việc lập trình trên Windows. Bộ công cụ này là cung cấp cho ta các lớp, các công cụ làm việc với các hàm API của Windows, do vậy việc lập trình trở nên nhanh chóng và tự động.

Trong phần này và phần sau, các ví dụ và hướng dẫn sẽ cung cấp cho bạn vào giao diện Dialog của MFC. Vì nghiên cứu một cách kỹ và bài bản về MFC là một việc cần thiết và tốn thời gian và công sức. Trong khuôn khổ cuốn sách này, ta chỉ xem xét một phần nhỏ và các thủ thuật làm việc nhanh chóng với MFC.

2. Khởi tạo project MFC

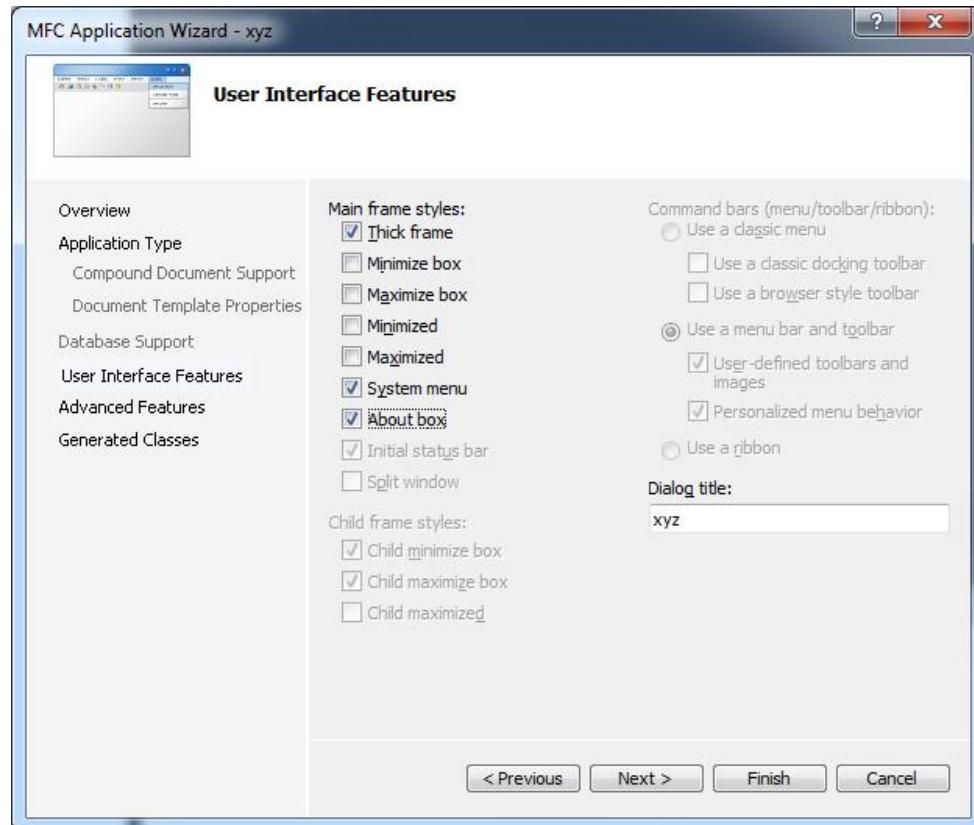
Khi mở Visual Studio, từ menu chọn *File -> New -> Project* (hoặc nhấn Ctrl + Shift + N). Khi đó New Project hiện ra, trong Visual C++ (có thể phân chia thành Other language trước khi hiện ra Visual C++) sau đó chọn *MFC Application*. Tạo tên cho project trong trường *Name* (giả sử tên là xyz) sau đó click OK để tiếp tục. Tiếp theo, tiếp tục chọn *Next* để tiếp tục.

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

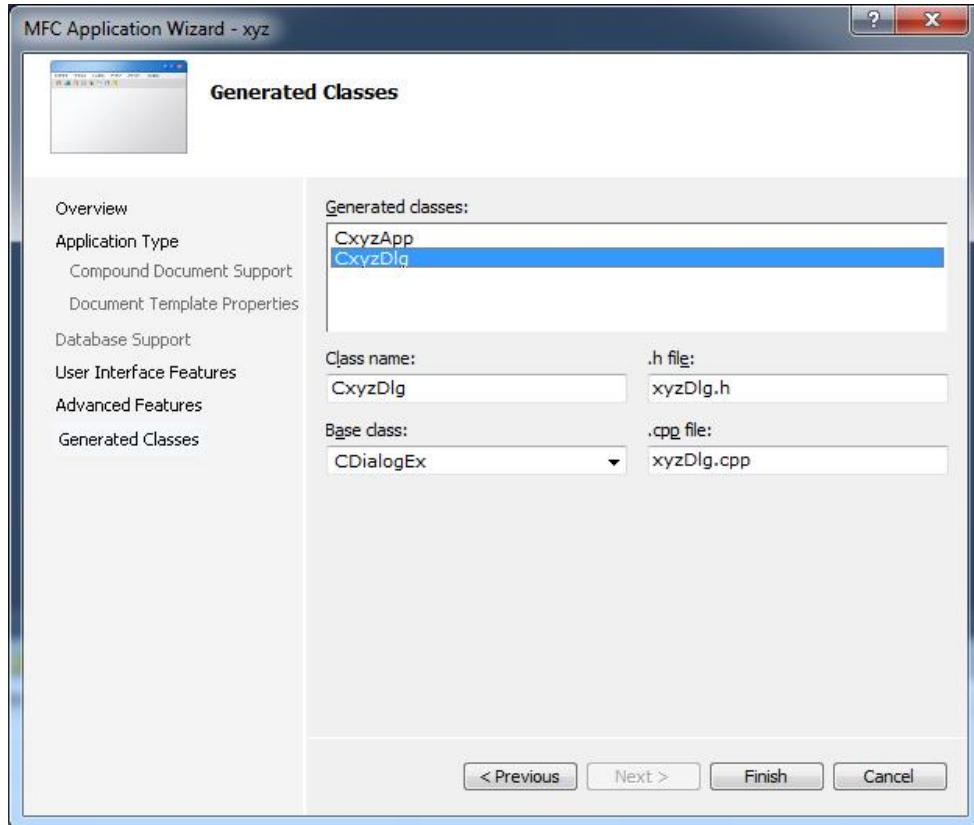


Application type ch ọn *Dialog based*, *Project type* ch ọn *MFC standard*... Chú ý r ằng vi c tick vào ch ọn *Use Unicode libraries* s ẽ có nh ững ý nghĩa và cách dùng khác nhau, ta s xét tr ong h elp này sau. Ta nh ấn *next* để tiếp h ỗn theo i ti p theo, h ỗn tho i này cho phép ta tùy ch nh m t s ch c n ng c a c a s nh ẽ có thêm nút phóng to, thu nh , menu... có th m c nh và ch ọn *next*

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



Ta nhấn *Next* để tiếp tục tùy chỉnh thêm các tùy chọn nâng cao, nếu chưa rõ ta có thể nhấn và nhấp *Next*. Hophil tho ị cuối cùng xuất hiện yêu cầu ta chọn MFC toolkit. Ta chọn là *CxyzDlg* và nhấn *Finish*.



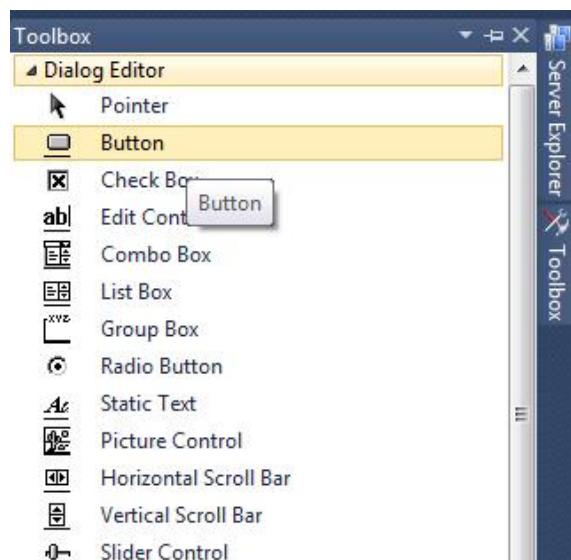
này ta đã khai báo xong một project MFC có dựa trên một Dialog, Dialog hiện ra mà có inh có một button OK, button Cancel và một label, ta có thể sử dụng hoặc xóa đi và thiền theo ý riêng của mình.

3. Làm việc với các iu khi n (Control) c a MFC

t tên biến cho các control

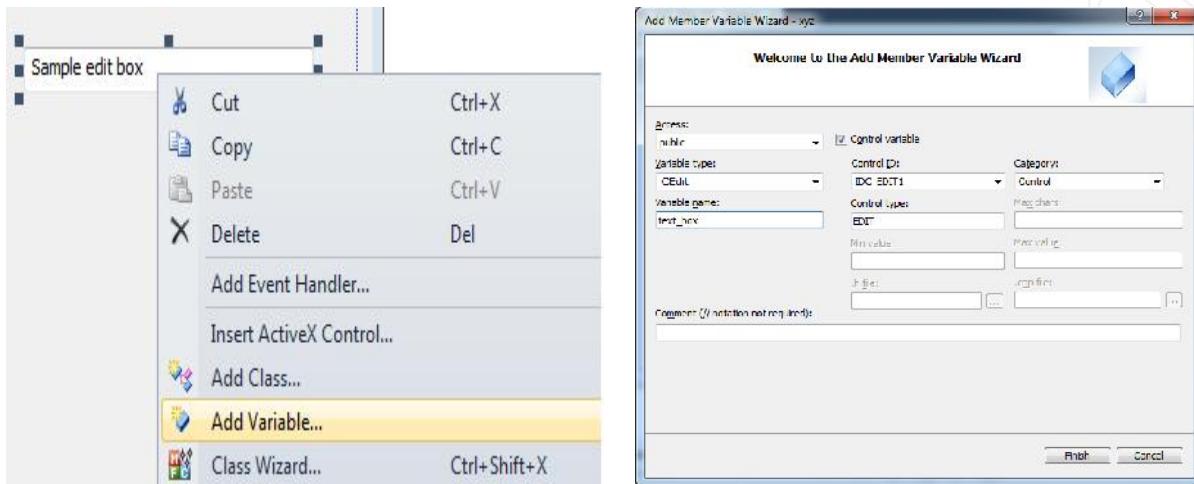
Khi muốn sử dụng một Control nào, ta kéo control đó từ Toolbox và cho vào dialog. làm việc với các control một cách dễ dàng, ta nên đặt tên biến cho các control.

Đặt tên biến cho một control, ta click chuột phải vào control, sau đó chọn Add Variable. Màn hình hiển thị Add Member Variable Wizard hiện ra và trong mục variable name ta đặt tên cho control đó. Chú ý là chỉ với các control mà ID của nó có dạng IDC_STATIC (như Static Text) thì ta chỉ có thể đặt tên biến khi i



Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

ID c a nó, ch ng h n nh i thành IDC_STATIC1, IDC_LABEL ...



L y giá tr nh p vào t m t Edit Control

L y giá tr n nh p vào t m t Edit Control, ta s d ng hàm `GetWindowText()`. Gi s nh Edit Control c t tên là `text_box`, khi ó ta có th l y giá tr text trong ô Edit Control nh sau:

```
CString text;  
text_box.GetWindowTextW(text);
```

Giá tr l y c s c l u trong m t chu i `CString text`. M t i m c n chú ý t gi v sau là các hàm trong MFC khi c build ch Unicode và ch Multi-byte thì các g i các hàm, cách s d ng các hàm c ng có nh ng i m khác nhau nho nh , ch ng h n nh c ng là l y giá tr trong m t ô Edit Control nh ng n u ch Multi-byte ta s dùng l nh sau:

```
CString text;  
text_box.GetWindowTextA(text);
```

Ta có th tùy ch nh trình d ch build theo ch Unicode ho c Multi-byte b ng cách vào *Project -> Properties* (ho c nh n Alt + F7), h p tho i *Properties* hi n ra, ch n *Configuration Properties -> General* và tùy ch nh m c *Character Set*.

Hi n th text lên các control

hi n th text lên các control (Button, Edit control, Static Text ..), ta dùng hàm `SetWindowText(CString text)`

```
text_box.SetWindowText(_T("text")); // Unicode  
text_box.SetWindowText("text"); // Multi-byte
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Hiển thị nhôm lên một control

Hiển thị nhôm lên một control, ta dùng hàm `SetBitmap()`. Ở code sau load một nhôm bitmap từ D và hiển thị lên một button có tên là `btn`:

```
HBITMAP image = (HBITMAP)LoadImageA(0, "D:/test.bmp", IMAGE_BITMAP, 0, 0,  
LR_LOADFROMFILE);  
btn.SetBitmap(image);
```

Enable, disable một control

Tạo sẵn ng hàm `EnableWindow` cho phép control có phép sờ ng hay không

```
text_box.EnableWindow(false); // Vô hiệu hóa edit control
```

```
text_box.EnableWindow(true); // Cho phép edit control hoạt động
```

Lấy giá trị thanh trượt (Slider Control)

Giá trị thanh trượt có tên là `slider`, khi đó ta có thể lấy giá trị hiện tại của thanh trượt bằng hàm `GetPos()`:

```
int value = slider.GetPos();  
Ta cũng có thể dùng hàm SetRange để giá trị lớn nhất và bé nhất cho thanh trượt, và  
dùng hàm SetPos để thiết lập giá trị cho thanh trượt:  
slider.SetRange(0, 100); slider.SetPos(50);
```

Lấy giá trị lựa chọn (Combo Box)

Combo box cho phép ta lựa chọn, chuyển đổi giữa các lựa chọn một cách nhanh chóng.

Thêm các lựa chọn vào Combo Box ta có thể nhập trực tiếp vào một `Data` trong properties của nó, (giả sử ta có các lựa chọn về tên thành Việt Nam như HaNoi, ThanhHoa, DaNang ...). Khi click chuột phím vào Combo box, chọn properties, trong bảng properties có Data ta nhìn vào HaNoi;ThanhHoa;DaNang... các lựa chọn có phân cách bằng ";"). Xem lựa chọn nào đang có chia sẻ, ta dùng hàm `GetCurSel()`.

```
int choice = combo_box.GetCurSel(); Giá trị trả về là thứ tự các dữ liệu trong mục Data của Combo box
```

Dialog mở file và lưu file

Một cách cơ bản dialog này là tạo ra một hộp thoại cho phép người dùng chọn một file và lưu file. Khi quay cuộn cùng mà ta quan tâm nhất là lấy ra ngay tên mà người dùng lựa chọn.

```
CString Filter=_T("image files (*.bmp; *.jpg) |*.bmp;*.jpg|All Files  
(*.*)|*.*||");  
CFileDialog dlg(TRUE, _T("*.jpg"), NULL,  
OFN_FILEMUSTEXIST|OFN_PATHMUSTEXIST|OFN_HIDEREADONLY,Filter,NULL); // Mở file  
CFileDialog dlg(FALSE, _T("*.jpg"), NULL,  
OFN_FILEMUSTEXIST|OFN_PATHMUSTEXIST|OFN_HIDEREADONLY,Filter,NULL); // Lưu file  
Filter sẽ chỉ hiển thị những file không mà ta cần quan tâm, trong trường hợp trên ta đang xét mở hoặc lưu một file nhỏ do đó file mở rộng là bmp hoặc jpg. Nếu muốn hiển thị tất cả các loại file ta chỉ việc filter là *.* , hoặc thêm file và lưu file
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

ch khac nhau thong s u tiên khi t o it ng *dlg*, n u là m file ta t là *TRUE*, l u file ta t là *FALSE*.

Ta hi n th h p tho i này và l y ng d n ng i dùng ch n nh sau:

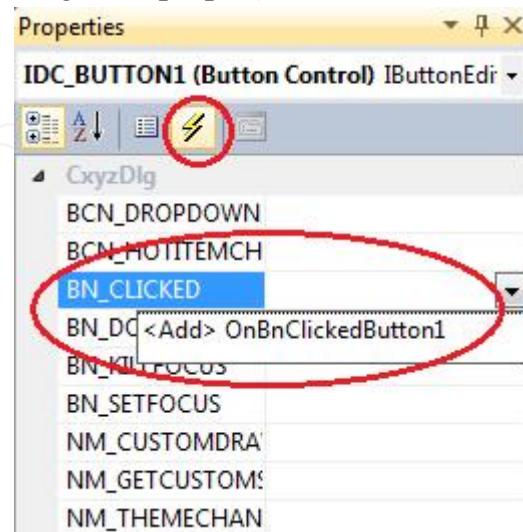
```
if(dlg.DoModal() == IDOK)
{
    CString file_name = dlg.GetPathName(); // lay duong dan day du
}
```

X lý s ki n khi click chu t vào button

H u h t các control trong MFC u có m t ho c nhi u s ki n khi ng i dùng t ng tác v i nó, ch ng h n s ki n click chu t vào button, s ki n kéo thanh tr t c a slider control ... x lý các s ki n cho các control, trong m c properties c a control t ng

ng ta ch n vào icon s ki n sau ó ch n các s ki n c n x lý. Ch ng h n i v i button, ta ch n *BN_CLICKED* r i ch n *OnBnClickedButton1*, khi ó ta s có m t hàm c t ng sinh ra và ta có th x lý các v n liên quan t i s ki n click chu t. Hàm sau s sinh ra m t Message Box khi click vào button1

```
void CxyzDlg::OnBnClickedButton1()
{
    MessageBoxA(NULL, "Button1 duoc
click", "Thong bao", 0);
}
```



X lý s ki n khi thay i l a ch n Combo Box

S ki n thay i l a ch n c a Combo Box là *CBN_SELENDOK*. o n code sau mô t s thay i l a ch n c a ng i dùng trên Combo Box, v i m i l a ch n ta sinh ra m t Message Box t ng ng.

```
void CxyzDlg::OnCbnSelendokCombo1()
{
    int index = combo_box.GetCurSel();
    switch(index)
    {
        case 0:
            MessageBoxA(NULL, "Ban chon HaNoi", "Thong bao", 0);
            break;
        case 1:
            MessageBoxA(NULL, "Ban chon ThanhHoa", "Thong bao", 0);
            break;
    }
}
```

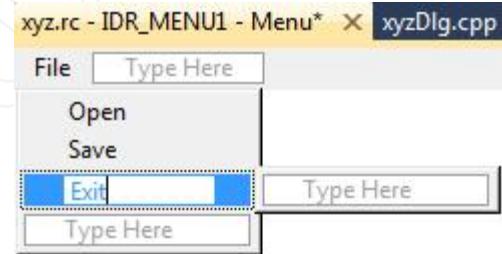
Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
case 2:  
    MessageBoxA(NULL, "Ban chon DaNang", "Thong bao", 0);  
    break;  
default:  
    MessageBoxA(NULL, "Khong chon?", "Thong bao", 0);  
    break;  
}  
}
```

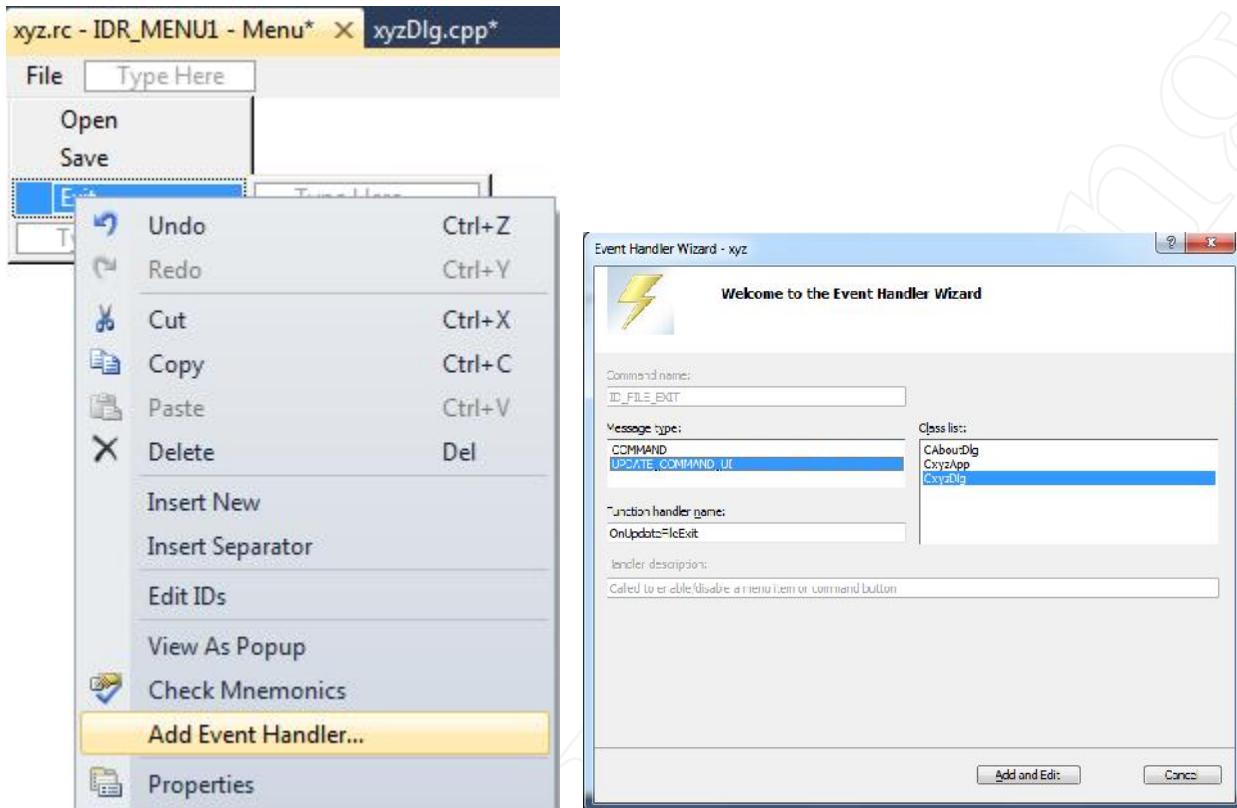
Thêm menu vào chương trình, xem lý sao khi click vào menu

Menu trong MFC xem là resource của chương trình. Vì vậy thêm menu vào dialog đòi hỏi ta phải thêm nó vào resource của chương trình. Trước hết ta hãy xem các resource của chương trình bằng cách mở menu của Visual Studio chọn View -> Resource View (hoặc nhấn phím Ctrl + Shift + E). Khi Resource View hiện ra, ta click chuột phải vào ô, chọn Add->Resource... Các Add Resource hiện ra, ta chọn Menu và click vào button New. Ngay sau đó ta sẽ có một resource mới menu chung, ta tiến hành đặt tên cho các menu mà ta muốn có trong chương trình như n. Hình bên ta thấy 3 menu chung là Open, Save và Exit... Khi đã tạo xong menu, nó vẫn chưa có trong resource của chương trình, menu này cần vào dialog khi chạy, ta vào properties của dialog, trong mục Menu chung IDR_MENU1, với IDR_MENU1 chính là ID của menu ta đặt trước.

Xem lý sao khi click chuột vào menu nào, ta click chuột phải vào menu và chọn Add Event Handler...



Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



h p tho i *Event Handler Wizard* ta ch n l p mà ta mu n thêm menu vào sau ó click vào button *Add and Edit* i t i hàm x lý s ki n khi click vào menu, ví d sau là ta g i hàm *OnCancel()* thoát kh i ch ng trình

```
void CxyzDlg::OnUpdateFileExit(CCmdUI *pCmdUI)
{
    OnCancel();
}
```

Ngoài m t s i u khi n thông d ng ã nh c t i trên, MFC còn cung c p r t nhi u các i u khi n khác giúp cho vi c t o ra giao di n m t cách d dàng và p m t h n. B n c tham kh o thêm các tài li u khác v ph n này.

4. Chuy n i các ki u d li u trong MFC

Các ki u d li u c a MFC v c b n là gi ng v i các ki u d li u trong C, tuy nhiên có m t s tr ng h p ta ph i chuy n i qua l i gi a các ki u d li u phù h p v i u vào, u ra c a m t vi c nào ó, ch ng h n khi ta dùng CFileDialog m m t ng d n sau ó c nh t ng d n này, k t qu tr v ng d n c a CFileDialog là m t chu i CString tuy nhiên hàm cv::imread l i c nh t m t chu i string, do ó ta ph i

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

chuyển đổi từ CString sang string. Một số chuyển đổi sau đây là hữu ích cho việc hiển thị giao diện, và lý do là vì nó giúp ta có thể hiển thị hai kiểu dữ liệu này dễ dàng:

- Chuyển đổi string sang CString và ngược lại*
- Vì chỉ biên độ không sử dụng Unicode, ta có thể chuyển đổi hai kiểu dữ liệu này dễ dàng:
8. Chuyển đổi string sang CString:

```
std::string str = "chuoi string";
CString cstr = str.c_str();
```
 9. Chuyển đổi CString sang string

```
CString cstr = "chuoi cstring";
std::string str = std::string(cstr);
```

vì chỉ biên độ có sử dụng Unicode:

10. Chuyển đổi string sang CString:

```
std::string str = "chuoi string";
CStringW cstr = (CStringW)(str.c_str());
```
11. Chuyển đổi CString sang string

```
CString cstr = _T("chuoi cstring");
std::wstring wstr = (std::wstring)cstr;
std::string str;
str.assign(wstr.begin(), wstr.end());
```

Chuyển đổi sang CString và ngược lại

Cách chuyển đổi này giúp ta dễ dàng sử dụng các ô Edit Control hoặc hiển thị lên các control.

12. Chuyển đổi CString sang số

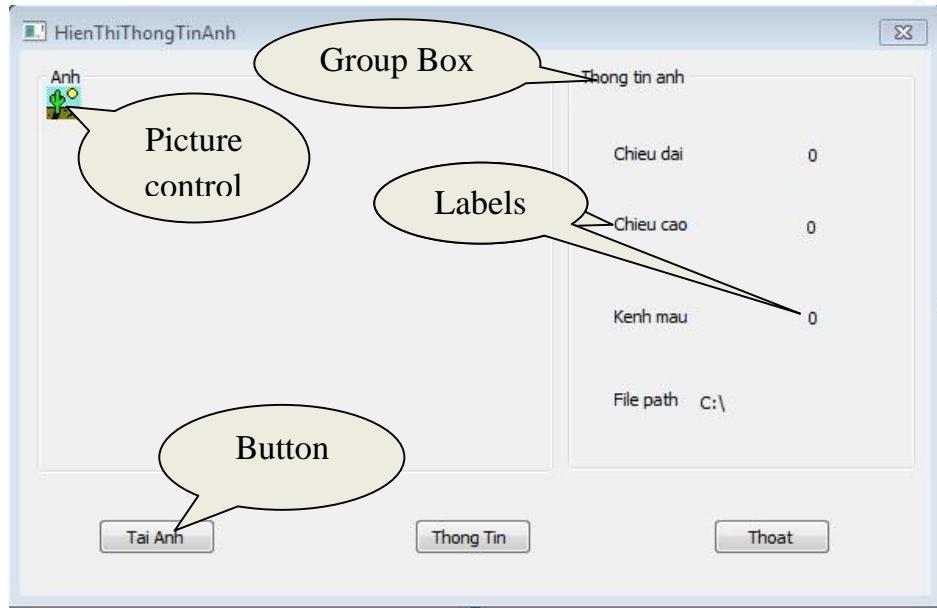
```
CString cstr1 = "123";
int num1 = atoi(cstr1); // Chuyển sang số nguyên không sử dụng Unicode
int num1 = _wtoi(cstr1); // Chuyển sang số nguyên sử dụng Unicode
CString cstr2 = "10.5";
float num2 = atof(cstr2); // Chuyển sang số thực không sử dụng Unicode
float num2 = _wtof(cstr2); // Chuyển sang số thực sử dụng Unicode
```
13. Chuyển đổi sang CString

```
char s1[20];
int num = 10;
sprintf(s1, "%d", num);
CString cstr1 = s1; // chuyển số nguyên sang cstring - không unicode
CString cstr2;
cstr2.Format(_T("%d"), num); // chuyển số nguyên sang cstring unicode
Tuy nhiên có thể chuyển đổi bằng cách thay "%d" bằng "%f".
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

5. Chương trình tính và hiển thị thông tin trên giao diện MFC

Tạo một mới mẻ project, tên project là *HienThiThongTinAnh* và thiết kế giao diện hình sau:



Trong đó, các Labels hiển thị thông tin chính là *Static Text*, ID *IDC_STATIC* của chúng có tên là *l_width*, *l_height*, *l_channels*, *l_path* (tất cả các static text dùng để hiển thị tóm tắt). ID của Picture Control cũng thành *IDC_STATIC_PICTURE*.

Bây giờ, ta sẽ tiến hành xử lý sẵn sàng cho các button *Tai Anh*, *Thong Tin* và *Thoat*.

```
void CHienThiThongTinAnhDlg::OnBnClickedButton1()
{
    // Load ảnh và hiển thị ảnh

    static CString Filter=_T("image files (*.bmp; *.jpg) |*.bmp;*.jpg|All
Files(*.*|*.*|");

    CFileDialog Load(TRUE, _T("*.jpg"), NULL,
OFN_FILEMUSTEXIST|OFN_PATHMUSTEXIST|OFN_HIDEREADONLY, Filter, NULL);
    Load.m_ofn.lpstrTitle= _T("Load Image");
    if (Load.DoModal() == IDOK)
    {
        path = Load.GetPathName();
        std::string filename(path);
        src = cv::imread(filename,1);
        cv::namedWindow("Hien thi anh", 1);
        HWND hWnd = (HWND) cvGetWindowHandle("Hien thi anh");
        HWND hParent = ::GetParent(hWnd);
        ::SetParent(hWnd, GetDlgItem(IDC_STATIC_PICTURE)->m_hWnd);
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
    ::ShowWindow(hParent, SW_HIDE);
    // resize và hiển thị ảnh
    cv::Mat dst;
    cv::resize(src, dst, cv::Size(320, 240), 0, 0, 1);
    cv::imshow("Hiển thị ảnh", dst);

}

}
```

Trong hàm trên hàm `cvGetWindowHandle()` trả về một `HWND` của các ảnh hiển thị trong OpenCV, sau đó hàm `SetParent()` sẽ đặt cửa sổ này vào cửa sổ của Picture Control của MFC, đây là cách ta dùng hiển thị ảnh lên bất kỳ control nào của MFC thay vì chuyển nó sang `HBITMAP` và dùng hàm `SetBitmap`. Hàm `cv::resize()` giúp chuyển đổi kích thước có kích thước 320x240 giúp hiển thị trong kích thước của Picture Control. Cuối cùng, khi gọi hàm `cv::imshow()` trong OpenCV, ta sẽ thấy nó hiển thị trong Picture Control.

```
void CChienThiThongTinAnhDlg::OnBnClickedButton2()
{
    // Hiển thị thông tin ảnh
    if(src.empty())
        MessageBoxA("Chưa tải ảnh", "Thông báo", 0);
    else
    {
        int width_ = src.cols;
        int height_ = src.rows;
        int channels_ = src.channels();

        CString result;
        result.Format("%d", width_);
        l_width.SetWindowText(result);

        result.Format("%d", height_);
        l_height.SetWindowText(result);
        result.Format("%d", channels_);

        l_channels.SetWindowText(result);
        l_path.SetWindowText(path);

    }
}
```

Trong hàm trên, trước tiên ta kiểm tra xem nó có thể hiển thị hay không, nếu có thì hiển thị thông báo cho biết có ảnh, nếu không hiển thị tính toán chỉ số dài, chỉ số rộng, kênh màu của nó... sau đó hiển thị thông tin này lên các label tương ứng. Hiển thị lên label họ và tên khi nào có cửa sổ của MFC ta dùng thuần túy

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

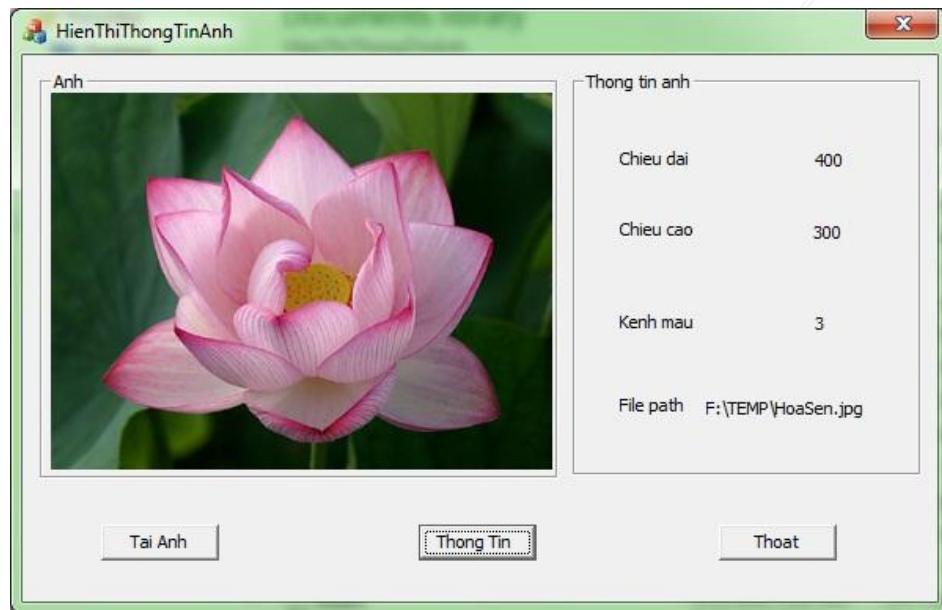
`SetWindowText()`. Tuy nhiên chỉ sau vào cách này là có thể dùng `CString`, do vậy các giá trị `int, float, double ..` đều không thể trên các biến khi nêu trên.

button `Thoat`, khi click vào vào chương trình sẽ thoát ra, vì cách thoát khác là tắt ứng dụng, có thể thực hiện bằng cách gọi hàm `OnCancel()`.

```
void CHienThiThongTinAnhDlg::OnBnClickedButton3()
{
    // Thoat khoi chuong trinh
    this->OnCancel();

}
```

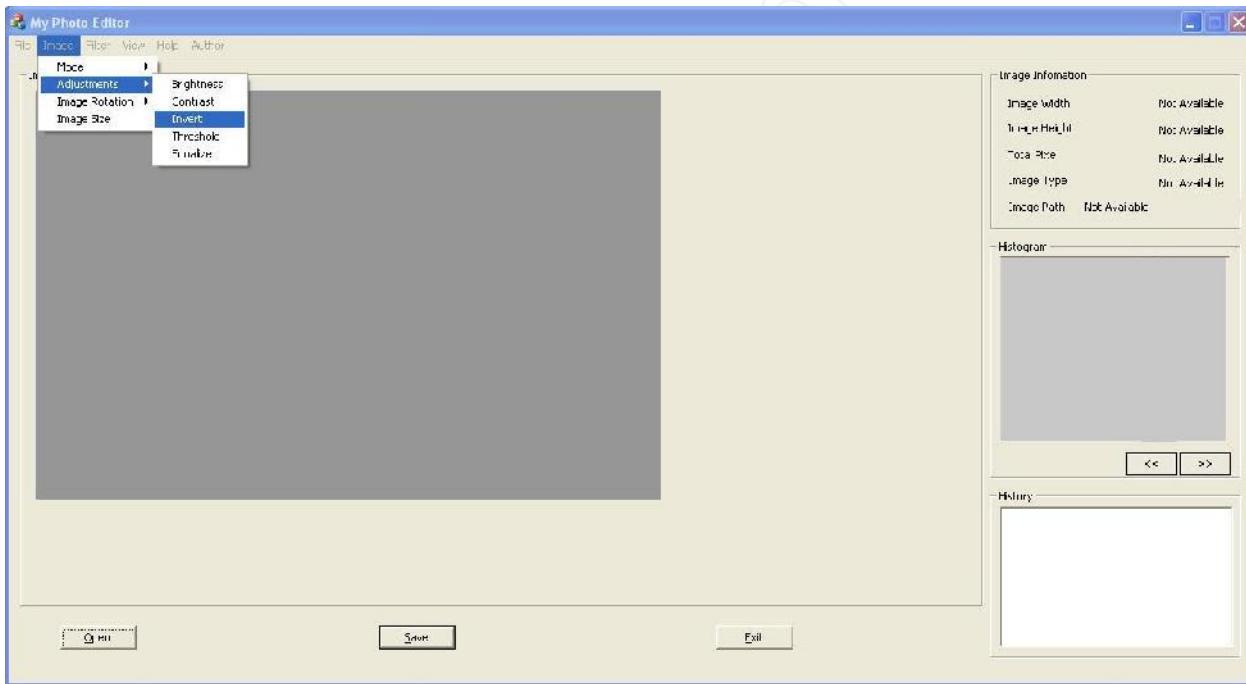
Kết quả chạy chương trình như sau:



Chương IV Mô tả ứng dụng xử lý ảnh trong thực tế.

1. My Photo Editor, phần mềm chỉnh sửa ảnhинг

Mục đích của bài này là trình bày cách kết hợp các kiến thức và kỹ năng cần có để thực hiện một ứng dụng xử lý ảnh thông qua việc viết một chương trình có khả năng chỉnh sửa ảnh như phần mềm Photoshop, tuy nhiên xét về mặt thẩm mỹ thì không phải là tuyệt vời. Mô tả cách chỉnh sửa ảnh mà ta có thể dùng để chỉnh sửa ảnh như: chuyển đổi màu sắc, giao diện chỉnh sửa sau:



quản lý chương trình cung cấp, ta chia chương trình làm 3 phần, một phần chuyên về các hàm xử lý ảnh có sẵn trong module và tên là *ImgProcessing*. Phần cho phép ta xem trước khi xử lý là một Dialog có chứa các thanh trượt giúp ta điều chỉnh các tham số xem trước khi quá trình xử lý. Phần này có nhu cầu trong lớp *ImgPreview*. Cuối cùng là phần chính giao diện chính của chương trình là Dialog cần thiết để ta có thể chỉnh sửa các biến và xử lý số liệu trong file *My Photo EditorDlg.h* và *Photo EditorDlg.cpp*. Ta sẽ lần lượt xem qua các phần này.

Phân tích các hàm xử lý chính

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Các hàm xử lý trong ImgProcessing có định nghĩa `void TenHam(cv::Mat &src, cv::Mat& dst, type value)` trong đó, `src` là hình ảnh vào xử lý, `dst` là hình ảnh kết quả và `value` thì tùy thuộc vào cách trả về các kết quả mà có các kiểu và giá trị khác nhau. Ta có thể nhìn header của lớp ImgProcessing như sau:

```
// header file

#pragma once

#include <opencv2\core\core.hpp>

using namespace cv;

class ImgProcessing
{
public:
    ImgProcessing(void);
    ~ImgProcessing(void);
    // Hàm xử lý
    void Grayscale(Mat&, Mat&);
    void ColorSpace(Mat&, Mat&, int);
    void Brightness(Mat&, Mat&, int);
    ...
}
```

Và file cài đặt ImgProcessing.cpp có định nghĩa sau (chỉ viết hàm tăng giảm sáng):

```
// Brightness
void ImgProcessing::Brightness(Mat &src, Mat &dst, int value)
{
    if(src.channels() == 1)
    {
        for(int i = 0; i < src.rows; i++)
            for(int j = 0; j < src.cols; j++)
                dst.at<uchar>(i,j) =
                    saturate_cast<uchar>(src.at<uchar>(i,j) + value);
    }

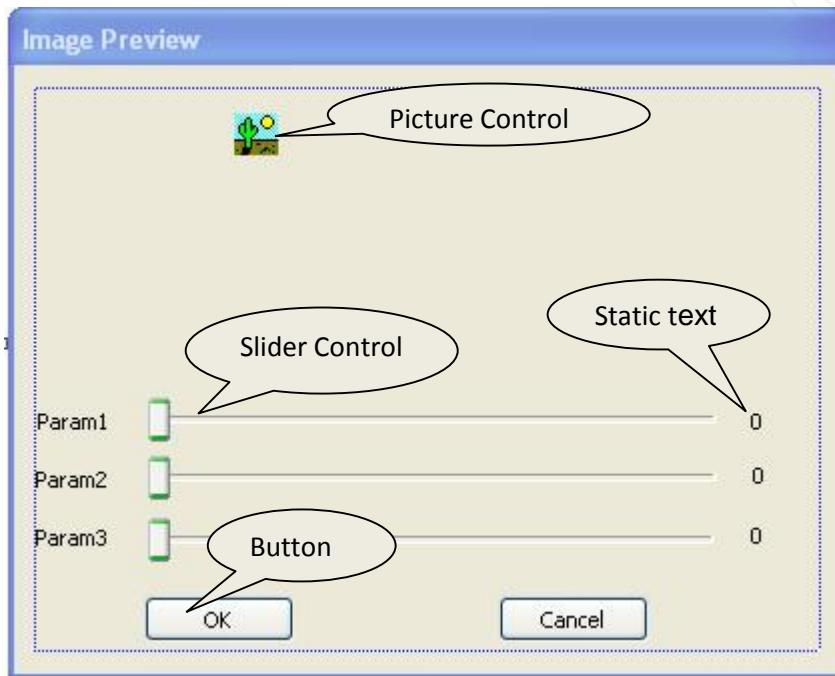
    if(src.channels() == 3)
    {
        for(int i = 0; i < src.rows; i++)
            for(int j = 0; j < src.cols; j++)
                for(int k = 0; k < 3; k++)
                    dst.at<Vec3b>(i,j)[k] =
                        saturate_cast<uchar>(src.at<Vec3b>(i,j)[k] +
                        value);
    }
}
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Các hàm khác có thể cài đặt một cách thông thường như bài trước đã cấp.

Phân xem trực tiếp qua cửa sổ

Phân này nằm trong lớp *ImgPreview* kế thừa lớp *CDialogEx* của MFC. Tô ra 1 phân này, cách nhìn gần nhất là thêm vào project một dialog và thêm 1 phân vào cho dialog đó. Cửa sổ này, ta cần Resource View -> My Photo Editor -> My Photo Editor.rc -> Dialog hãy click chuột phải và chọn Insert Dialog, thì sẽ tìm thấy dialog có dưới đây sau:



Thêm 1 phân vào dialog này ta click chuột phải vào dialog sau đó chọn Add Class, hoặc click MFC Add Class Wizard hiện ra, ta đặt tên cho class là *Class name* là *ImgProcessing* và là *Base class* ta chọn là *CDialogEx*. Nhấn Finish hoàn thành quá trình khởi tạo và ta có hai file mới cần thêm vào, là *ImgProcessing.h* và *ImgProcessing.cpp*. Tiếp theo ta thêm biến cho các biến khi click (Control) bằng cách click chuột phải vào các biến sau đó chọn Add Variable. Các biến có slider tùy chỉnh tham số có tên là *slider1*, *slider2*, *slider3*, các biến là nhãn (Static text) dùng để đặt tên cho slider có tên lần lượt là *l_disp1*, *l_disp2*, *l_disp3* và cùng tên của biến hiến là tên khi kéo trượt slider là *param1*, *param2*, *param3*.

Vì trong chương trình có rất nhiều các hàm xử lý cần nêu ví dụ xem trực tiếp và vì có một hàm xử lý cần nêu m trong dialog xem trực tiếp vì ta phải tự mở ra sẵn 1 phân dialog lên để nghe tiếng và thử các tính năng này, ta chỉ mở m trong preview duy nhất và tùy thuộc vào yêu cầu xử lý của tinh hàm mà ta

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

cho nó mà không cần tốn nhiều thời gian. Do vậy ta sẽ nêu một số cách xử lý trong file *ImgProcessing.h* như sau:

```
enum {m_brightness = 0, m_contrast = 1, m_threshold = 2, m_rotation = 3,
m_size = 4, m_blur = 5, m_noise = 6, m_distort = 7, m_edge = 8};
```

Và khi dialog Image Preview có khởi động, trong hàm *OnInitDialog()* ta sẽ cung cấp vào yêu cầu của hàm *cnx_lý_toracs* thích hợp

```
BOOL ImgPreview::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // Khởi tạo các thanh phan
    this->slider1.SetRange(0, 450);
    this->slider1.SetPos(225);
    this->slider2.SetRange(0, 450);
    this->slider2.SetPos(225);
    this->slider3.SetRange(0, 450);
    this->slider3.SetPos(225);

    this->slider2.EnableWindow(FALSE);
    this->slider3.EnableWindow(FALSE);

    switch(type)
    {
        case m_brightness:
        {
            this->param1.SetWindowText("Brightness");
        }
        break;

        case m_contrast:
        {
            this->param1.SetWindowText("Contrast");
            this->slider1.SetRange(0, 100);
            this->slider1.SetPos(10);
        }
        ...
    }
}
```

Trong đó, *type* là là kiểu xử lý mà chúng ta chèn vào trình chỉnh sửa ngay trong hàm xử lý khác nhau. *type* có thể nêu trong file *ImageProcessing.h* là

```
private:
    int type;
và để gán giá trị thông qua phương thức SetType(int type_):
void ImgPreview::SetType(int type_)
{
    this->type = type_;
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
}

Khi thanh tr  t i u ch nh các thong s thay i v trí, nh ph i c x lý và hi n th tr c quan l ên Picture Control. S ki n thay i v trí thanh tr t c cài t nh sau:

void ImgPreview::OnNMCustomdrawSlider1(NMHDR *pNMHDR, LRESULT *pResult)
{
    LPNMCUSTOMDRAW pNMCD = reinterpret_cast<LPNMCUSTOMDRAW>(pNMHDR);
    switch(type)
    {
        case m_brightness:
        {
            i_param1 = slider1.GetPos() - (int)slider1.GetRangeMax()/2;
            l_disp1.SetWindowTextA(ToString(i_param1));
            process.Brightness(img_display, img_dst, i_param1);
            cv::imshow("Image Preview", img_dst);

        }
        break;

        case m_contrast:
        {
            d_param1 = double(slider1.GetPos() / 10.0);
            l_disp1.SetWindowTextA(ToString(d_param1));
            process.Contrast(img_display, img_dst, d_param1);
            cv::imshow("Image Preview", img_dst);

        }
        break;
        ...
    }
}
```

Trong ó, *img_dst* là m t nh thu nh c a nh g c, c thay i kích th c hi n th phù h p trong khung hình thông qua hàm *ImageDisplay()*

Khi vi c xem tr c hoàn t t, n u ta ng ý v i k t qu ó và b m vào nút OK, m t bi n thành viên *is_ok* s c gán giá tr cho ch ng trình chính bi t r ng ng i dung ā ng ý v i vi c ch nh s a ó, ng c l i n u nút Cancel c b m, bi n này s có giá tr *false*.

```
void ImgPreview::OnBnClickedButton1()
{
    // Button OK
    this->is_ok = true;
    ImgPreview::OnCancel();
}
```

Phân chia trình chính

Phân này chia giao diện thành các module trong quá trình xử lý ảnh. Nó bao gồm các thao tác m file ảnh, hiển thị các hàm xử lý ảnh đã nêu trên, hiển thị các cửa sổ view khác nhau và lưu file ảnh sau khi hoàn tất xử lý.

Tại thời điểm giao diện có menu, button và các cửa sổ hiển thị có tạo ra một Picture Control... như hình 1. Các Control khác trong MFC là khá quen thuộc, ví dụ hiển thị menu, thêm vào dialog đầu tiên ta thiết kế có menu bằng cách click chuột phải vào Project sau đó chọn Add -> Resource-> Menu. Sau khi đã hoàn thành hiển thị menu ta hãy vào dialog chính, mở Properties của dialog này chọn Menu và chọn ID của menu mà ta đặt trước, giờ là IDR_MENU1



Khi click chuột vào menu, ta hãy quay lại thiết kế menu, click chuột phải vào menu cần xử lý sau đó chọn Add Event Handler. Sau khi click chuột vào menu Open có dòng như sau:

```
void CMYPhotoEditorDlg::OnUpdateFileOpen1(CCmdUI *pCmdUI)
{
    // Open Image

    static CString Filter=_T("image files (*.bmp; *.jpg) |*.bmp;*.jpg|All
Files     (*.*)|.*||");
    CFileDialog Load(TRUE, _T("*.bitmap"), NULL,
OFN_FILEMUSTEXIST|OFN_PATHMUSTEXIST|OFN_HIDEREADONLY,Filter,NULL);
    Load.m_ofn.lpstrTitle= _T("Load Image");
    if (Load.DoModal() == IDOK)
    {
        img_path = Load.GetPathName();
        img_name = Load.GetFileName();
        src = cv::imread(img_path);
        ...
    }
    ...
}
```

Sau khi nhấn nút Open có thể chỉnh nghĩa một cách thông thường.

Sau khi file ảnh được mở, ta có thể tiến hành thực hiện các phép xử lý ảnh thông qua một menu vừa thiết kế trên. Ví dụ các phép xử lý cho kỹ thuật số mà không

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

còn phải xem trước khi tách (như chuyển sang màu xám, làm ngược nhau...) thì sẽ kích nút click chuột vào menu Options, xử lý có sẵn sau:

```
void CMyPhotoEditorDlg::OnUpdateAdjustmentsInvert(CCmdUI *pCmdUI)
{
    // Invert Image
    process.Invert(src, src);
    ImageDisplay(src);
    ...
}
```

Điều này trong menu Options xem trước khi tách sẽ lý như sau: khi ta click nút c tham số dialog preview, nó sẽ tự động mở dialog preview, truy cập tham số cho dialog này và nhận tham số để xử lý. Số lượng các hành động menu Options có sẵn như sau:

```
void CMyPhotoEditorDlg::OnUpdateAdjustmentsThreshold(CCmdUI *pCmdUI)
{
    // Threshold
    ImgPreview dlg;
    dlg.SetType(m_threshold);
    dlg.src = src;
    dlg.DoModal();
    if(dlg.is_ok)
    {
        process.Threshold(src, src, dlg.i_param1);
        ImageDisplay(src);
    }
}
```

Trong đoạn code trên, ta truy cập hai tham số cho lớp *ImgPreview* là *m_threshold* cho phép này biết khi nào xem lý là gì và *src* là hình ảnh cần xem trước trong quá trình xử lý. Hàm *ImageDisplay()* có tác dụng tùy chỉnh kích thước hiển thị trong phù hợp trong khung hình sau khi hiển thị lên. *Process* là một biến của class *ImgProcessing* cài khai báo trong file *My Photo EditorDlg.h*.

Bên cạnh quá trình xử lý, ta thiêm thêm các chức năng khác như thông tin chi tiết như ang xem lý, ở là các phép xem thông tin chi tiết về kích thước (kích thước chiều dài, rộng, ...), các histogram màu (màu caciu kênh) và các phép xem lịch sử ảnh. Các chức năng này thông qua menu view cho phép nó hoạt động hay không hoạt động và trả về những biến kiểm tra xem chức năng này có kích hoạt hay không trong các hàm xử lý. Nếu nó kích hoạt (có checkbox menu view) thì sau mỗi hàm xử lý ta phải phân thông tin cho nó, nếu không thì bỏ qua bước này, sẽ tốn kém thời gian xử lý.

```
void CMyPhotoEditorDlg::OnUpdateViewHistogram(CCmdUI *pCmdUI)
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

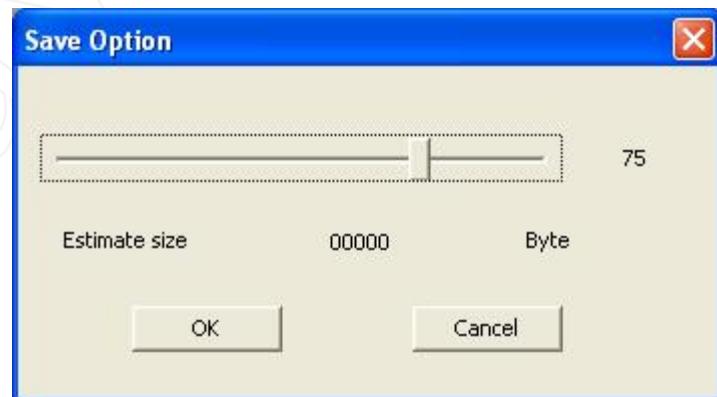
```
{  
    // View histogram  
    is_histogram = !is_histogram;  
    CMenu *m_histogram = GetMenu();  
    m_histogram->GetSubMenu(3);  
    if(is_histogram)  
    {  
        m_histogram->CheckMenuItem(ID_VIEW_HISTOGRAM, MF_CHECKED);  
        UpdateHistogram(0);  
    }  
    else  
    {  
        m_histogram->CheckMenuItem(ID_VIEW_HISTOGRAM, MF_UNCHECKED);  
        cv::imshow("Histogram", hist_init);  
    }  
}  
}
```

Và trong các hàm xử lý ta kiểm tra biến *is_histogram* để update cách view

```
void CMyPhotoEditorDlg::OnUpdateAdjustmentsInvert(CCmdUI *pCmdUI)  
{  
    // Invert Image  
    process.Invert(src, src);  
    ImageDisplay(src);  
    if(is_histogram) UpdateHistogram(0);  
    if(is_history) history_list.AddString("Invert Image");  
}
```

Trong đó hàm *UpdateHistogram* sẽ tính toán histogram và hiển thị histogram của ảnh lên cửa sổ, *history_list* là một Edit Control hiển thị lịch sử các bước thao tác.

Khi quá trình chỉnh sửa hoàn tất ta có thể lưu hình thông qua nút *Save* hoặc menu *Save*. Khi lưu, ta cho phép người dùng chọn cách lưu thông qua tệp nén nh. Tuy nhiên càng cao thì tệp nén có dung lượng càng thấp, tuy nhiên thông tin có thể bị mất đi. Ngược lại nếu tệp không chi tiết tuy nhiên dung lượng lớn. Ta thiết kế một dialog *SaveOption* và xử lý số lượng byte cần lưu.



Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Một phần của ứng dụng quản lý ảnh trong quá trình xử lý có là xác định tính hỗ trợ các menu. Theo đó, để tránh gặp lỗi phát sinh thì menu chỉ có phép sử dụng khi đã có các mục con có sẵn. Khi nhấp chuột vào chương trình thì toàn bộ các menu sẽ có menu và nút Open và có phép hoạt động nếu không khi click vào bất kỳ menu nào hết ngay lập tức do nó là nhữngh. Khi đang làm việc với hình xám (grayscale) thì menu chuyển sang các không gian màu khác (HSV chương trình không muốn chuyển sang grayscale) và thoát ra ngoài ý muốn. Ta xác định tính hỗ trợ các menu thông qua các biến và hàm *ValidateMenu()*. Ví dụ biến *is_load* kiểm tra xem nhấp chuột hay chưa, biến *is_gray* kiểm tra có phải là hình xám hay không... các biến này sẽ được cập nhật sau các hàm tương ứng của chúng :

```
void CMyPhotoEditorDlg::OnUpdateFileOpen1(CCmdUI *pCmdUI)
{
    // Open Image
    ...
    if (Load.DoModal() == IDOK)
    {
        ...
        is_load = true;
        ...
    }
}
```

Và hàm *ValidateMenu()* có dưới đây sau:

```
void CMyPhotoEditorDlg::ValidateMenu()
{
    // Kiểm tra tính hợp lệ của menu trong từng trường hợp
    CMenu *m_file = GetMenu();
    m_file->GetSubMenu(0);

    CMenu *m_image = GetMenu();
    m_image->GetSubMenu(1);

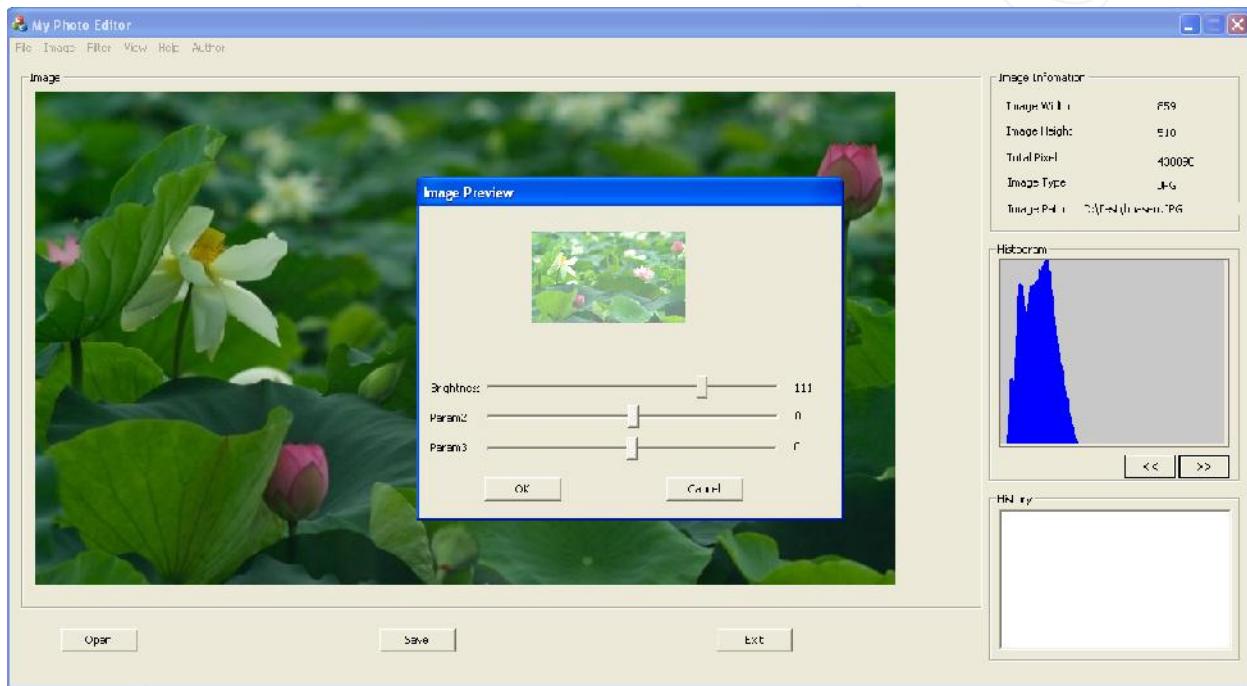
    ...
    if(!is_load)
    {
        m_file->EnableMenuItem(ID_FILE_SAVE1, MF_DISABLED|MF_GRAYED);
        m_image->EnableMenuItem(ID_MODE_GRAYSCALE, MF_DISABLED|MF_GRAYED);
        save_btn.EnableWindow(false);
        ...
    }
    else
    {
        m_file->EnableMenuItem(ID_FILE_SAVE1, MF_ENABLED);
        m_image->EnableMenuItem(ID_MODE_GRAYSCALE, MF_ENABLED);
        ...
    }
    if(is_gray)
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
{  
    m_image->EnableMenuItem(ID_MODE_HSV, MF_DISABLED|MF_GRAYED);  
    m_image->EnableMenuItem(ID_MODE_YCrCb, MF_DISABLED|MF_GRAYED);  
    ...  
}  
}
```

Hàm *EnableMenuItem()* là hàm cho phép mở menu nhanh thông qua tham số *MF_DISABLED* hoặc *MF_ENABLED*.

Hình sau là kết quả của quá trình xử lý khi tăng độ sáng camera.



2. Nhận diện biển xe

Bài toán nhận diện biển xe có nhiều ý nghĩa trong thực tế, nó giúp việc giám sát, quản lý, thống kê... các phương tiện một cách dễ dàng, tiện lợi và nhanh chóng. Một số ứng dụng nhận diện biển số khai trong thực tế như: ứng dụng trong quản lý bãi xe thông minh, ứng dụng thu phí các trạm thu phí, ứng dụng phát hiện lỗi vi phạm giao thông một cách tự động... Trong phần này ta nghiên cứu về một kỹ thuật bài toán nhận diện biển xe, vì thế ng trình demo và các bước cần thiết để áp dụng bài toán vào ứng dụng thực tế.

Giờ ta đang xây dựng bài toán nhận diện biển xe ô tô, và đưa vào là một nhu cầu biển xe và العرا là một chuỗi ký tự của biển số cần nhận diện. Nếu quan sát kỹ sẽ thấy có thể dễ dàng nhận biết các chữ số biển, tuy nhiên với máy tính, nó là một iu không dễ dàng gì, nó rất dễ bị nhầm lẫn với các hình khác xung quanh tảng

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

t, b i các i u ki n th i ti t, góc ... h n ch b t c nh ng khó kh n này, nhi u h th ng nh n d ng trong th c t th ng gi i h n các i u ki n, ch ng h n nh camera thu nh c c nh m t v trí, xe c i vào m t khe h p nh t nh...

Có nhi u các khác nhau th c hi n bài toán nh n d ng này, các ph n m m th ng m i hoàn ch nh, nó là s k t h p và t i u c a khá nhi u thu t toán ph c t p, trong bài này, ta i theo h ng ti p c n chia bài toán thành hai bài toán nh : phát hi n bi n s xe, cách ly kí t và nh n d ng các kí t .



Phát hi n vùng ch a bi n s xe và cách ly kí t .

Vì bi n s xe có nh ng c tr ng c b n c quy nh b i các c quan ch c n ng nê ta có th d a vào c tr ng này phân bi t v i các i t ng khác. Theo quy nh c a b công an, bi n s xe ng tr c c a các lo i xe dân d ng là m t hình ch nh t, có kích th c 470x110 (mm), phông n n màu tr ng và các kí t ch cái in hoa màu en. Các kí t ch s bao g m t 0 t i 9 và các kí t ch s bao g m A, B, C, D, E, F, G, H, K, L ,M , N, P, S, T, U, V, X, Y, Z (20 kí t). Ta s d a vào các c tr ng hình h c này trích ch n ra vùng ch a bi n s xe, các b c th c hi n nh sau:

B c 1: Load nh, ti n x lý nh (kh nhi u, làm m m ...)

B c 2: Chuy n nh ban u thành nh xám r i nh phân hóa nh ó. nh nh phân thu c k t qu t t và không b ph thu c vào các i u ki n ánh sáng khác nhau, ta s d ng ph ng pháp nh phân hóa v i ng ng (adaptive threshold) nh ã nói trên.

B c 3: Tìm các ng bao quanh i t ng. Sau khi nh phân nh, các i t ng s là các kh i hình en trên n n tr ng, v i m i i t ng ta luôn v c m t ng biên

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

bao quanh i t ng ó. tránh trình tr ng các i t ng không t o ra c các ng biên khép kín do các v t d n n t ho c nhi u gây ra, tr c khi tìm biên ta nên làm m m ng biên b ng các phép giãn n (ho c co) nh ã nói trong bài tr c.

B c 4: Xác nh hình ch nh t bao quanh các ng bao quanh ã tìm c t b c 3. ng bao quanh ã tìm c b c 3 là m t chu i các i m biên n i li n c a i t ng, d a vào t a c a các i m biên này ta s xác nh c hình ch nh t ngo i ti p bao quanh i t ng.

B c 5: Tìm ra các hình ch nh t có kh n ng là vùng ch a bi n s , n u hình ch nh t thu c b c 4 là vùng ch a bi n s thì nó ph i th a mǎn ít nh t c m t s i u ki n sau:

- o T 1 chi u dài/ r ng ph i x p x 4.3. Trong cài t th c t ta có th cho t l này dao ng trong kho ng [4.0, 4.5].
- o S i t ng th a mǎn là m t kí t bi n s trong vùng hình ch nh t ph i là m t s l n h n m t ng ng nào ó. V i bi n s xe 4 ch s , s kí t này là 7, v i bi n s xe m i v i 5 kí t s , s kí t này là 8, c ng có m t s bi n s xe có nhi u h n 8 kí t do có 2 kí t ch cái ... xác nh m t i t ng là kí t hay không, ta c ng s d a vào c i m hình h c c a kí t ó nh t 1 chi u dài/r ng i t ng, t 1 chi u cao, dài c a i t ng so v i t 1 chi u cao, dài c a hình ch nh t ang c xem xét là bi n s , t 1 pixel en/tr ng c a d i t ng ... N u xác nh ó là m t kí t c a bi n s xe ta c ng s ng th i l u n ó l i, ó chính là cách ta cách ly i t ng trong bi n s .
- o Ngoài ra, tùy thu c vào i u ki n c a bài toán ta có th c nh thêm m t s c tính ch c ch n h n vùng ch a bi n s , ch ng h n nh kích th c c a hình ch nh t không c v t quá m t n a kích th c c a nh u vào, t 1 pixel en/tr ng trong hình ch nh t ph i n m trong m t ng ng nào ó ...

Sau khi ã xác nh c vùng có kh n ng là bi n s , ta s c t vùng hình ó, ng th i l y ra các kí t trong vùng ó l u vào m t m ng th c hi n vi c nh n d ng.

Nh n d ng kí t

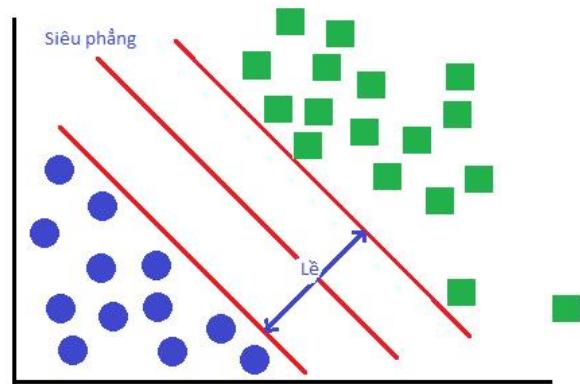
Các kí t sau khi c cách ly là nh ng kí t n l trong m t khung hình ch nh t có kích th c nh t nh. nh n d ng c kí t này ta có th s d ng r t nhi u ph ng pháp khác nhau, t n g i n nh ph ng pháp s d ng t ng quan chéo (cross correlation) cho n nh ng ph ng pháp s d ng các mô hình máy h c (machine learning) nh m ng Neuron nhân t o, SVM ...

i v i các ph ng pháp s d ng máy h c, ta c n s u t m m t l ng m u các kí t nh t nh, t vài tr m cho t i hàng nghìn m u r i a vào các b hu n luy n, k t qu hu n

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

luyện này sử dụng những dữ liệu các mẫu. chính xác cách quy trình dữ liệu nói chung của pháp này tùy thuộc vào cách tạo mô hình, khai thác dữ liệu luyện, thời gian tính toán. Trong phần này ta sử dụng pháp SVM như sau.

Phương pháp SVM. (Chi tiết về phương pháp này bạn có thể tham khảo trong các tài liệu chuyên ngành khác). SVM (Support Vector Machine) là một mô hình máy học giám sát sử dụng trong việc phân tích, phân lớp dữ liệu vào các siêu phẳng. Giả sử ta có một tập dữ liệu hai chiều không hình bên, khi đó ta có thể phân lớp dữ liệu này thành hai phần nhỏ nhất. Siêu phẳng trong một không gian 3 chiều là một mặt phẳng, trong không gian 3 chiều là một mặt phẳng và tách ngang trong không gian n chiều là một không gian n-1 chiều. Trong trường hợp dữ liệu không tuyến tính, ta cần áp dụng dữ liệu lên một không gian có số chiều lớn hơn để phân loại dữ liệu, như vậy là cần phải tìm siêu phẳng sao cho khoảng cách từ các biên cách đến là lớn nhất. Hiểu một cách đơn giản, phương pháp này như sau: cho một tập các mẫu luyện, ví dụ một cung vào một nhãn, quá trình luyện SVM sẽ xây dựng một mô hình cho phép đoán một mẫu dữ liệu khác thuộc nhãn nào, từ phân loại dữ liệu đó thu được vào lớp nào.



Quay lại bài toán nhận dạng kí tự biển số xe ta cần xét, làm sao nhận dạng các kí tự này dựa trên mô hình SVM?

Trong cách này SVM là một máy phân loại dữ liệu, nhưng nó ta cần phải có dữ liệu, dữ liệu để xác định các kí tự mà ta cần nhận dạng. Đây chính là các đặc trưng trong nhãn của kí tự. Giả sử ta cần phân loại 301 pixel dữ liệu (tổng cộng 30 ký tự trong biển số xe), ví dụ pixel, ta tính toán một vector đặc trưng (10 màu), và mỗi vector đặc trưng có giá trị các đặc trưng trong một nh. Khi đó ta sẽ vào bộ huấn luyện SVM toàn bộ dữ liệu này, sau đó ta cần xem xét xem dữ liệu này (từ vector đặc trưng này) thuộc vào lớp nào trong số những lớp mà nó đã được huấn luyện.

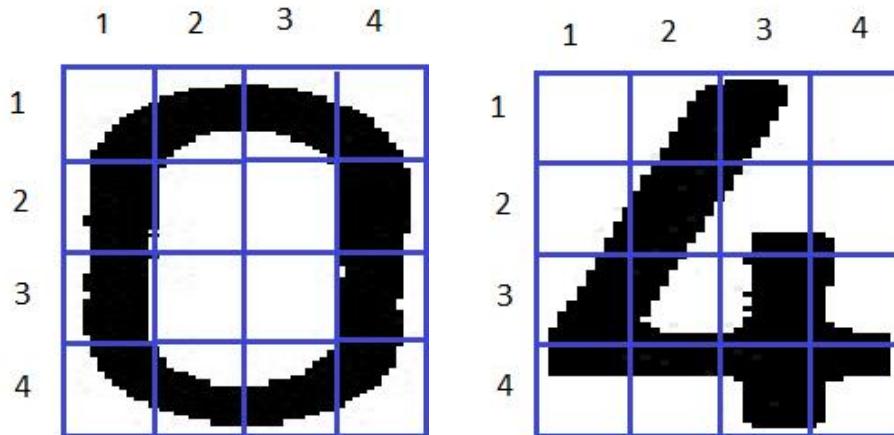
Tính toán đặc trưng trong nh.

Đặc trưng trong nh là những đặc điểm riêng biệt giúp phân biệt nh này với nh khác. Vì xem xét nh nào là các đặc trưng trong nh là một việc không có quy tắc, nó phụ thuộc vào cách nghiên cứu, cài đặt thuật toán học và vấn đề nghiên cứu. Để giải quyết vấn đề này, chúng ta có thể áp dụng phương pháp tính toán đặc trưng trong nh hay nh, hiệu quả hơn.

Giả sử ta có hai kí tự nh hình bên, nhìn bằng mắt thường ta có thể dễ dàng phân biệt hai kí tự 0 và 4, tuy nhiên làm sao máy tính phân biệt hai kí tự này? Đây gi

Ta sẽ xem hai ký tự này và cùng nhìn kích thước, chia nhau thành 16 ô nhau khác nhau sau:

0 4



Ta nhận thấy nếu tính tổng các pixel đen trong các ô của hai ký tự thì số 0 và số 4 có thể phân biệt được vào các ô (1,1), (1,4), (2,2), (3,3) ... tinh ngô, tổng số các i m nh en là khác nhau hoàn toàn. Tính toán số i m nh en của 16 ô vuông này ta có thể thu c 16 c tr ng c a m t nh, 16 c tr ng này phân bi t ký t 0 và 4. Tuy nhiên, với 30 ký tự ta cần phải tính toán nhiều hơn các c tr ng, các c tr ng không nh thi t ph i là 0 (t c không có i m nh en nào) hoặc 1(t c là toàn s i m nh en trong ô) mà có thể là m t t l t ng i nào ó. T 16 c tr ng c b n trên, ta k t h p chúng l i t o ra nh ng c tr ng khác, ch ng h n nh l y t ng các c tr ng trên ng chéo (1,1) + (2,2) + (3,3) + (4,4) hoặc t ng các c tr ng xung quanh ng biên c a nh ... s c tr ng càng l n thì vi c phân lo i các l p càng ít b sai, có nghĩa là xác su t nh n d ng càng l n.

Cài đặt chương trình Nhận diện biển xe ô tô

Ta tạo một chương trình dựa trên Dialog c a MFC, tên project là LPR (License Plate Recognition). V c b n có thể chia chương trình thành 3 phần chính sau: phần chia giao diện chính c a chương trình (c nh nghĩa trong file *LPRDlg.h* và *LPRDlg.cpp*), phần chia các hàm chính cho việc phát hiện, nhận diện biển xe và không liên quan t i giao diện (c nh nghĩa trong file *LprCore.h* và *LprCore.cpp*) và phần chia m t công c giúp cho ta có thể hu n luy n c mô hình SVM m t cách tùy ý (c nh nghĩa trong file *TrainSVM.h* và *TrainSVM.cpp*). Ngoài ra còn có m t s phần nh giúp cho việc lập trình c d dàng h n nh ph n chuy n i các biến dữ liệu trong MFC ch ng h n (*unity_conversion.h*).

Hu n luy n mô hình SVM

Tạo c mô hình SVM ph c v cho việc nhận diện ký tự sau này, ta cần hu n luy n và lưu mô hình sau khi hu n luy n.

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Chu n b c s d li u. Ta c n chu n b c s d li u là t p h p c a các kí t trong bi n s xe. Có 30 kí t th ng g p trong bi n s xe, do ó ta c n phân lo i 30 l p này, trong tr ng h p ây gi s v im il p, t c là m i t t ta có 10 nh, ta s 1 u các nh này vào các folder, tên các folder c t tên theo các kí t , ch ng h n nh folder 0 ch a 10 nh c a kí t 0, folder 1 ch a 10 nh c a kí t 1, ... folder 30 ch a 10 nh c a kí t Z. ta c n ánh tên folder theo s t t , vì s th t c ng chính là nhän t ng ng a vào vi c nh n d ng. Ta s tính toán c tr ng c a t ng kí t và l u t t c các c tr ng này vào m t ma tr n ph c v cho vi c hu n luy n. Hàm tính toán c tr ng trong m t nh s d a trên ý t ng t ng các i m en trong m t khung hình, tuy nhiên nó c chu n hóa b ng cách chia cho t ng t t c các i m nh en c a kí t .

```
vector<float> calculate_feature(Mat src)
{
    Mat img;
    if(src.channels() == 3)
        cvtColor(src, img, CV_BGR2GRAY);
    threshold(img, img, 100, 255, CV_THRESH_BINARY);

    vector<float> r;
    resize(img, img, Size(40, 40));
    int h = img.rows/4;
    int w = img.cols/4;
    int S = count_pixel(img);
    int T = img.cols * img.rows;
    for(int i = 0; i < img.rows; i += h)
    {
        for(int j = 0; j < img.cols; j += w)
        {
            Mat cell = img(Rect(i,j, h , w));
            int s = count_pixel(cell);
            float f = (float)s/S;
            r.push_back(f);
        }
    }

    for(int i = 0; i < 16; i+= 4)
    {
        float f = r[i] + r[i+1] + r[i+2] + r[i+3];
        r.push_back(f);
    }

    for(int i = 0; i < 4; ++i)
    {
        float f = r[i] + r[i+4] + r[i+8] + r[i+ 12];
        r.push_back(f);
    }

    r.push_back(r[0] + r[5] + r[10] + r[15]);
}
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
r.push_back(r[3] + r[6] + r[9] + r[12]);
...
return r; //32 dac trung
}
```

Trong ó hàm *count_pixel* là hàm tính toán số pixel đen trong một nh

```
int count_pixel(Mat img, bool black_pixel = true)
{
    int black = 0;
    int white = 0;
    for(int i = 0; i < img.rows; ++i)
        for(int j = 0; j < img.cols; ++j)
    {
        if(img.at<uchar>(i,j) == 0)
            black++;
        else
            white++;
    }
    if(black_pixel)
        return black;
    else
        return white;
}
```

o n code sau sẽ huấn luyện một mô hình svm, v i u vào là một folder chứa dữ liệu huấn luyện nh á nói trên (folder này chứa 30 folder con, mỗi folder con chứa 10 kí tự m ú)

```
const int number_of_class = 30;
const int number_of_sample = 10;
const int number_of_feature = 32;

CvSVMParams params;
params.svm_type     = CvSVM::C_SVC;
params.kernel_type = CvSVM::RBF;
params.gamma = 0.5;
params.C = 16;
params.term_crit   = cvTermCriteria(CV_TERMCRIT_ITER, 100, 1e-6);
SVM svm;
Mat data = Mat(number_of_sample * number_of_class, number_of_feature,
CV_32FC1);
Mat label = Mat(number_of_sample * number_of_class, 1, CV_32FC1);
int index = 0;

vector<string> folders = list_folder("D:/Data");
for(size_t i = 0; i < folders.size(); ++i)
{
    cout<<i<<"..."<<endl;
    vector<string> files = list_file(folders.at(i));
```

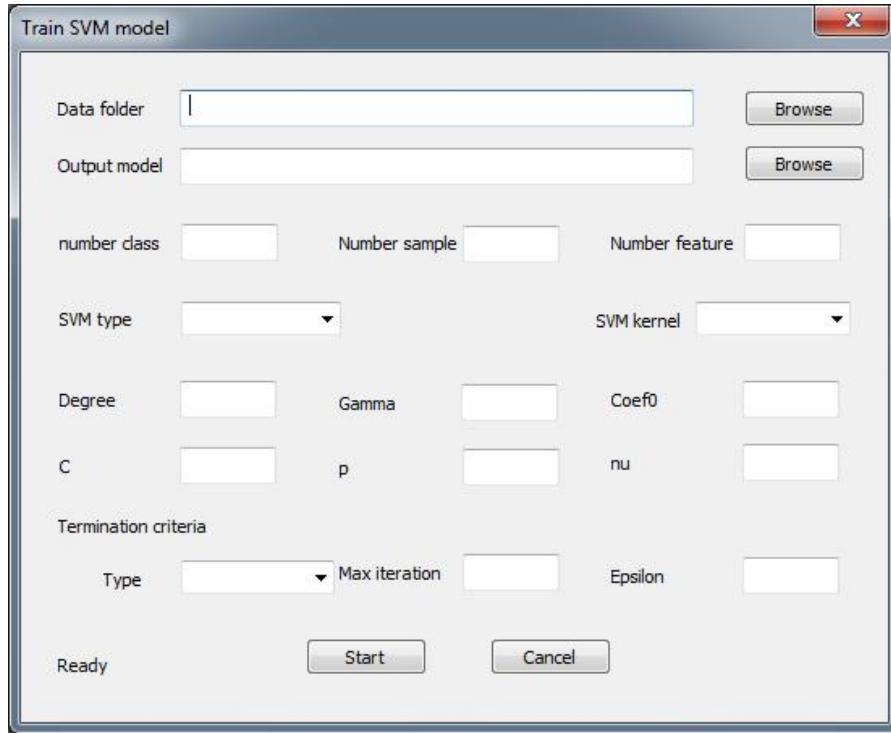
Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
string folder_path = folders.at(i);
string label_folder = folder_path.substr(folder_path.length() - 1);
for(size_t j = 0; j < files.size(); ++j)
{
    src = imread(files.at(j));
    if(src.empty()) continue;
    vector<float> feature = calculate_feature(src);
    if(feature.size() < number_of_feature)
    {
        cout<<"error " << endl;
        continue;
    }

    for(size_t t = 0; t < feature.size(); ++t)
        data.at<float>(index, t) = feature.at(t);
    label.at<float>(index, 0) = i;
    index++;
}
svm.train_auto(data, label, Mat(), Mat(), params);
svm.save("E:/svm.txt");
```

Trong đoạn code trên, ta lần lượt qua từng folder con trong thư mục *D:/Data* bằng hàm *list_folder* (hàm này ta viết ở trên header *dirent.h*, và nó có ích li kê danh sách các folder trong một thư mục), sau đó ta tính toán từng các đặc trưng của ảnh trong các folder này và lưu vào ma trận *data*. Ma trận label chứa nhãn của các ảnh trong folder, nó có tên là tên của các folder trong thư mục. Hàm *train_auto* sẽ chỉ định cách huấn luyện mà không cần cách tách dữ liệu và tách các thông số mô hình. Đây là cách thông thường để huấn luyện SVM mà ta có thể áp dụng. Công cụ cài đặt trong chương trình là một lớp *TrainSVM* kế thừa từ lớp *CDialog* để dễ dàng hơn.

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



Phát hiện và nhận diện biển số xe

Ta cài đặt module phát hiện và nhận diện biển số xe theo nhu cầu của bài toán nói trên, và đặt tên cho lớp này *LprCore*. Ta sẽ cài đặt lớp này và giao diện sao cho việc sử dụng nó là dễ dàng như tệp giao diện (interface) của lớp này có thể như sau:

```
// lprcore.h header
#pragma once
#include <opencv2/core/core.hpp>
#include <opencv2/ml/ml.hpp>

using namespace std;
using namespace cv;

class LprCore
{
public:
    LprCore(void);
    ~LprCore(void);
    void set_image(Mat);
    void set_svm_model(string);
    void do_process();
    vector<string> get_text_recognition();
    vector<Mat> get_plate_detection();
    vector<Mat> get_character_isolation();
    vector<vector<Mat>> get_character();
    vector<double> get_process_time();
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
    void clear();  
private:  
    bool done;  
    bool ready;  
    SVM svm;  
    Mat image;  
    vector<Mat> plates;  
    vector<Mat> draw_character;  
    vector<vector<Mat>> characters;  
    vector<string> text_recognition;  
    vector<double> process_time;  
    char character_recognition(Mat);  
    bool plate_detection();  
    bool plate_recognition();  
};
```

các l p khác khi s d ng giao di n này, ch vi c a nh u vào, a mô hình hình svm à c hu n luy n, g i hàm `do_process` và sau ó có th l y ra các k t qu nh bi n s nh n d ng c, k t qu nh n d ng c, th i gian c a các quá trình ...

B n ch t c a hàm `do_process` s g i hàm `plate_detection` và hàm `plate_recognition`. N u g i hai hàm này thành công, có ngh a là quá trình th c hi n ã thành công và ta gán cho bi n tr ng thái là true

```
void LprCore::do_process()  
{  
    if(this->plate_detection() && this->plate_recognition())  
        done = true;  
}  
Các hàm l y k t qu s tr v k t qu t ng ng nh bi n s , chu i kí t nh n d ng c  
...  
vector<string> LprCore::get_text_recognition()  
{  
    if(done)  
        return this->text_recognition;  
}  
  
vector<Mat> LprCore::get_plate_detection()  
{  
    if(done)  
        return this->plates;  
}
```

Ta s xét hai hàm cài t hàm chính c a l p ó là hàm `bool plate_detection()` và hàm `char character_recognition(Mat)`. Hàm `bool plate_recognition()` th c ch t là vi c là vi c g i hàm `char character_recognition(Mat)` cho m t chu i các nh kí t trong bi n s .

Hàm `bool plate_detection()`, tr v k t qu `true` n u phát hi n c bi n s ng th i l u các bi n s phát hi n c vào vector `vector<Mat> plates`:

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Trong code trên, ta thấy có các phép biến đổi thông thường thu được từ phân tích quan sát, sau đó áp dụng phép tìm khung bao quanh (contour), ví dụ khung bao, ta vẽ khung bao quanh và xem xét các đặc điểm của khung bao:

```
...
cvtColor(image, gray, CV_BGR2GRAY);
adaptiveThreshold(gray, binary, 255, CV_ADAPTIVE_THRESH_GAUSSIAN_C,
CV_THRESH_BINARY, 55, 5);

...
findContours(binary, contours, hierarchy, CV_RETR_TREE,
             CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
if(contours.size() <= 0) return false;
for(size_t i = 0; i < contours.size(); ++i)
{
    Rect r = boundingRect(contours.at(i));
    if(r.width > image.cols/2 || r.height > image.cols/2 || r.width < 120
       || r.height < 20 || (double)r.width/r.height > 4.5
       || (double)r.width/r.height < 3.5)

        continue;
    ...
}
```

Nếu các khung bao quanh không thỏa mãn điều kiện là biển số qua không xét tiếp mà chuyển sang cách khác. Trong trường hợp này, ta tiếp tục vòng lặp bằng cách tìm các khung bao quanh tiếp theo trong khung bao quanh trước.

```
...
Mat sub_image = image(r);
vector<Rect> r_characters;
for(size_t j = 0; j < sub_contours.size(); ++j)
{
    Rect sub_r = boundingRect(sub_contours.at(j));
    if(sub_r.height > r.height/2 && sub_r.width < r.width/8 && sub_r.width
       > 5 && r.width > 15 && sub_r.x > 5)
    {
        Mat cj = _plate(sub_r);
        double ratio = (double)countPixel(cj)/(cj.cols*cj.rows);
        if(ratio > 0.2 && ratio < 0.7)
        {
            r_characters.push_back(sub_r);
            rectangle(sub_image, sub_r, Scalar(0,0,255), 2, 8, 0);
        }
    }
}
...
```

Trong đoạn code trên, thay đổi các khung bao quanh tiếp theo là một khung biển số, ngoài ra nó sẽ loại bỏ khung quan dài, riêng cách tiếp theo so với biển số còn so sánh tần số pixel trên từng số pixel của khung và ta giả sử rằng nó là một khung biển số thì

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

nó n m trong kho ng 0,2 n 0,7. Sau b c này, n u s l ng *r_characters* l n m t ng ng nào ó (th ng là > 7) thì ta s công nh n nó r là m t vùng ch a bi n s , các kí t c t ra c sau ó c n ph i s p x p l i theo th t t trái sang ph i cho nó t ng ng v i th t các ch cái trong bi n s tr c khi áp d ng cho vi c nh n d ng.

Hàm `char character_recognition(Mat img_character)` tr v k t qu là m t kí t ki u *char* v i m t nh u vào t ng ng. th c hi n vi c nh n d ng, u tiên ta c n tính toán các c tr ng c a nh u vào *img_character*. Vi c tính toán các c tr ng s cho ta m t vector ch a các c tr ng c a nh ó, ta s dùng hàm *predict* c a d i t ng *svm* xem xem vector ó thu c v l p nào. K t qu tr v c a hàm *predict* là m t s th c t ng ng v i nhän mà ta ã hu n luy n, có 30 nhän t ng ng v i các s t 0 – 9 và t A – Z. Chú ý là không ph i t t c các ch cái u c s d ng, do v y sau k t qu thu c t hàm *predict* c n ph i c chuy n i sang các kí t t ng ng. Chú ý là tr c khi s d ng hàm predict d oán ta c n ph i load mô hình SVM c hu n luy n t b c trên.

```
void LprCore::set_svm_model(string file_name)
{
    this->svm.load(file_name.c_str());
    ready = true;
}
```

Hàm cài t cho nh n d ng kí t nh sau:

```
char LprCore::character_recognition(Mat img_character)
{
    char c = '*'; // truong hop khong cho ket qua tra ve *
    if(img_character.empty()) return c; // neu anh rong
    if(!ready) return c; // neu chua load mo hinh svm

    vector<float> feature = calculate_feature(img_character);
    Mat m = Mat(number_of_feature, 1, CV_32FC1);
    for(size_t i = 0; i < feature.size(); ++i)
        m.at<float>(i, 0) = feature[i];
    float r = this->svm.predict(m); // du doan mau moi
    int ri = (int)r;
    if(ri >= 0 && ri <= 9)
        c = (char)(ri + 48); //ma ascii 0 = 48
    if(ri >= 10 && ri < 18)
        c = (char)(ri + 55); //ma accii A = 65, --> tu A-H
    if(ri >= 18 && ri < 22)
        c = (char)(ri + 55 + 2); //K-N, bo I,J
    if(ri == 22) c = 'P';
    if(ri == 23) c = 'S';
    if(ri >= 24 && ri < 27)
        c = (char)(ri + 60); //T-V,
    if(ri >= 27 && ri < 30)
        c = (char)(ri + 61); //X-Z
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
    return c;
}

Và cuối cùng, toàn bộ kí tự trong biển số sẽ được nhận dạng qua hàm bool
plate_recognition()

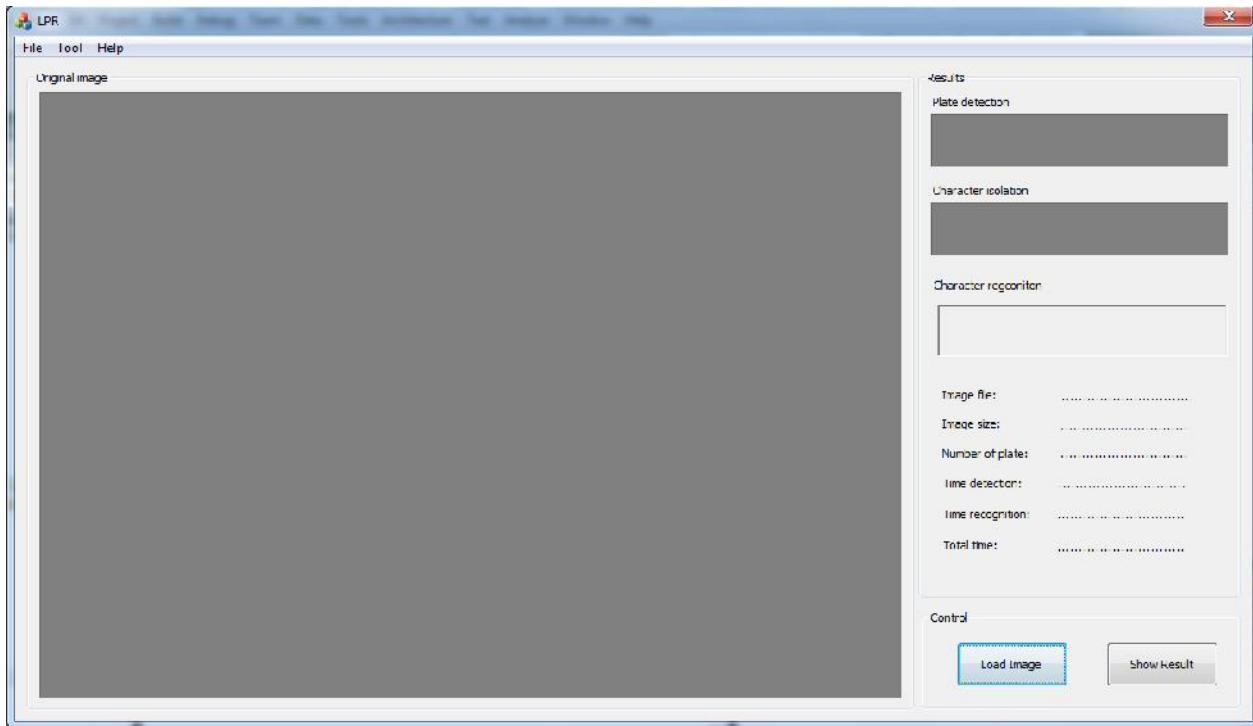
bool LprCore::plate_recognition()
{
    if(plates.size() <= 0) return false;
    double t = (double)cvGetTickCount();
    for(size_t i = 0; i < characters.size(); ++i)
    {
        string result;
        for(size_t j = 0; j < characters.at(i).size(); ++j)
        {
            char cs = character_recognition(characters.at(i).at(j));
            result.push_back(cs);
        }

        this->text_recognition.push_back(result);
    }
    t = (double)cvGetTickCount() - t;
    t = (double)t/(cvGetTickFrequency()*1000.); //convert to second
    process_time.push_back(t);
    return true;
}
```

Giao diện chính

Tại đây giao diện chính bao gồm các picture control hiển thị, các button, menu và các label hiển thị sau:

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



Các button, label ... c khai báo trong header *LPRDlg.h* và các hàm xử lý sẽ n

cài đặt trong file *LPRDlg.cpp*

```
// LPRDlg.h : header file
#pragma once
#include <opencv2/core/core.hpp>
#include "LprCore.h"
#include "afxwin.h"

// CLPRDlg dialog
class CLPRDlg : public CDialogEx
{
public:
    CLPRDlg(CWnd* pParent = NULL);
    enum { IDD = IDD_LPR_DIALOG };
    ...

private:
    cv::Mat src;
    cv::Mat plate;
    cv::Mat character;
    LprCore lpr;
    string file_name;
    string text_recognition;
// Implementation
protected:
    ...
    DECLARE_MESSAGE_MAP()
public:
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
afx_msg void OnUpdateFileOpenimage(CCmdUI *pCmdUI);
afx_msg void OnBnClickedButton1();
afx_msg void OnBnClickedButton2();
CStatic text_result;
...
};
```

Menu *Open image* hoặc button *Load Image* có chức năng tạo ra các dialog để file, file c m s c l vào biến *src* và sử dụng vào cho chương trình.

```
// CLPRDlg.cpp
void CLPRDlg::OnUpdateFileOpenimage(CCmdUI *pCmdUI)
{
    // Open Image
    ...
    CFileDialog dlg(TRUE, _T("*.bitmap"), NULL, ...)
    if(dlg.DoModal() == IDOK)
    {
        file_name = to_string(dlg.GetPathName());
        src = imread(file_name);
        if(src.empty()) return;
        ...
    }
}
```

Button Show Result sẽ hiển thị các hàm xử lý và hiển thị kết quả khi nó được nhấp vào.

```
// CLPRDlg.cpp
void CLPRDlg::OnBnClickedButton2()
{
    // Show results
    if(src.empty()) return;
    Mat disp_plate, disp_character;

    lpr.set_image(src);
    lpr.do_process();
    vector<Mat> plates = lpr.get_plate_detection();
    vector<Mat> characters = lpr.get_character_isolation();
    vector<double> t = lpr.get_process_time();
    vector<string> text = lpr.get_text_recognition();
    if(plates.size() > 0)
    {
        plate = plates[0];
        resize(plate, disp_plate, Size(280,50));
        imshow("plate", disp_plate);
        character = characters[0];
        resize(character, disp_character, Size(280,50));
        imshow("character", disp_character);

        text_recognition = text[0];
}
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

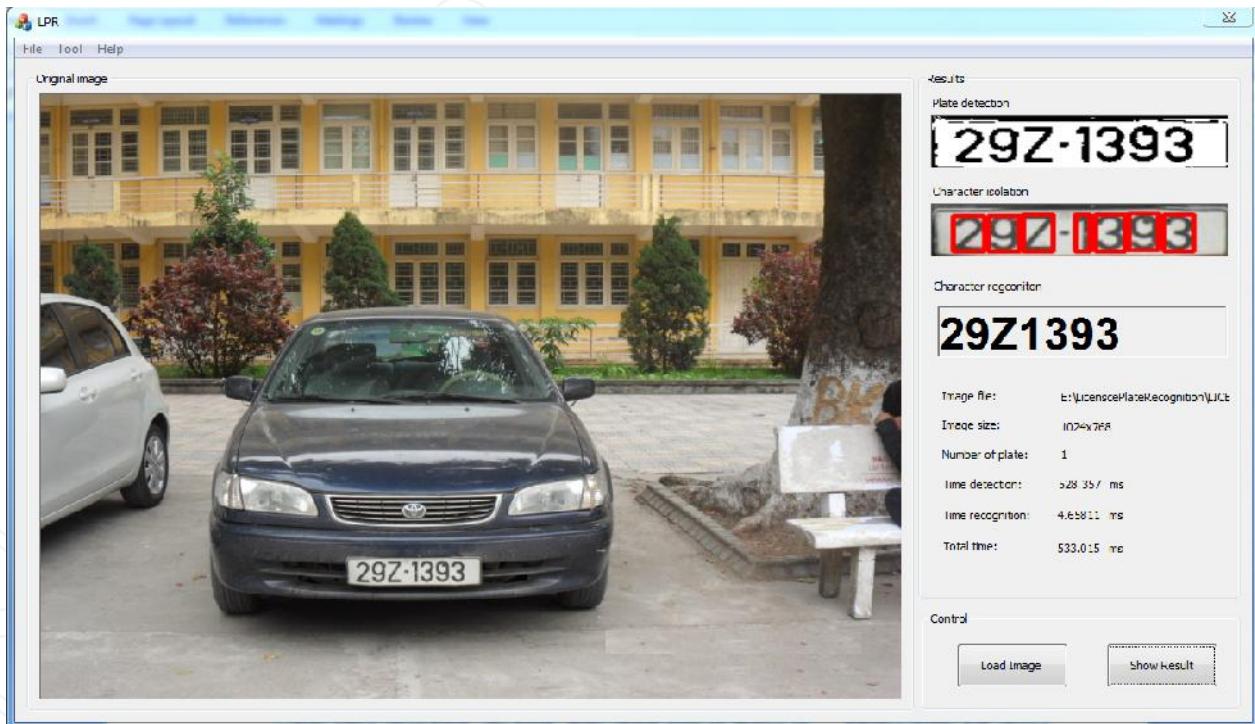
```
text_result.SetWindowTextW(to_wstring(text_recognition));
num_plate.SetWindowTextW(to_wstring((int)plates.size()));
time_detection.SetWindowTextW(to_wstring(t[0]) + " ms");
...
}

else
{
    ...
}
}
```

Trong o n code trên, ta s ki m ki m tra xem nh u vào có r ng không tr c khi t nó vào i t ng *lpr* th c hi n các phép x lý. Sau khi th c hi n các phép x lý, ta s ki m tra xem n u nh k t qu u ra mà l n h n m t bi n s (*plates.size() > 0*) thì ta s hi n th các k t qu l ên. L u ý là hi n t i ta m i ch hi n th k t qu u tiên, trong trong nh có nhi u h n m t bi n s ta có th l n l thi n th các k t qu ó m t cách d dàng.

Ngoài ra, ch ng trình còn có m t s menu khác nh m giúp ta l u k t qu nh bi n s ho c k t qu nh n d ng c d i d ng m t chu i text, menu hi n th dialog cho vi c hu n luy n mô hình SVM ...

Hình sau mô t k t qu ch y ch ng trình:



Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Bài toán nhận diện biển số xe này xây dựng mô hình chung t ng quát, ng d ng trong th c t ta c n gi i h n b t l i m t s i u ki n giúp cho vi c tìm ki m bi n s c chính xác h n, thêm vào ó các m u hu n luy n kí t c n ph i c s u t p nhi u h n, các vector c tr ng c ng ph i c tính toán t m h n giúp cho k t qu nh n d ng có chính xác cao h n n a. Ngoài ra, còn nhi u khía c nh khác liên quanh n bài toán ng d ng này trong th c t nh c n ph i xây d ng h c s d li u l u tr k t qu , so sánh k t qu , xây d ng h th ng ph n c ng, i u khi n ph n c ng i u khi n h th ng nh các h th ng th t RFID, h camera, h ng c cho các i u khi n c h c ...

3. MyCam, m t s hi u ng nh v i video.

Trong phần này, ta sẽ tìm hiểu về cách sử dụng các hàm xử lý ảnh trong thư viện OpenCV để thu hình từ webcam và bôi vẽ các khung trống có sẵn trên video. Các hàm chính như: thu hình video từ webcam, các chức năng xử lý video như làm mờ, làm méo, chuyển màu... để tạo ra các track video khác nhau như track video, track audio, track thời gian...

Cấu trúc của một file video. Ta thường xem một bài clip ca nhạc, một bộ phim trên máy tính, trên Youtube... Chúng cũng chung là một file video. Một file video có thể có nhiều định dạng khác nhau như avi, mp4, mov..., xét về cấu trúc, một file video có thể bao gồm:

- Track Video
- Track Audio
- Track Ph
- Track thời gian
- ...

Tất cả phân biệt về định dạng file video (file extension) và bài giật mã video (video codec). Định dạng file, chúng ta thường thấy avi, mp4, mov, flv... là cách cấu trúc của một file trong máy tính, nó chỉ ra cách lưu trữ các dữ liệu trên máy tính nào, các phần tử nào có thể xuất hiện... trong khi bài giật mã (video codec) chỉ ra video đã mã hóa như thế nào, có được nén hay không và cách nào để giải mã video đó.... Các bài giật mã mà ta thường thấy như H.264, XVID, MPEG, UYVY... Cùng là một định dạng file nhưng có thể có nhiều codec khác nhau, chúng ta thường thấy *.avi có thể đã mã hóa và giải mã bằng codec XVID, UYVY..., nếu có lý do mã hóa/còn file avi này không có mã hóa/còn file avi không có file avi gốc, nguyên nhân là do máy chủ đã cài đặt codec mà file avi gốc chưa có mã hóa/giật mã. Tuy nhiên, audio có thể có riêng riêng cho nó như ACC, MP3... nếu máy chủ không có audio codec thì trong một số trường hợp, file video chỉ có âm thanh mà không hiển thị hình ảnh.

Xử lý video. Xử lý video là một khái niệm chung, nó không theo đúng nghĩa của một video, khi nói đến xử lý video có nghĩa là ta đang xử lý tất cả các track của một file video bao gồm track video, track audio... tuy nhiên trong khuôn khổ bài viết này chúng ta chỉ làm việc với track video mà không làm việc với các track còn lại của file video. Xét một cách tổng quát, ta có thể xem những track video (hay video stream) là một chuỗi các khung liên tiếp nhau gọi là các frame, cách hiển thị các frame này có quy luật nhất định thông qua frame rate, chính là số frame xuất hiện trong một giây và có tính là frame/giây (frame per second hay fps). Vì xử lý video thành ra lỗi là việc xử lý các frame này. Tuy nhiên cần phải chú ý là các phép xử lý này không có nhu cầu khép kín trong một khung thời gian, tuy nhiên việc video khi mà mỗi giây ta sẽ phải xử lý tiếp theo nhau (các video thông thường có tốc độ 15-30 khung/giây)

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

ho ch n n a) thì ta c n ph i t i u hóa phép x lý t i m c có th video có th ch y c trong th i gian th c (không b gi t c c, t c hình).

Th vi n OpenCV i v i x lý video. Th vi n OpenCV ch h tr ta các hàm b t nh t m t thi t b nh webcam, camera ..., ghi các frame nh thành file video, còn vi c làm x lý video ta có th s d ng các hàm x lý nh mà th vi n cung c p x lý các frame video. Tuy nhiên, th vi n OpenCV là m t th vi n chuyên v x lý nh, nên nó ch h tr track video trong c u trúc c a file video nói trên, ngh a là khi thu video, ghi video ta ch có ph n hình (video stream) mà không th nghe c ph n ti ng. M t h n ch n a là ban u OpenCV ch h tr làm vi c trên các video avi không nén, i u ó s làm cho vi c kích th c c a vi c l u tr file t ng lên r t nhi u, gi s ta c n ghi m t o n video avi không nén trong th i gian là m t phút v i t c là 25 hình/giây và nh thu t webcam có kích th c 640x480 (nh màu, m i kênh màu c a m t pixel là 8 bit) khi ó kích th c c a m t file c n ph i có là $1*60*25*640*480*3*8 = 11059200000$ bit t c kho ng 1Gb. Tuy nhiên, hi n t i ta có th làm vi c c trên nhi u d ng file avi nén, file mpeg, flv ... mi n máy có cài các b codec t ng ng.

c video t m t file video ho c m t thi t b webcam, camera ... th vi n OpenCV cung c p cho ta l p cv::VideoCapture. Ví d ta c t m t file video, ta có th kh i t o i t ng VideoCaputre nh sau:

```
cv::VideoCapture capture = cv::VideoCapture("D:/test.avi");
```

c video t m t thi t b :

```
cv::VideoCapture capture = cv::VideoCapture(int device)
```

Trong ó, device là m t s nguyên ch ra thi t b video trên máy, thông th ng ta device b ng 0 ch ra r ng thi t b ang s d ng là webcam m c nh trên máy tính laptop, n u không bi t c th thi t b ó là gì, ta có th vi t m t ch ng trình nh ki m tra thi t b phù h p b ng cách t n bi n device t 0 t i m t s nào ó (100 ch ng h n), v i m i giá tr c a device, ki m tra xem capture có c t o ra hay không, n u thi t b phù h p, capture s tr v m t giá tr khác null, ng c l i n ó s là null:

```
for(int device = 0; device < 100; ++device)
{
    cv::VideoCapture capture = cv::VideoCapture(int device)
    if(capture)
        cout<<"thiet bi phu hop" <<device <<endl
}
```

l y ra c các frame nh trong video, ta có th s d ng hàm *read(cv::Mat& img)*, k t qu là frame c a video s c l u vào trong bi n *img*. Ta c ng có th s d ng toán t >> l y ra các frame t i t ng *capture*:

```
cv::Mat img;
capture << img;
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Lưu các frame thành video, OpenCV xây dựng một lớp `cv::VideoWriter`. Constructor của nó như sau:

`VideoWriter(string& filename, int fourcc, double fps, cv::Size frameSize, bool isColor)`

Trong đó, `filename` là file lưu trữ có tên `filename`, `fourcc` là mã mã nguyên định cho bộ mã hóa (tên của bộ mã hóa là tên kí tự) thì ta có thể tìm kiếm nguyên này dưới vào macro `CV_FOURCC(char c1, char c2, char c3, char c4)`, `frameSize` là kích thước của các frame lưu, `fps` là số khung hình/ giây video. Nếu ta biết tên của bộ mã hóa (tên của bộ mã hóa là tên kí tự) thì ta có thể tìm kiếm bằng `CV_FOURCC('X', 'V', 'T', 'D')`. `isColor` là một biến kiểm tra xem video lưu vào có phải là màu trên các kênh màu hay không, nó có giá trị là `true`.

Lưu video, ta có thể ghi hàm `write`, hoặc toán tử `<<`. Ví dụ sau mô tả cách lưu video từ webcam và lưu nó vào một file video.

```
cv::VideoCapture capture = cv::VideoCapture(0);
if(!capture.isOpened()) exit(0);
cv::Mat img;
capture >> img;
cv::VideoWriter writer = cv::VideoWriter("D:/test.avi", CV_FOURCC('X', 'V',
'I', 'D'), 25, img.size(), true);
while(1)
{
    capture >> img;
    writer << img;
}
```

Tạo chương trình MyCam.

Chương trình có界面 trên nền Dialog của MFC, cùng các hàm xử lý ảnh trong OpenCV đã biên dịch trên. Ta chia chương trình làm hai phần, một phần chứa các hàm xử lý ảnh và một phần thuần giao diện MFC. Phân chia các hàm xử lý ảnh có tên là lớp `MyCamCore`, nó bao gồm một file header tên là `mycamcore.h` và một file cài đặt `mycamcore.cpp`. Cấu trúc của hai file này có như sau:

`Mycamcore.h:`

```
#pragma once
#include <opencv2\core\core.hpp>

class MyCamCore
{
public:
    MyCamCore(void);
    ~MyCamCore(void);

    void Gray(cv::Mat&, cv::Mat&);
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
void Invert(cv::Mat&, cv::Mat&);  
void ToHSV(cv::Mat&, cv::Mat&);  
void Blur(cv::Mat&, cv::Mat&);  
...  
...  
};
```

Và *mycamcore.cpp*:

```
#include "MyCamCore.h"  
#include <opencv2\core\core.hpp>  
#include <opencv2\highgui\highgui.hpp>  
#include <opencv2\imgproc\imgproc.hpp>  
  
using namespace cv;  
  
MyCamCore::MyCamCore(void)  
{  
}  
  
MyCamCore::~MyCamCore(void)  
{  
}  
void MyCamCore::Gray(Mat &src, Mat &dst)  
{  
    if(src.channels() != 1)  
        cvtColor(src, dst, CV_BGR2GRAY);  
    else  
        dst = src;  
}  
  
void MyCamCore::Invert(Mat &src, Mat &dst)  
{  
    ...  
}  
  
void MyCamCore::ToHSV(Mat &src, Mat &dst)  
{  
    ...  
}  
  
void MyCamCore::Blur(Mat &src, Mat &dst)  
{  
    ...  
}
```

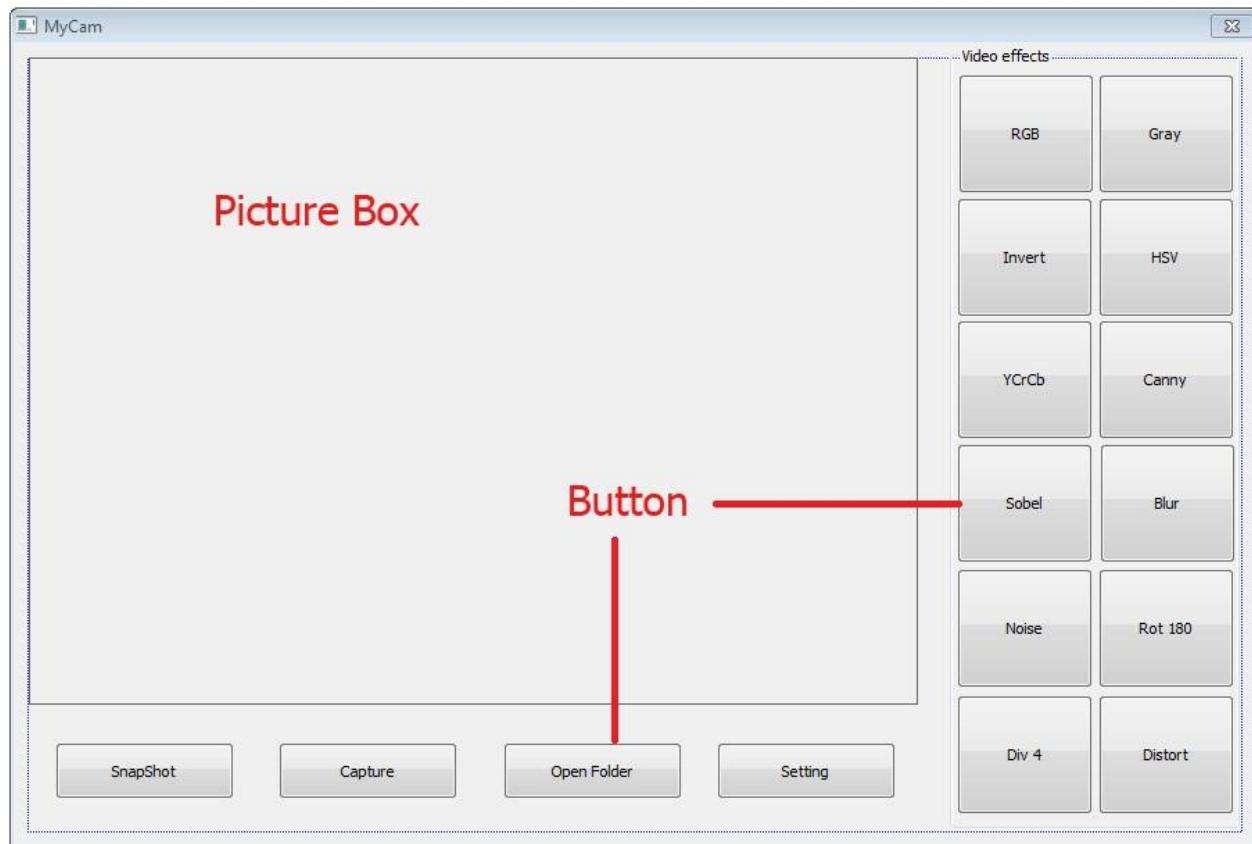
Cách cài đặt này là khá giống với cách cài đặt cách cài đặt cách cài đặt chương trình xử lý hình ảnh trên, nghĩa là chúng ta nhúng vào `src` thì qua phép xử lý sẽ cho ra một `dst`, tuy nhiên có một số điểm khác biệt là các hàm cài đặt thường giúp cho việc hiển thị video có thể liên tục, không bị tắc cù, chúng không nhảy và hàm `invert`. Hàm `invert` có tác dụng làm ngược các giá trị pixel nh, nghĩa là $dst(x,y) = 255 - src(x,y)$. Cài đặt hàm này theo cách thông thường, ta phải duyệt qua tất cả các phần tử của các kênh màu và thay đổi giá trị của các pixel toàn bộ là 255. Vì OpenCV đã tinh hóa các phép tính trên ma trận (như sử dụng các block các ô nhỏ để phân ma trận thay vì tiếp cận từng điểm, chia phép toán thành nhiều tiểu trình nhỏ song song ...) nên phép thay đổi trên sẽ nhanh hơn rất nhiều. Hàm `invert` bây giờ cài đặt như sau:

```
void MyCamCore::Invert(Mat &src, Mat &dst)
{
    static Mat mask = 255*Mat::ones(src.rows, src.cols, CV_8UC1);
    std::vector<Mat> src_bgr;
    std::vector<Mat> dst_bgr;
    split(src, src_bgr);
    for(size_t i = 0; i < src_bgr.size(); ++i)
    {
        Mat temp = mask - src_bgr[i];
        dst_bgr.push_back(temp);
    }
    cv::merge(dst_bgr, dst);
}
```

Trong hàm trên, phép chuyển ma trận thành các thay đổi riêng trên mỗi kênh màu, nên trước tiên ta tách nh thành 3 kênh màu riêng biệt, sau khi thay đổi phép toán xong ta合一 trên 1 trong 3 kênh màu này vào.

Để hiển thị phím giao diện cách chương trình, ta thiết kế một giao diện trên Dialog như sau:

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



Các button có tên biến có dạng `btn_rgb`, `btn_capture` ... Video hiển thị hình liên tiếp các frame nhảy lên một cách nhanh chóng này có thể handle sau đó cung cấp vào picture box bằng cách hiển thị trên giao diện MFC trong các ví dụ trước đó. Lắp đặt giao diện và công là công trình chính khi chạy là lập trình CMyCam, lập trình này có khả năng tự động khi ta tiến hành thoát công trình và chọn Dialog base. Ta sẽ thêm vào header `MyCamDlg.h` và file lập trình `MyCamDlg.cpp` một số biến, một số hàm hữu ích trong công trình. Header `CMyCam.h` có dạng như sau:

```
// MyCamDlg.h : header file
#pragma once

#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include "MyCamCore.h"
#include "afxwin.h"

class CMyCamDlg : public CDialogEx
{
public:
    CMyCamDlg(CWnd* pParent = NULL);
    enum { IDD = IDD_MYCAM_DIALOG };
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
enum {RGB = 0, GRAY = 1, INVERT = 2, HSV = 3, YCRCB = 4, CANNY = 5,
       SOBEL = 6, BLUR = 7, NOISE = 8, ROT = 9, DIV = 10, DISTORT =
11};

protected:
    virtual void DoDataExchange(CDataExchange* pDX);

private:
    // Các biến, các hàm chức năng trong chương trình
    bool is_stoped;
    bool is_captured;
    int type;
    cv::Mat src;
    cv::Mat dst;
    double fps;
    std::string output_folder;
    cv::Size size;
    std::string video_file_name;
    std::string img_file_name;
    cv::VideoWriter writer;
    cv::VideoCapture capture;
    MyCamCore mycam;
    void Start();
    void InitAppearance();
    bool ReadConfig(std::string);
    std::string CreateFileName();

protected:
    HICON m_hIcon;
    // các ham override
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnClose();
    DECLARE_MESSAGE_MAP()

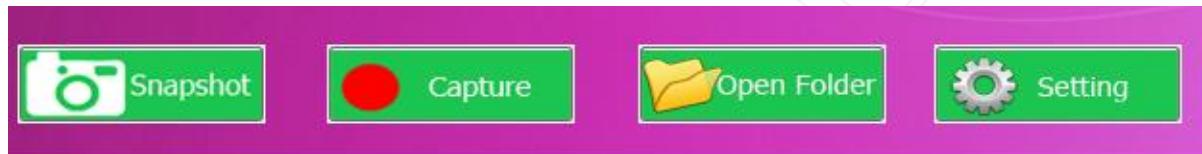
public:
    // các button
    CButton btn_snapshot;
    CButton btn_capture;
    CButton btn_setting;
    CButton btn_rgb;
    CButton btn_gray;
    ...
    // Các hàm khi click vào button
    afx_msg void OnBnClickedSnapshot();
    afx_msg void OnBnClickedCapture();
    afx_msg void OnBnClickedSetting();
    afx_msg void OnBnClickedRgb();
    afx_msg void OnBnClickedGray();
    ...
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

};

Các biến này khiничacnghìnhtrìnhbao gồm *is_stoped*, *is_captured*, *type*...nhưng m
kiểm tra xem vòng lặp có tiếp tục không, button capture có cùm
chạy có thàm x lý nhào mà ta đang ghi... Mất s hàm cài đặt trong file
MyCamDlg.cpp như :

Hàm `void InitAppearance()`: Hàm này có tác động cài đặt giao diện khi tạo
cho chương trình, mục đích là làm cho giao diện trở nên phong cách thêm
các nút trang trí vào các button. Chẳng hạn như button Snapshot, Capture,
Open Folder, Setting trang trí bằng các nút sau:



Trong MFC, ta chỉ cần nhấp vào một button hay một Control nào đó ta có thể dùng
hàm SetBitmap với giá trị của hàm là một HBITMAP, nếu vậy ta có thể cài đặt
hàm tạo ra giao diện trang trí *InitAppearance* như sau:

```
void CMyCamDlg::InitAppearance()
{
    HBITMAP snapshot_bmp = (HBITMAP)LoadImageA(0,
        "./MyCam/res/snapshot.bmp", IMAGE_BITMAP, 0, 0,
        LR_LOADFROMFILE);
    btn_snapshot.SetBitmap(snapshot_bmp);

    HBITMAP capture_bmp = (HBITMAP)LoadImageA(0,
        "./MyCam/res/capture.bmp", IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);
    btn_capture.SetBitmap(capture_bmp);

    HBITMAP setting_bmp = (HBITMAP)LoadImageA(0,
        "./MyCam/res/setting.bmp", IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);
    btn_setting.SetBitmap(setting_bmp);

    HBITMAP folder_bmp = (HBITMAP)LoadImageA(0,
        "./MyCam/res/folder.bmp", IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);
    btn_folder.SetBitmap(folder_bmp);
    ...
}
```

Hàm `bool ReadConfig(string)` có tác động đọc file config của chương
trình, file này chứa các tham số như tần số hình trên giây (fps), folder chia nhỏ
chỗ lưu video ghi vào (output_folder)... Bạn lưu ý các tham số này chia giá trị

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

m c nh, tuy nhiên khi ng i dùng i u ch nh b ng cách nh n vào button Setting thì các tham s này s c thay i và l u l i trong file config, nh ng l n ch y sau, các thông s ó s c c và có giá tr là giá tr mà ng i dùng ã t y ch nh cho chúng, hàm này s c g i ngay khi ch ng trình c ch y, hàm cài t ch y u d a vào các hàm chu n trong C++ có d ng nh sau:

```
bool CMyCamDlg::ReadConfig(std::string file_name)
{
    std::ifstream ifs(file_name);
    if(!ifs) return false;
    std::vector<std::string> lines;
    std::string line;
    while(ifs >> line)
    {
        lines.push_back(line);
    }
    if(lines.size() < 2) return false;
    fps = atof(lines.at(0).c_str());
    output_folder = lines.at(1);
    ifs.close();
    return true;
}
```

Hàm `std::string CreateFileName()` có tác d ng t o ra m t chu i tên khác nhau các th i m kh ác nhau nh m m c ích t tên cho các file nh ch p ho c file video quay. t o ra c các chu i tên không b trùng nhau, t t nh t là ta l y theo th i gian, nh v y m i th i i m ch có duy nh t m t tên c t o ra. N u nh ta nh n vào nut Snapshot (ch p nh) th i i m là 11 gi 30 phút 10 giây th 6 ngày 28 tháng 12 n m 2012 thì tên c a file nh l u l i s là *Fri Dec 28 11_30_10 2012.jpg*. Ý t ng cho vi c cài t hàm này là l y th i gian c a h th ng, sau ó thay th các kí t hai ch m ":" c a th i gian b ng kí t g ch n i d i "_" (vì kí t hai ch m là không h p l t tên trong window):

```
std::string CMyCamDlg::CreateFileName()
{
   .CreateDirectoryA(output_folder.c_str(), NULL);
    time_t rawtime;
    time(&rawtime);
    std::string t = (std::string)ctime(&rawtime);
    int pos = t.find(":");
    if(pos != tnpos)
    {
        t.replace(pos, 1, "_");
        t.replace(pos + 3, 1, "_");
        t.pop_back();
    }
    std::string path = output_folder + "/";
    return path + t;
}
```

14. Hàm `void Start()` là hàm chia vòng lặp có chức năng xử lý hình thu được từ webcam và hiển thị qua xem lý. nó có xem lý nhau tùy vào biến `type`. Hàm này sẽ bắt đầu ngay sau khi các giao diện và các thông số đã khởi động, biến `is_stoped` dùng để thúc đẩy vòng lặp của hàm này, biến này có giá trị là true khi chương trình kết thúc trong hàm `OnClose()`. Hàm `Start()` cài đặt như sau:

```
void CMyCamDlg::Start()
{
    if(!capture.isOpened()) return;
    size = cv::Size((int) capture.get(CV_CAP_PROP_FRAME_WIDTH),
    (int) capture.get(CV_CAP_PROP_FRAME_HEIGHT));

    while(!is_stoped)
    {
        capture >> src;
        switch(type)
        {
            case RGB :
                dst = src;
                break;
            case GRAY :
                mycam.Gray(src, dst);
                break;
            case INVERT:
                mycam.Invert(src, dst);
                break;
            case HSV :
                mycam.ToHSV(src, dst);
                break;
            case ... :
                ...
                break;

            default:
                dst = src;
                break;
        }
        cv::imshow("Video", dst);
        if(is_captured && writer.isOpened())
            writer << dst;
        cv::waitKey(1);
    }
}
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

15. Các hàm dùng i u khi n quá trình x lý video khi click vào các button u có chung m t dòng l nh và có d ng nh sau (gi s hàm khi click vào button btn_invert):

```
void CMyCamDlg::OnBnClickedInvert()
{
    type = INVERT;

}
```

16. Hàm khi click vào button Snapshot (ch p nh):

```
void CMyCamDlg::OnBnClickedSnapshot()
{
    img_file_name = CreateFileName() + ".jpg";
    std::vector<int> p(2);
    p.at(0) = CV_IMWRITER_JPEG_QUALITY;
    p.at(1) = 90;
    if(!dst.empty())
    {
        cv::imwrite(img_file_name, dst, p);
    }
}
```

Hàm này có tác d ng l u nh ã qua x lý v i tên file nh là tên chu i c t o ra t hàm CreateFileName nh ã nói trên và nh d ng là jpg.

17. Hàm khi click vào button Capture (ghi hình)

```
void CMyCamDlg::OnBnClickedCapture()
{
    is_captured = !is_captured;
    if(is_captured)
    {
        video_file_name = CreateFileName() + ".avi";
        if(size.height > 0 && size.width > 0)

            writer = cv::VideoWriter(video_file_name,
            CV_FOURCC('X','V','I','D'),8, size, true);
            btn_capture.SetBitmap((HBITMAP)(LoadImageA(0, "../MyCam/res/
stop.bmp",0, 0, 0, LR_LOADFROMFILE)));
            btn_snapshot.EnableWindow(false);
            btn_setting.EnableWindow(false);
            btn_folder.EnableWindow(false);
    }

    else
    {
        btn_capture.SetBitmap((HBITMAP)(LoadImageA(0, "../MyCam/res/
capture.bmp",0, 0, 0, LR_LOADFROMFILE)));
        btn_snapshot.EnableWindow(true);
        btn_setting.EnableWindow(true);
        btn_folder.EnableWindow(true);
    }
}
```

Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

```
    if(writer.isOpened())
        writer.release();

    }
}
```

Khi button Capture c nh n, i t ng writer s c t o m i v i tên c a file video c t o r a t hàm CreateFileName(), nh d ng ta s d ng ây là avi v i codec là XVID, ng th i tr ng thái c a ch ng trình c ng chuy n sang tr ng thái ghi hình, ngh a là button Capture bây gi c thay b i nh có tên lá Stop, các button khác nh Snapshot, Open Folder, Setting s b vô hi u hóa. N u button này c click l n th hai (khi quá trình ghi hình ang di n ra) thì ta s ng ng vi c ghi video l i, h y i t ng writer và tr ng thái ch ng trình tr v nh ban u.

18. Hàm khi click vào button Open Folder (m folder ch a file nh file video):

```
void CMyCamDlg::OnBnClickedFolder()
{
    std::string str_command = "explorer.exe " + output_folder;
    const char *command = str_command.c_str();
    system(command);
}
```

Hàm này s m folder ch a ch a nh ch p và video c ghi l i b ng hàm system(command), trong ó command là m t chu i ki u char g i hàm explorer.exe v i i s là ng d n t i folder c n m .

19. Hàm khi click button Setting:

```
void CMyCamDlg::OnBnClickedSetting()
{
    Setting setting;
    is_stoped = true;
    setting.DoModal();
    fps = setting.Fps;
    output_folder = setting.OutputPath;
    if(fps <= 0) fps = 8;
    if(output_folder == "") output_folder = "..\\Output";

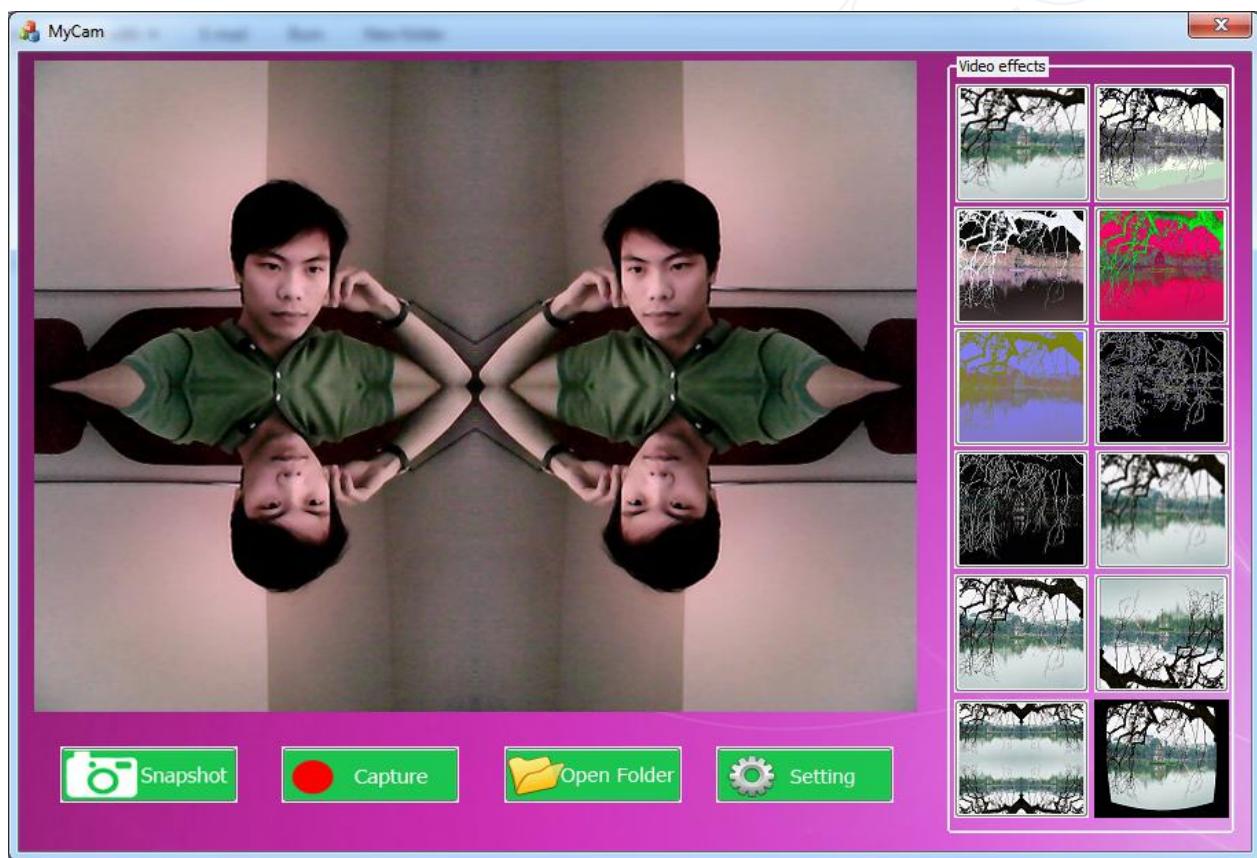
    std::ofstream ofs("config.txt");
    if(!ofs) return;
    ofs << fps << std::endl;
    ofs << output_folder;
    ofs.flush();
    ofs.close();

    is_stoped = false;
    Start();
}
```

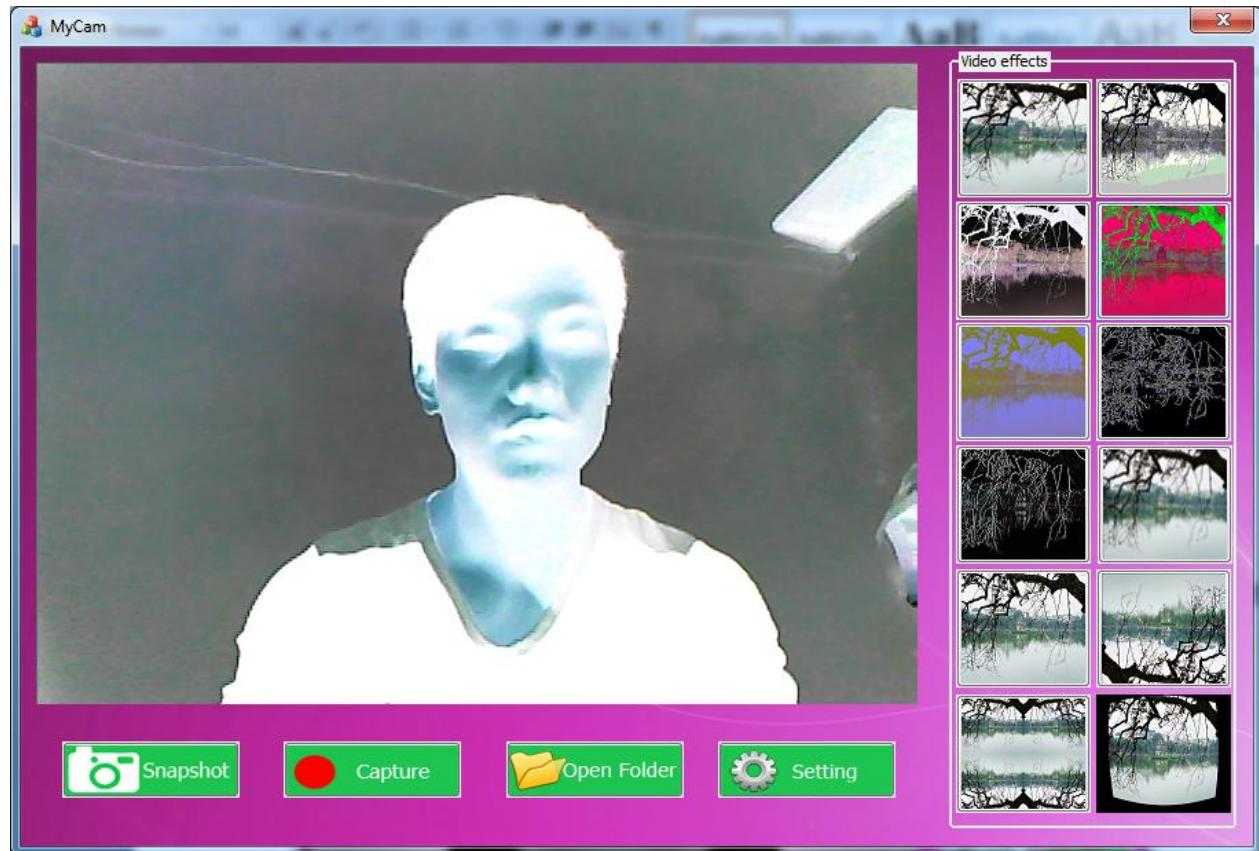
Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV

Hàm này sẽ mở ra một dialog để dùng để các giá trị cho chương trình như fps, output_folder (bên cạnh có thêm vào một số thông số như kích thước sang cho nhau trong quá trình xử lý hình ảnh)... Khi đó sau phép cài đặt này sẽ ghi vào file config và cung cấp cho các lệnh chạy chương trình tiếp theo.

Sau đây là kết quả của chương trình với một số ảnh và âm thanh:



Ứng dụng xử lý ảnh trong thực tế với thư viện OpenCV



Chương trình trên cài đặt ngang tay ra mà không có gì khác với các video stream mà đây là thu từ webcam. Trên máy tính nào nó cũng có thể chạy được và có giao diện đơn giản. Bên cạnh đó, nó có thể phát triển thêm nhiều module tùy theo nhu cầu của người dùng như giao diện đồ họa, robot vision - mô phỏng v.v.

Hệ thống

Một số ứng dụng chính có thể sử dụng trong phần mềm: Phát hiện - nhận dạng đối tượng, theo dõi vật thể (object tracking), camera calibration - stereo vision và tái tạo không gian 3D, robot vision – mô phỏng v.v. và áp dụng xử lý trong các kỳ thi robocon ...