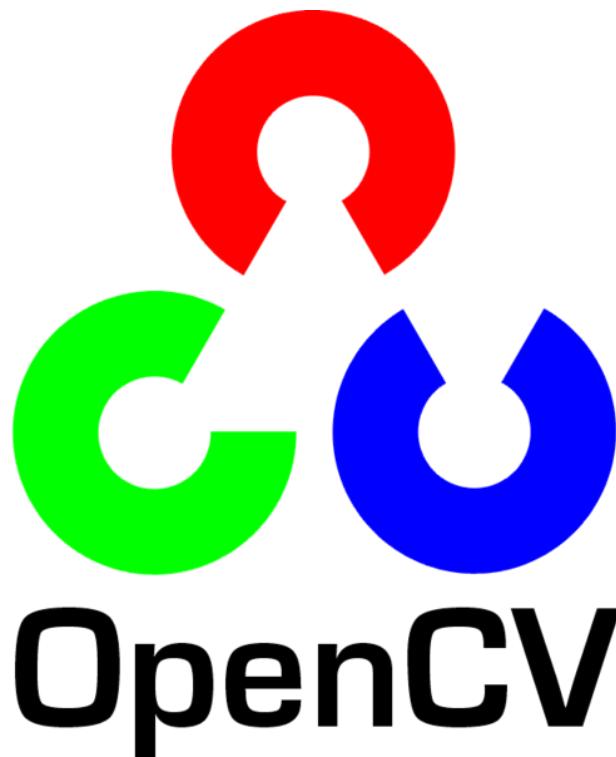


[CTT451] - [Nhập môn Thị giác Máy tính]

Tháng 3/2013

SỬ DỤNG OPENCV



MỤC LỤC

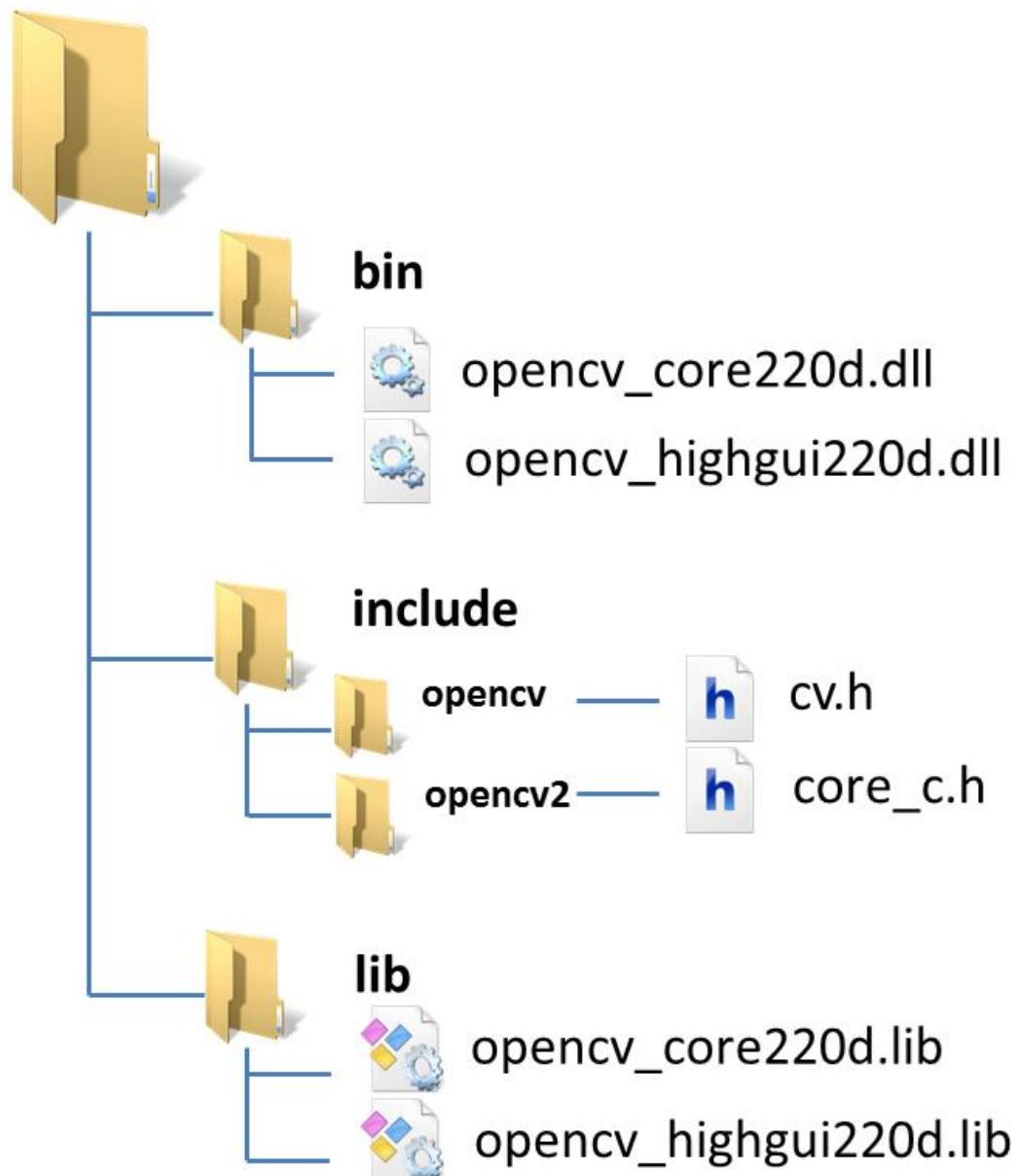
1.	Chuẩn bị:.....	2
2.	Cấu hình OpenCV với Visual Studio C++	2
3.	Chương trình đầu tiên	6
4.	Chương trình thứ hai:	7
5.	Bài tập	8

CẤU HÌNH OPENCV VỚI VS C++

1. Chuẩn bị:

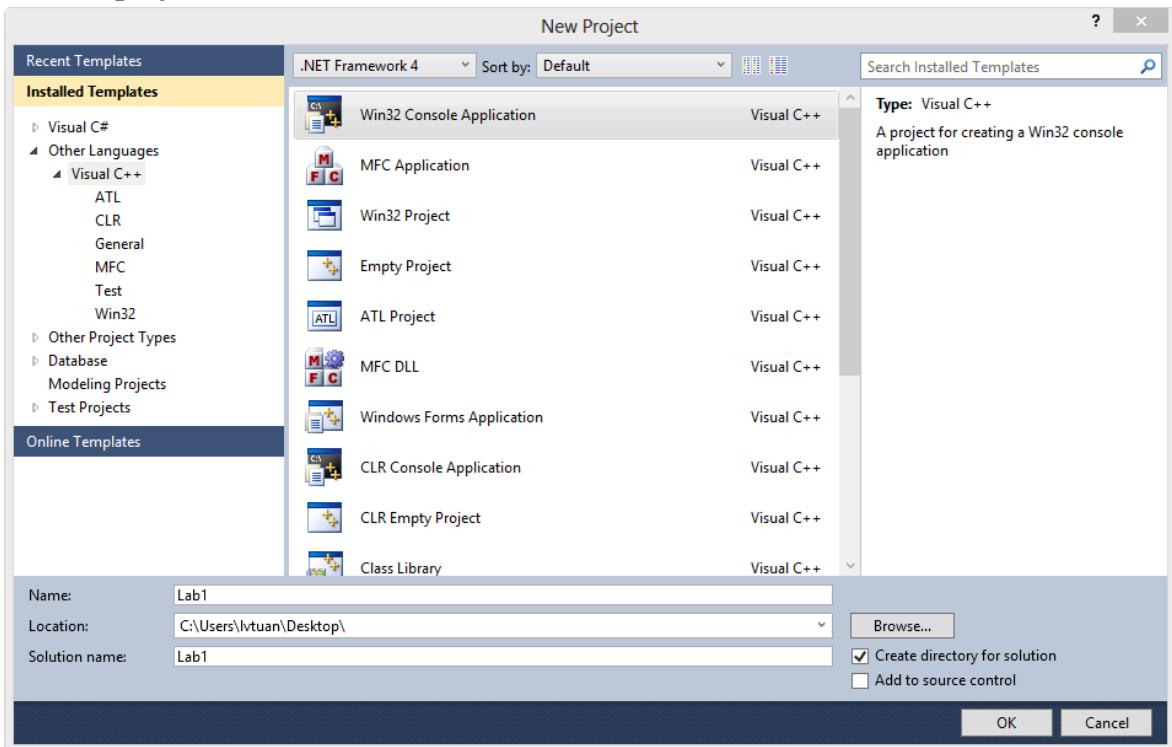
Download OpenCV tại: <http://opencv.org/downloads.html>, hoặc download thư viện OpenCV đã được biên dịch sẵn: <http://www.mediafire.com/?gda0fgqi5u8h7>.

Gồm 3 thư mục như sau:

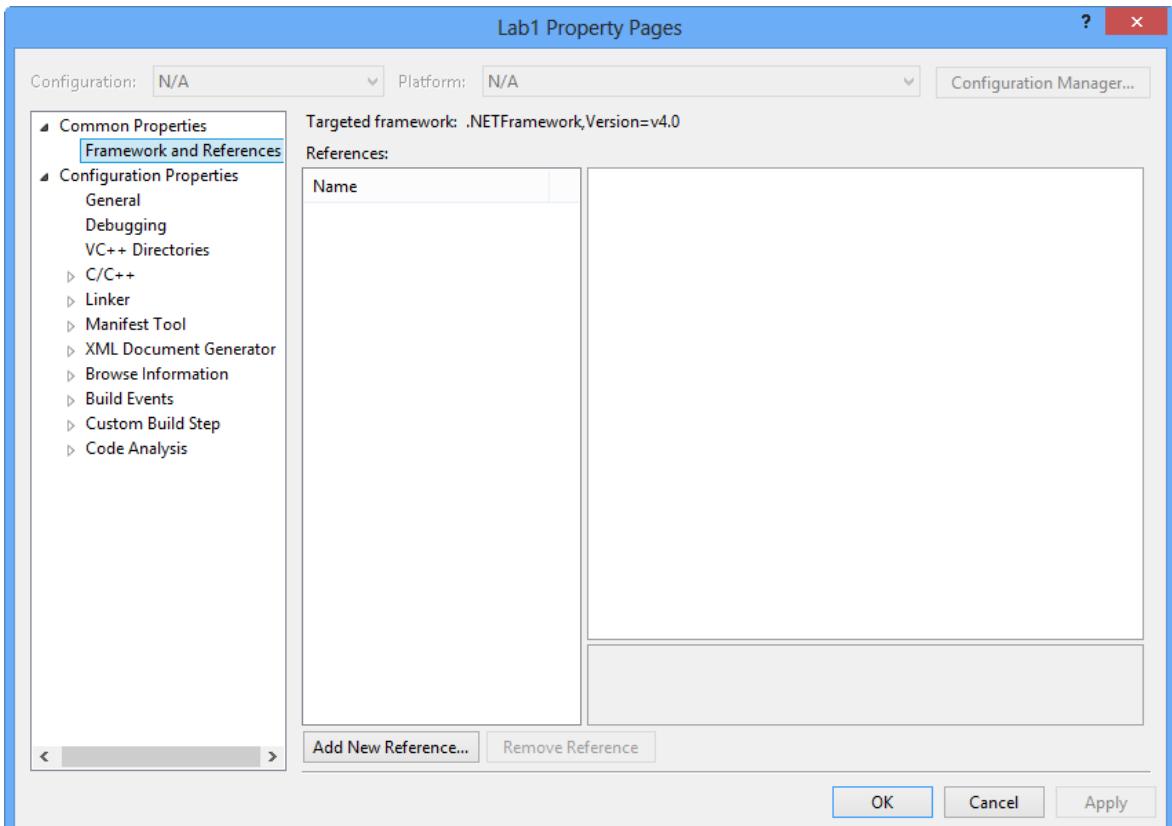


2. Cấu hình OpenCV với Visual Studio C++

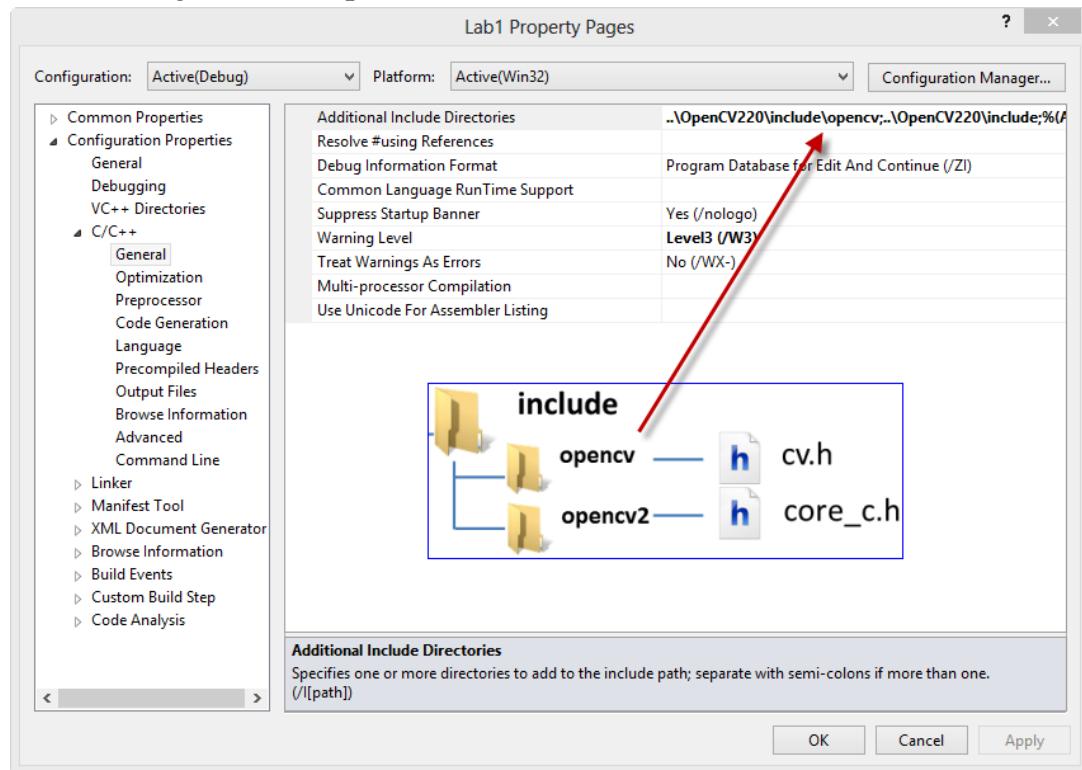
Tạo project Lab1:



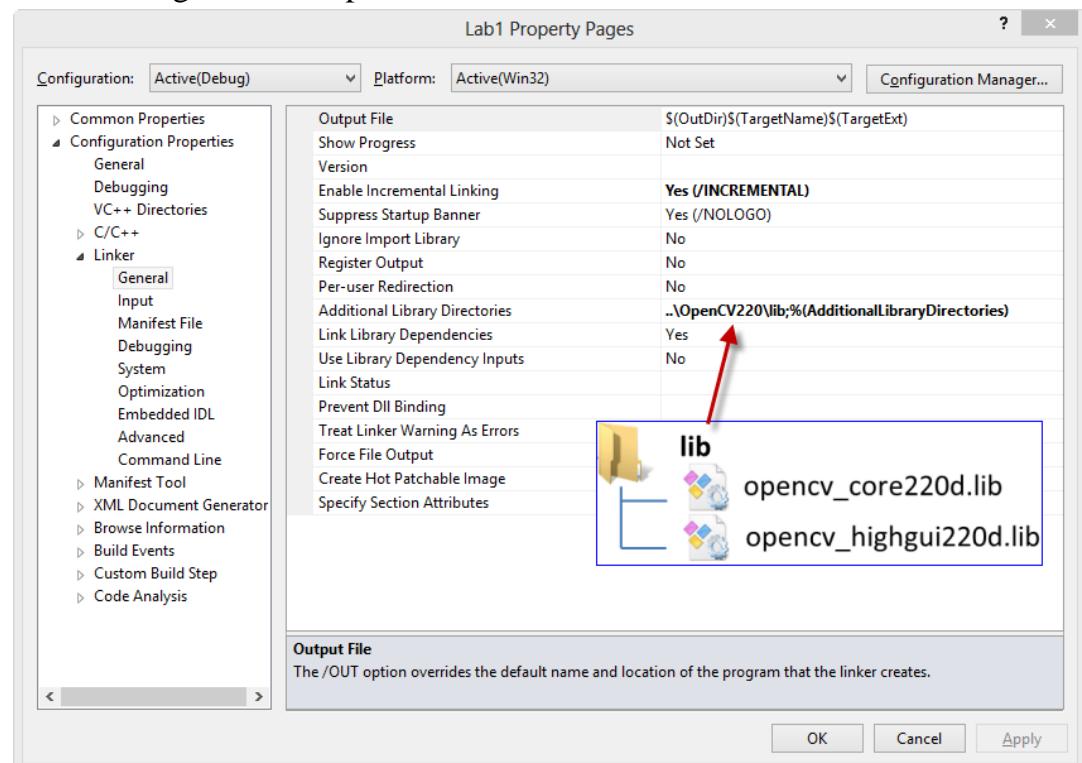
Từ Menu Project → References...



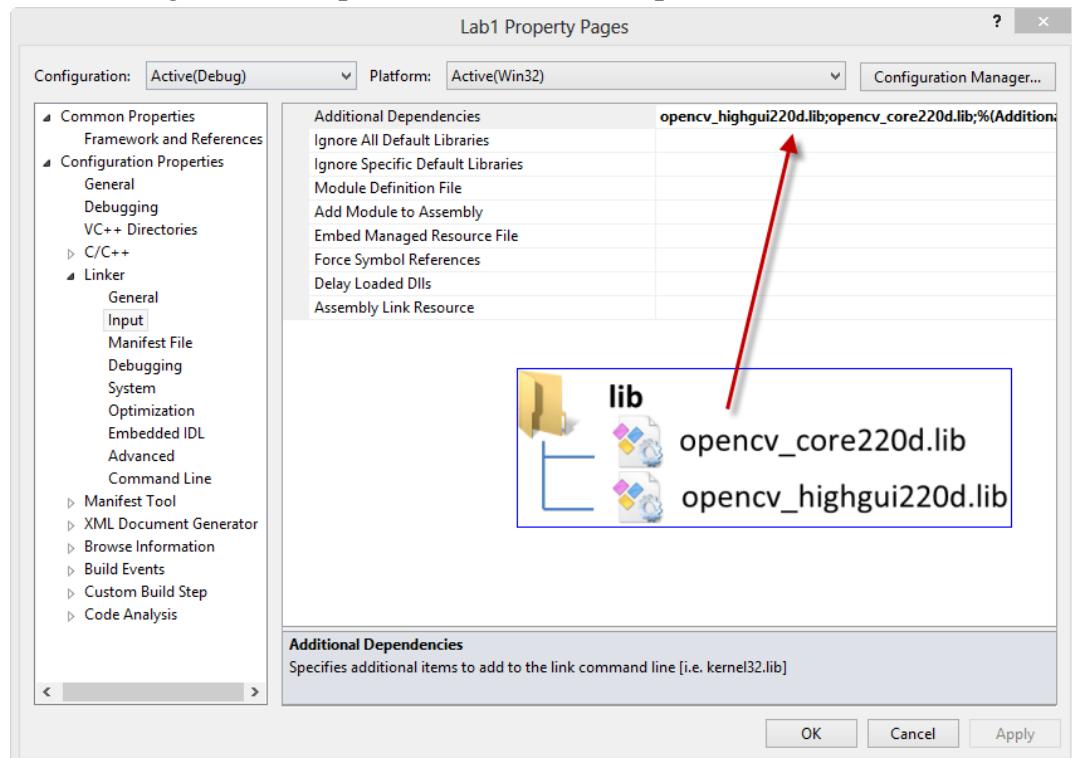
Chọn Configuration Properties → C/C++ → General



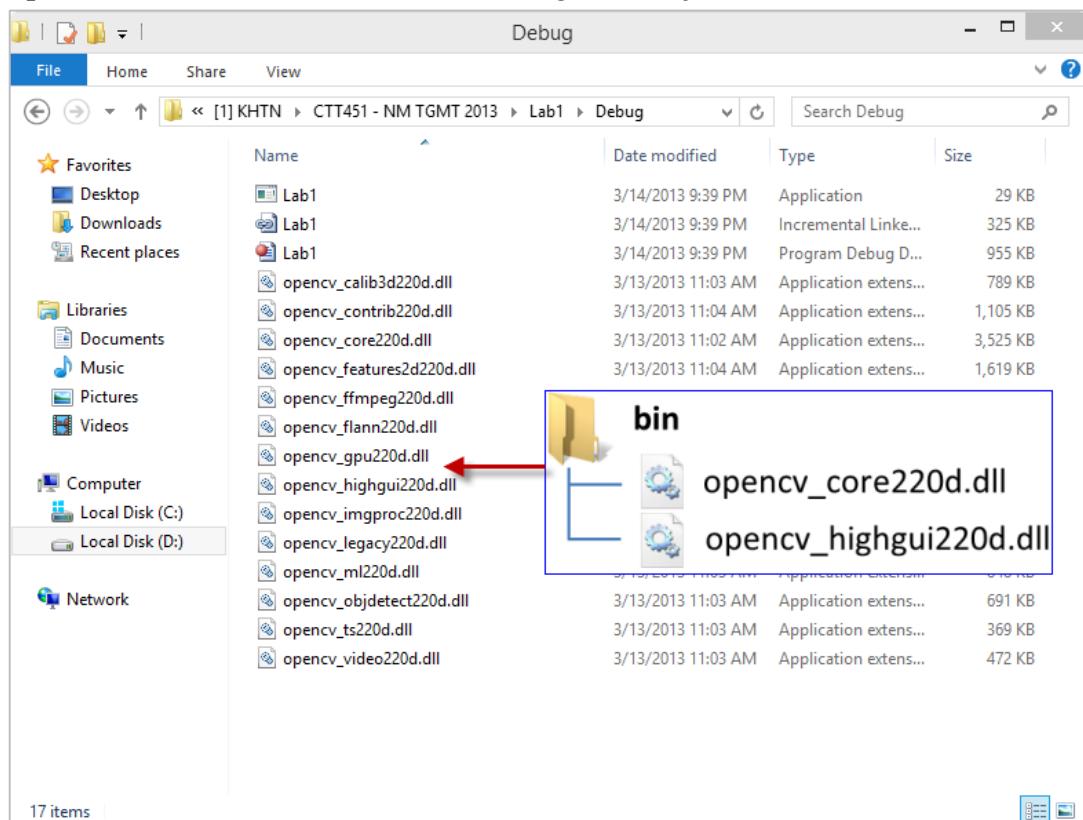
Chọn Configuration Properties → Linker → General



Chọn Configuration Properties → Linker → Input



Sao chép các file đuôi *.dll vào thư mục debug của Project



3. Chương trình đầu tiên

Viết chương trình tải và hiện một tập tin ảnh cho trước lên màn hình.

Sau khi tạo project, bổ sung đoạn chương trình sau:

```
#include "stdafx.h"
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>

int _tmain(int argc, _TCHAR* argv[])
{
    IplImage * img = cvLoadImage("C:\\\\hinh1.jpg");
    if ( img != NULL )
    {
        cvNamedWindow( "My window" );
        cvShowImage( "My window", img );
        cvWaitKey(); //Đợi người dùng nhấn 1 phím bất kỳ
        cvReleaseImage( &img ); //Giải phóng vùng nhớ
        cvDestroyWindow( "My window" ); //Đóng cửa sổ
    }
    return 0;
}
```



Đoạn chương trình trên sẽ tải một ảnh lên bộ nhớ và hiển thị ra màn hình. Ta xem xét một số dòng lệnh chính:

```
IplImage * img = cvLoadImage("C:\\hinh1.jpg");
```

Hàm `cvLoadImage()` thực hiện tải ảnh dựa vào tên file được truyền vào, đồng thời cấp phát một vùng nhớ cần thiết cho cấu trúc dữ liệu ảnh. Hàm này trả về một con trỏ trỏ tới vùng nhớ được cấp phát trên.

```
cvNamedWindow( "My window" );
```

Hàm `cvNamedWindow` tạo một cửa sổ trên màn hình để chứa và hiển thị ảnh.

```
cvShowImage( "My window", img );
```

Chúng ta đã có một ảnh dưới dạng một con trỏ `IplImage *`, và hiển thị nó lên cửa sổ vừa tạo thông qua hàm `cvShowImage()`.

4. Chương trình thứ hai:

Viết chương trình phát một đoạn video:

```
#include <highgui.h>
int main( int argc, char** argv )
{
    cvNamedWindow( "Example2", CV_WINDOW_AUTOSIZE );
    CvCapture* capture =
    cvCreateFileCapture("C:\\video1.avi");
    IplImage* frame;
    while(1) {
        frame = cvQueryFrame( capture );
        if( !frame ) break;
        cvShowImage( "Example2", frame );
        char c = cvWaitKey(33);
        if( c == 27 ) break;
    }
    cvReleaseCapture( &capture );
    cvDestroyWindow( "Example2" );
}

CvCapture* capture = cvCreateFileCapture("video1.avi");
```

Hàm `cvCreateFileCapture()` với tham số truyền vào là tên đoạn video cần load đoạn video và sẽ trả về con trỏ trỏ tới cấu trúc `CvCapture`. Cấu trúc này sẽ chứa toàn bộ thông tin đoạn video đã được đọc.

```
frame = cvQueryFrame( capture );
```

Bên trong vòng lặp while, bắt đầu đọc đoạn video. Hàm `cvQueryFrame()` bắt frame kế tiếp trong đoạn video trên vào vùng nhớ (vùng nhớ này là một phần của cấu trúc `CvCapture` đã được cấp phát trước đó). Một con trỏ được trả về cho frame được bắt giữ này. Vì `cvQueryFrame` sử dụng vùng nhớ đã được cấp phát cho cấu trúc nên không cần gọi hàm `cvReleaseImage()` cho frame này.



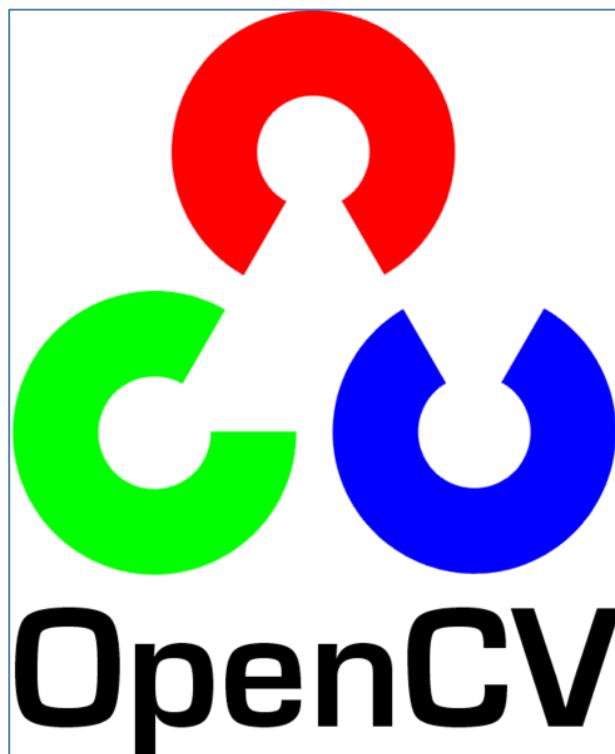
5. Bài tập

Viết chương trình chuyển ảnh sang ảnh mức xám (grayscale).

[CTT451] - [Nhập môn Thị giác Máy tính]

Tháng 3/2013

LẬP TRÌNH VỚI OPENCV



MỤC LỤC

1.	Một số hàm thông dụng trong OpenCV	2
2.	Tải và hiển thị ảnh	3
3.	Chuyển ảnh từ hệ màu sang ảnh độ xám	5
4.	Tách các kênh màu của ảnh RGB.....	6
5.	Bài tập	7

LẬP TRÌNH VỚI OPENCV

1. Một số hàm thông dụng trong OpenCV

Hàm đọc ảnh từ file:

```
IplImage* cvLoadImage(string strPathName);
```

Hàm sao chép ảnh:

```
void cvCopyImage(IplImage* src, IplImage* dst);
IplImage* cvCloneImage( const IplImage* image);
```

Hàm hủy đổi tượng ảnh:

```
void cvReleaseImage ( IplImage** image);
```

Hàm tạo cửa sổ:

```
cvNamedWindow(char* strWindowName, int flag);
flag nếu là số lẻ thì hiển thị đúng kích thước ảnh.
```

Hiển thị ảnh trên cửa sổ:

```
cvShowImage (char* strWindowName, IplImage* img);
```

Hàm chuyển đổi hệ màu:

```
void cvCvtColor(IplImage* src, IplImage* dst, int code);
```

Hàng số code quy định cách chuyển đổi có dạng:

CV_<Hệ Màu Nguồn>2<Hệ Màu Đích>

VD:

```
CV_BGR2HSV
CV_RGB2GRAY
CV_HSV2BGR
```

Tách các kênh màu:

```
cvCvtPixToPlane ( IplImage* src,
                   IplImage* img1, IplImage* img2,
                   IplImage* img3, IplImage* img4);
```

Trộn các kênh màu:

```
void cvCvtPlaneToPix( const CvArr* src0,
                      const CvArr* src1,
                      const CvArr* src2,
                      const CvArr* src3,
                      CvArr* dst);
```

2. Tải và hiển thị ảnh

Tạo lớp `IplImageWrapper` như sau:

```
class IplImageWrapper
{
protected:
    IplImage* _srcImg;
    IplImage* _destImg;
    int _width, _height;
    IplImage *r_plane, *b_plane, *g_plane;
public:
    IplImageWrapper();
    ~IplImageWrapper();

    void LoadImage(char* path);
    void ShowImage(char* windowName, int img = 0);
    void RGB2GRAY1();
    void RGB2GRAY2();
    void PixToPlane();
    void PixToPlane2();

};

};
```

Lớp gồm có 2 biến `_srcImg` và `_destImg` là ảnh nguồn và ảnh đích, hai biến `_width`, `_height` lưu lại kích thước ảnh và các biến `r_plane`, `b_plane`, `g_plane` chứa từng kênh màu của ảnh.

Hàm constructor và destructor:

```
IplImageWrapper::IplImageWrapper()
{
    _srcImg = NULL;
    _destImg = NULL;
}

IplImageWrapper::~IplImageWrapper()
{
    if(_srcImg != NULL)
        cvReleaseImage(&_srcImg);
    if(_destImg != NULL)
        cvReleaseImage(&_destImg);
}
```

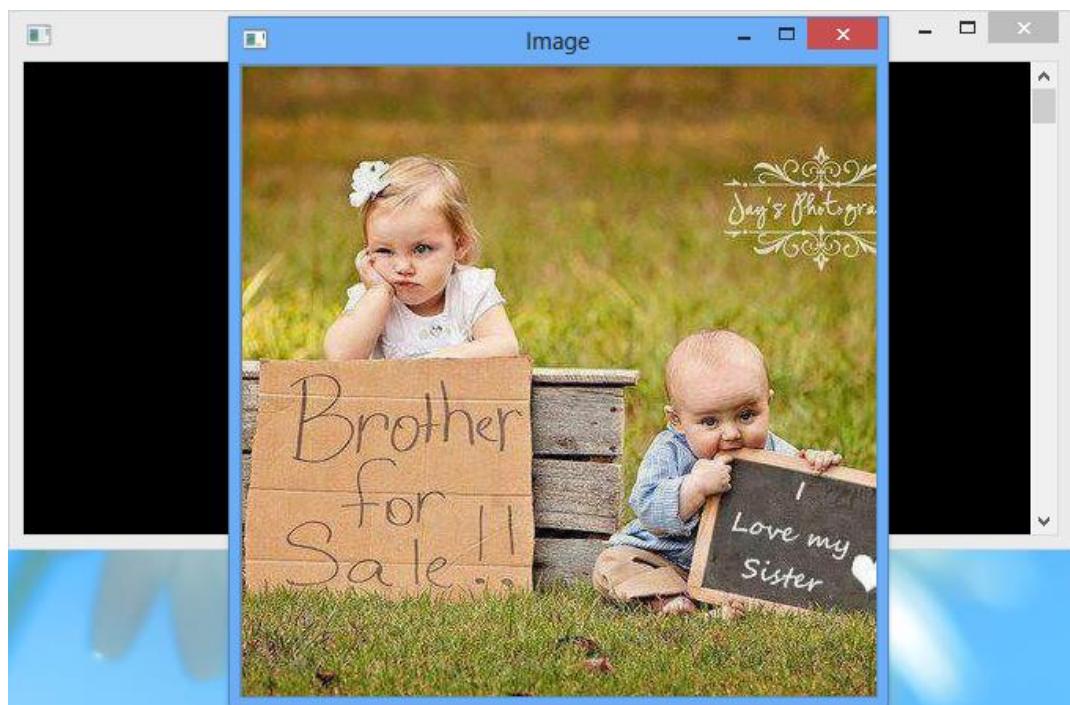
Xây dựng hàm `LoadImage` và `ShowImage` như sau:

```

void IplImageWrapper::LoadImage(char* path)
{
    if(_srcImg != NULL)
        cvReleaseImage(&_srcImg);
    _srcImg = cvLoadImage(path);
    if(_srcImg != NULL)
    {
        _width = _srcImg->width;
        _height = _srcImg->height;
    }
}

```

Hàm có chức năng load ảnh từ đường dẫn path và được trả bởi con trỏ _srcImg.



```

void IplImageWrapper::ShowImage(char* windowName, int img)
{
    IplImage* image;
    if(img == 0)
        image = _srcImg;
    else if(img == 1)
        image = _destImg;
    else if(img == 2)
        image = b_plane;
}

```

```

        else if(img == 3)
            image = g_plane;
        else if(img == 4)
            image = r_plane;
        if(image != NULL)
        {
            cvNamedWindow(windowName,1);
            cvShowImage(windowName,image);
        }
    }

```

Hàm tạo cửa sổ có tên `windowName` và hiển thị ảnh lên cửa sổ này.

3. Chuyển ảnh từ hệ màu sang ảnh độ xám

Chuyển ảnh từ hệ màu sang ảnh độ xám sử dụng hàm của OpenCV

```

void IplImageWrapper::RGB2GRAY1()
{
    if(_destImg != NULL)
        cvReleaseImage(&_destImg);
    _destImg = cvCreateImage(cvSize(_width, _height),
IPL_DEPTH_8U, 1);
    cvCvtColor(_srcImg, _destImg, CV_RGB2GRAY);
}

```

Chuyển ảnh từ hệ màu sang ảnh độ xám dựa vào công thức:

$$\text{Gray} = 0.299 * R + 0.587 * G + 0.114 * B$$

```

void IplImageWrapper::RGB2GRAY2()
{
    int step, channels;
    step = _srcImg->widthStep;
    channels = _srcImg->nChannels;

    _destImg = cvCloneImage(_srcImg);
    uchar* dataGray;
    dataGray = (uchar*)_destImg->imageData;
    int i, j;
    for(i = 0; i < _height; i++)
        for (j = 0; j < _width; j++)
    {
        uchar r,g,b,gray_value;
        b = dataGray[i*step+j*channels];
        g = dataGray[i*step+j*channels + 1];
    }
}

```

```

        r = dataGray[i*step+j*channels + 2];
        gray_value = (int)(r*0.3 + g*0.59
                            + b*0.11);
        dataGray[i*step+j*channels] =
            dataGray[i*step+j*channels + 1] =
            dataGray[i*step+j*channels + 2] =
            gray_value;
    }
}

```



4. Tách các kênh màu của ảnh RGB

Tách các kênh màu sử dụng hàm của OpenCV

```

void IplImageWrapper::PixToPlane()
{
    r_plane = cvCreateImage(cvSize(_width,_height), 8,
1);
    g_plane = cvCreateImage(cvSize(_width,_height), 8,
1);
    b_plane = cvCreateImage(cvSize(_width,_height), 8,
1);
    cvCvtPixToPlane(_srcImg, b_plane, g_plane, r_plane,
NULL);
}

```

Tách các kênh màu (truy cập dữ liệu ảnh)

```
void IplImageWrapper::PixToPlane2()
```

```

{
    int i, j, step, channels;
    step = _srcImg->widthStep;
    channels = _srcImg->nChannels;

    b_plane = cvCloneImage(_srcImg);
    g_plane = cvCloneImage(_srcImg);
    r_plane = cvCloneImage(_srcImg);

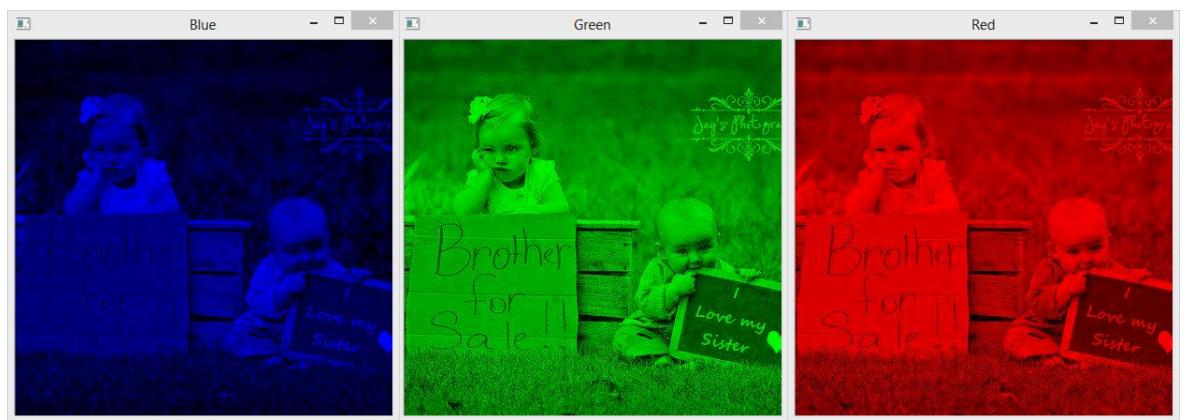
    uchar *dataB, *dataG, *dataR;
    dataB = (uchar *)b_plane->imageData;
    dataG = (uchar *)g_plane->imageData;
    dataR = (uchar *)r_plane->imageData;

    for(i = 0; i < _height; i++)
        for (j = 0; j < _width; j++)
    {
        dataB[i*step+j*channels+1] = 0;
        dataB[i*step+j*channels+2] = 0;

        dataG[i*step+j*channels] = 0;
        dataG[i*step+j*channels+2] = 0;

        dataR[i*step+j*channels] = 0;
        dataR[i*step+j*channels+1] = 0;
    }
}

```



5. Bài tập

- Tạo ảnh âm bản (negative image) của ảnh mức xám
- Tạo ảnh âm bản của ảnh màu

- Thực hiện cộng thêm một hằng số vào giá trị tại mỗi điểm ảnh trên ảnh mức xám.
- Thực hiện với 3 hằng số, mỗi hằng số được dùng để cộng vào giá trị của kênh màu tương ứng tại mỗi điểm trên ảnh màu.

[CTT451] - [Nhập môn Thị giác Máy tính]

Tháng 3/2013

CAMERA CALIBRATION



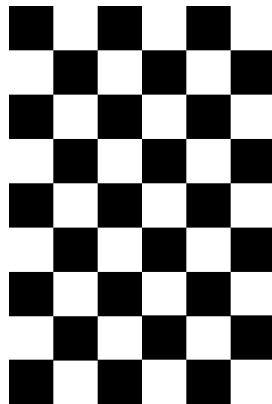
MỤC LỤC

1.	Một số hàm Camera Calibration trong OpenCV	2
2.	Tính các tham số intrinsic và extrinsic của camera.....	3
2.1.	Xác định các điểm góc trên ảnh bàn cờ:	3
2.2.	Tìm các tham số intrinsic và extrinsic	4
3.	Bài tập	4

CAMERA CALIBRATION

1. Một số hàm Camera Calibration trong OpenCV

Sử dụng bàn cờ như sau:



- Hàm xác định vị trí các góc của bàn cờ:

```
int cvFindChessboardCorners(  
    const void* image,  
    CvSize patternSize, CvPoint2D32f* corners,  
    int* cornerCount=NULL,  
    int flags=CV_CALIB_CB_ADAPTIVE_THRESH)
```

- Vẽ các góc trên bàn cờ:

```
void cvDrawChessboardCorners(  
    CvArr* image,  
    CvSize pattern_size,  
    CvPoint2D32f* corners,  
    int count,  
    int pattern_was_found);
```

- Tính các tham số intrinsic và extrinsic của camera:

```
void cvCalibrateCamera2(  
    CvMat* object_points,  
    CvMat* image_points,  
    int* point_counts,  
    CvSize image_size,  
    CvMat* intrinsic_matrix,  
    CvMat* distortion_coeffs,  
    CvMat* rotation_vectors = NULL,  
    CvMat* translation_vectors = NULL,
```

```
int flags = 0);
```

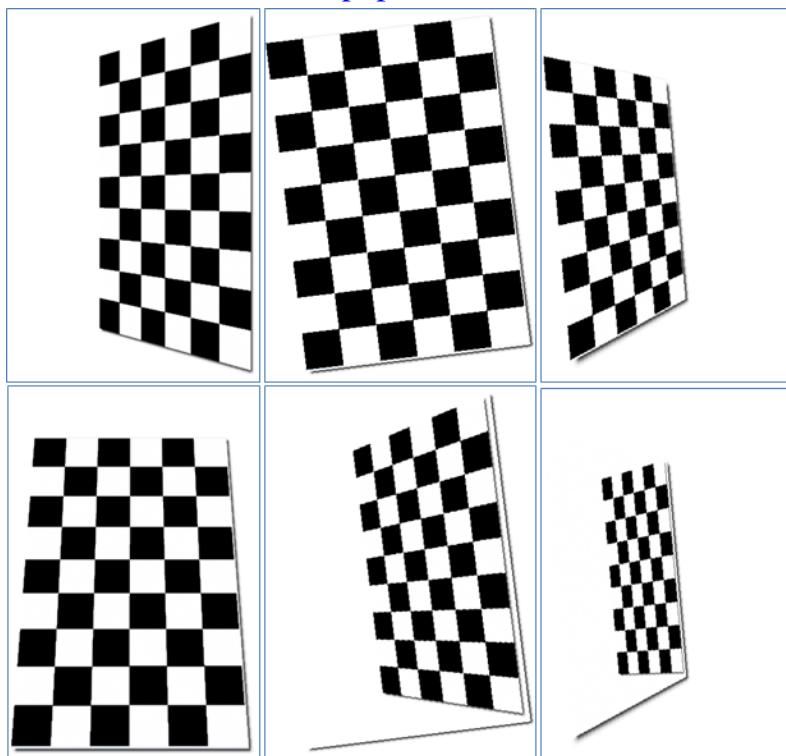
- Tính undistortion map:

```
void cvInitUndistortMap(  
    const CvMat* cameraMatrix,  
    const CvMat* distCoeffs,  
    CvArr* map1,  
    CvArr* map2);
```

2. Tính các tham số intrinsic và extrinsic của camera

Cho tập ảnh bàn cờ sau:

(<http://courses.fit.hcmus.edu.vn/file.php/1893/Checkerboard.rar>)



2.1. Xác định các điểm góc trên ảnh bàn cờ:

```
while(done != -1)  
{  
    char path[200];  
    strcpy(path, search_dir);  
    strcat(path, fileinfo.name);  
  
    //Load image from folder  
    image = cvLoadImage(path, 1);  
    cvShowImage("Original Image", image);  
    gray_image = cvCreateImage( cvGetSize( image ), 8, 1 );  
  
    // Find chessboard corners:
```

```

int found = cvFindChessboardCorners( image, board_sz, corners,
                                      &corner_count, CV_CALIB_CB_ADAPTIVE_THRESH |
                                      CV_CALIB_CB_FILTER_QUADS );

// Get subpixel accuracy on those corners
cvCvtColor( image, gray_image, CV_BGR2GRAY );
cvFindCornerSubPix(gray_image, corners, corner_count, cvSize(11, 11 ),
                    cvSize( -1, -1 ), cvTermCriteria( CV_TERMCRIT_EPS+CV_TERMCRIT_ITER,
                    30, 0.1 ));

// Draw it
cvDrawChessboardCorners(image,board_sz,corners,corner_count, found );
cvShowImage( "Calibration", image );

// If we got a good board, add it to our data
if( corner_count == board_n )
{
    step = successes*board_n;
    for( int i=step, j=0; j < board_n; ++i, ++j )
    {
        CV_MAT_ELEM( *image_points, float, i, 0 ) =
        corners[j].x;
        CV_MAT_ELEM( *image_points, float, i, 1 ) =
        corners[j].y;
        CV_MAT_ELEM( *object_points, float, i, 0 ) = j/board_w;
        CV_MAT_ELEM( *object_points, float, i, 1 ) = j%board_w;
        CV_MAT_ELEM( *object_points, float, i, 2 ) = 0.0f;
    }
    CV_MAT_ELEM( *point_counts, int, successes, 0 ) = board_n;
    successes++;
}
cvWaitKey(500);
done = _findnext( handle, &fileinfo );
}

```

Đoạn code sẽ lần lượt load các ảnh trong thư mục có đường dẫn là `search_dir`. Với mỗi ảnh sẽ tiến hành tìm các điểm góc.

2.2. Tìm các tham số intrinsic và extrinsic

```

// Calibrate the camera
cvCalibrateCamera2( object_points2, image_points2, point_counts2,
                    cvGetSize( image ), intrinsic_matrix, distortion_coeffs,
                    NULL, NULL, CV_CALIB_FIX_ASPECT_RATIO );

// Save the intrinsics and distortions
cvSave( "Intrinsics.xml", intrinsic_matrix );
cvSave( "Distortion.xml", distortion_coeffs );

```

3. Bài tập

- Từ tập các ảnh bàn cờ (<http://courses.fit.hcmus.edu.vn/file.php/1893/Checkerboard.rar>) tìm các điểm góc và tính các tham số intrinsic và extrinsic.
- Hiển thị ảnh Undistort.

[CTT451] - [Nhập môn Thị giác Máy tính]

Tháng 4/2013

CANNY EDGE DETECTION



Bộ môn TGMT và KH Rô-bốt
Khoa Công nghệ thông tin
ĐH Khoa học tự nhiên TP HCM



MỤC LỤC

MỤC LỤC	1
1 Thuật toán Phát hiện cạnh Canny	3
2 Cài đặt thuật toán phát hiện cạnh bằng OpenCV	4
3 Bài tập	6
Tài liệu tham khảo	7

1 Thuật toán Phát hiện cạnh Canny

Bước 1: Giảm nhiễu

Thông thường để giảm nhiễu sử dụng các bộ lọc làm mờ. Có thể sử dụng bộ lọc Gaussian để tích hợp với ảnh:

$$\frac{1}{159} *$$

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

Bước 2: Tính độ lớn và góc của Gradient

Tính đạo hàm $D_x(x, y)$ và $D_y(x, y)$ theo chiều x và y của ảnh. Một số bộ lọc như: Roberts, Prewitt, Sobel.

Bộ lọc Sobel 3x3:

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

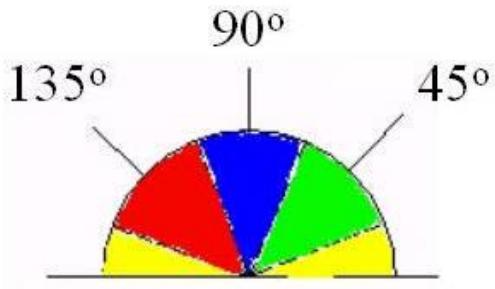
Độ lớn Gradient:

$$D = \sqrt{D_x^2(x, y) + D_y^2(x, y)}$$

Góc Gradient:

$$\theta = \arctan\left(\frac{D_x(x, y)}{D_y(x, y)}\right)$$

Tính θ' bằng cách làm tròn từ góc θ vào một trong bốn hướng: $0^0, 45^0, 90^0, 135^0$.



Bước 3: chặn không cực đại (Non-Maximum Supression)

Bước này chỉ giữ lại những pixel thuộc cạnh mà có độ lớn gradient lớn nhất

Xem xét 3 pixel trong vùng 3×3 xung quanh pixel (x,y) :

- Nếu $\theta(x, y) = 0^\circ$ thì $(x+1, y), (x, y)$ và $(x-1, y)$ được xem xét.
- Nếu $\theta(x, y) = 90^\circ$ thì $(x, y+1), (x, y)$ và $(x, y-1)$.
- Nếu $\theta(x, y) = 45^\circ$ thì $(x+1, y+1), (x, y)$ và $(x-1, y-1)$.
- Nếu $\theta(x, y) = 135^\circ$ thì $(x-1, y+1), (x, y)$ và $(x+1, y-1)$.

Nếu pixel (x, y) có gradient lớn nhất của 3 pixel xem xét thì pixel đó là cạnh.

Bước 4: Nguõng Hysteresis (Hysteresis Thresholding)

Hysteresis sử dụng 2 ngưỡng, ngưỡng t_{high} và t_{low} . Pixel mà có độ lớn gradient $D < t_{low}$ thì được loại ngay lập tức. Những pixel $t_{low} < D < t_{high}$ được giữ lại nếu là một cạnh liên tục với những pixel có độ lớn gradient $D > t_{high}$.

2 Cài đặt thuật toán phát hiện cạnh bằng OpenCV

Xây dựng class Canny như sau:

```

class Canny
{
protected:
    IplImage *_srcImg, *_destImg;
    int _width, _height;
    int _lowThreshold, _maxThreshold;

public:
    Canny(int low, int max)
    {
        _srcImg = NULL;
    }
}

```

```

        _destImg = NULL;
        _lowThreshold = low;
        _maxThreshold = max;
    }

    int CannyEdgeDetection(char *path);
};

```

Hàm CannyEdgedetection:

```

int Canny::CannyEdgeDetection(char *path)
{
    _srcImg = cvLoadImage(path);
    if(_srcImg == NULL)
        return 0;
    cvNamedWindow("Image");
    cvShowImage("Image", _srcImg);

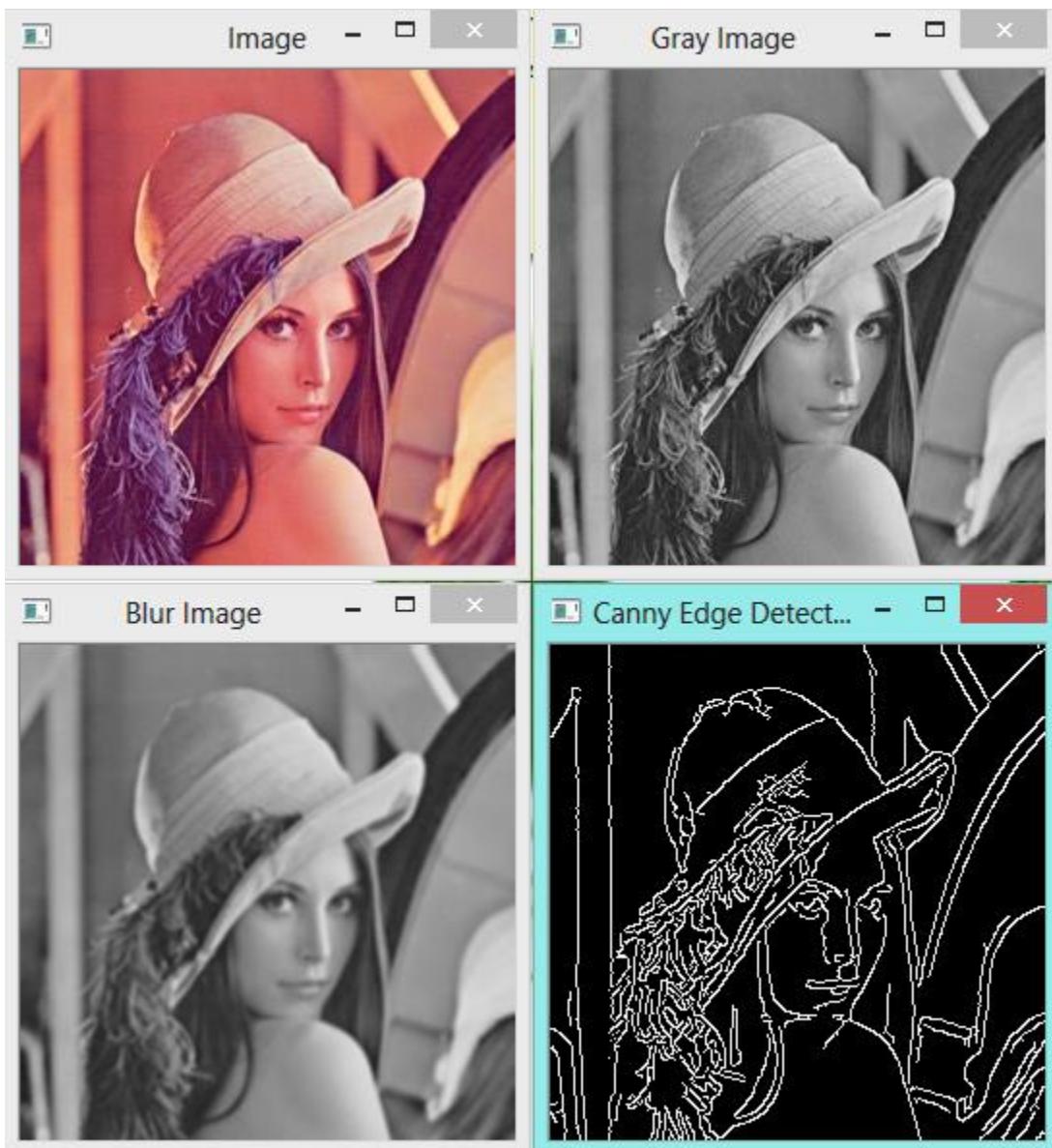
    IplImage *grayImg, *blurImg;
    grayImg = cvCreateImage(cvGetSize(_srcImg), IPL_DEPTH_8U, 1);
    cvCvtColor(_srcImg, grayImg, CV_BGR2GRAY);
    cvNamedWindow("Gray Image");
    cvShowImage("Gray Image", grayImg);

    blurImg = cvCreateImage(cvGetSize(_srcImg), IPL_DEPTH_8U, 1);
    cvSmooth(grayImg, blurImg, CV_GAUSSIAN, 5, 5);
    cvNamedWindow("Blur Image");
    cvShowImage("Blur Image", blurImg);

    _destImg = cvCreateImage(cvGetSize(_srcImg), IPL_DEPTH_8U, 1);
    cvCanny(blurImg, _destImg, _lowThreshold, _maxThreshold, 3);

    cvNamedWindow("Canny Edge Detection");
    cvShowImage("Canny Edge Detection", _destImg);
    return 1;
}

```



3 Bài tập

Viết chương trình phát hiện biên cạnh dựa vào thuật toán Canny như trong mục 1.

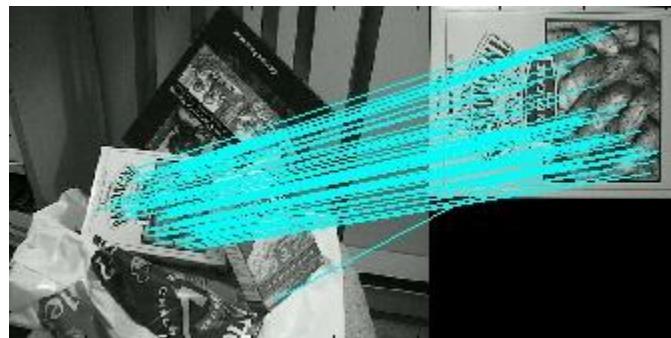
Tài liệu tham khảo

- [1] Canny, J., *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

[CTT451] - [Nhập môn Thị giác Máy tính]

Tháng 4/2013

SIFT – HARRIS - BLOB



Bộ môn TGMT và KH Rô-bốt
Khoa Công nghệ thông tin
ĐH Khoa học tự nhiên TP HCM



MỤC LỤC

MỤC LỤC	1
1 Scale invariant feature transform (SIFT).....	3
1.1 Xây dựng không gian tỉ lệ	3
1.2 Dò tìm cực trị cục bộ	4
1.3 Loại bỏ keypoint có độ tương phản (contrast) thấp.....	4
1.4 Loại bỏ keypoint nằm trên biên cạnh	5
1.5 Gán hướng cho keypoint	5
1.6 Miêu tả đặc trưng.....	6
2 Thuật toán phát hiện góc Harris	7
3 Thuật toán phát hiện Blob	8
4 Bài tập.....	9
Tài liệu tham khảo	10

1 Scale invariant feature transform (SIFT)

1.1 Xây dựng không gian tỉ lệ

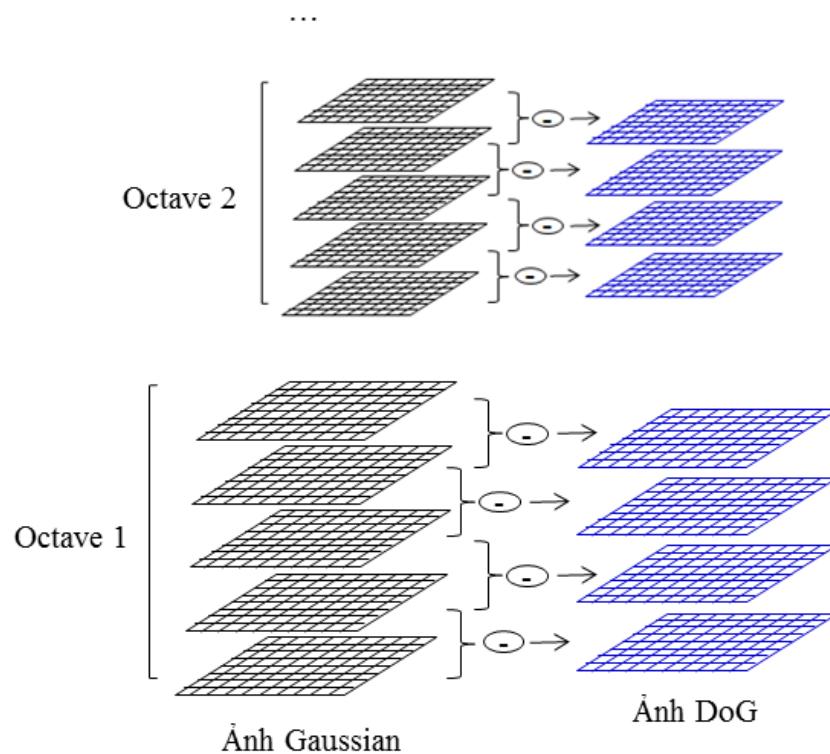
- Không gian tỉ lệ (scale space) gồm các ảnh $L(x, y, \sigma)$, tích chập (convolution) của lọc Gaussian $G(x, y, \sigma)$ với ảnh đầu vào $I(x, y)$.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

- Không gian tỉ lệ được chia thành từng octave, mỗi octave gồm 1 số nguyên s ảnh. Các ảnh tích chập với lọc Gaussian khác nhau ở tham số k , $k = 2^{1/s}$. Phải tạo ra $s + 3$ ảnh cho mỗi octave, ảnh được làm mờ thứ i trong octave sẽ là $L(x, y, k^i \sigma)$.
- Difference-of-Gaussian được sử dụng để phát hiện điểm trọng yếu (keypoint) trong không gian tỉ lệ, bằng cách trừ 2 ảnh L kế nhau trong octave như trong **Error! Reference source not found..**

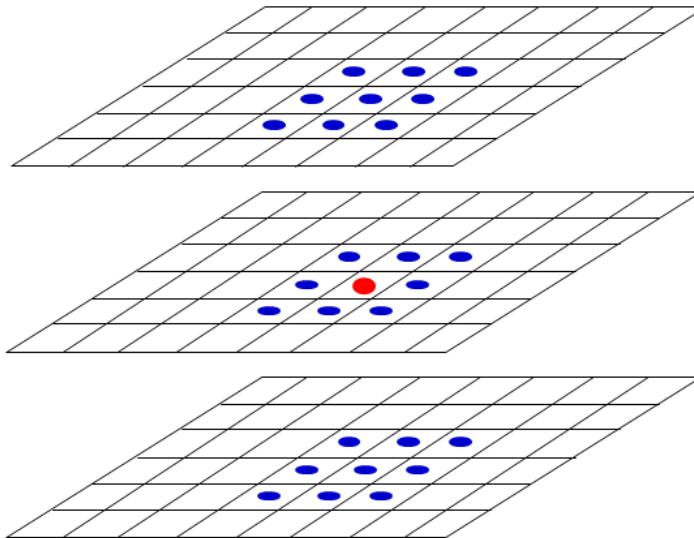
$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \quad (1.1)$$

- Sau khi xử lý xong trên 1 octave, ảnh đầu tiên trong octave kế tiếp được giảm kích thước đi một nữa.



1.2 Dò tìm cực trị cục bộ

- Tìm cực đại và cực tiểu của các $D(x, y, \sigma)$, mỗi điểm được so sánh với 8 lân cận trong ảnh hiện tại, và 9 lân cận trong scale trên và scale dưới. Một điểm được chọn chỉ khi nó lớn hơn tất cả hoặc nhỏ hơn tất cả các lân cận.



1.3 Loại bỏ keypoint có độ tương phản (contrast) thấp

Áp dụng khai triển Taylor cấp 2 cho hàm $D(x, y, \sigma)$.

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \quad (1.2)$$

Trong đó: $x = (x, y, t)^T$

Đạo hàm cấp 1 của D tại x:

$$\frac{\partial D}{\partial x} = \left(\frac{\partial D}{\partial x} \quad \frac{\partial D}{\partial y} \quad \frac{\partial D}{\partial \sigma} \right)$$

Đạo hàm cấp 2 của D tại x:

$$\frac{\partial^2 D}{\partial x^2} = \begin{pmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial y \partial x} & \frac{\partial^2 D}{\partial \sigma \partial x} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial \sigma \partial y} \\ \frac{\partial^2 D}{\partial x \partial \sigma} & \frac{\partial^2 D}{\partial y \partial \sigma} & \frac{\partial^2 D}{\partial \sigma^2} \end{pmatrix}$$

Lấy đạo hàm của $D(x)$ và cho bằng không:

$$\hat{x} = - \frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x} \quad (1.3)$$

Nếu $\hat{x} > 0.5$ tại bất kỳ hướng nào thì điểm cực trị nằm gần một keypoint khác hơn là keypoint đang xét, loại keypoint đang xét.

Giá trị của hàm $D(\hat{x})$ được sử dụng để loại những cực trị có độ tương phản thấp, thay \hat{x} vào D ta được:

$$D(\hat{x}) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} \hat{x}$$

Những cực trị có giá trị $|D(\hat{x})| < 0.03$ sẽ được loại bỏ (giả sử giá trị độ xám của điểm ảnh nằm trong đoạn $[0,1]$).

1.4 Loại bỏ keypoint nằm trên biên cạnh

Tiến hành loại bỏ các điểm nằm trên biên cạnh, sẽ không ổn định nếu có một lượng nhiễu nhỏ. Phương pháp Harris được dùng để xác định xem 1 keypoint nằm ở góc, biên cạnh hay trên vùng phẳng.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \quad (1.4)$$

Đạo hàm được ước lượng thông qua hiệu số các điểm lân cận. Để tránh tính các giá trị riêng, mà chỉ cần quan tâm tới tỉ số của chúng. Gọi λ_1, λ_2 là hai giá trị riêng của H .

$$Tr(H) = D_{xx} + D_{yy} = \lambda_1 + \lambda_2$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 = \lambda_1 \lambda_2$$

Gọi r là tỉ số giữa 2 giá trị riêng: $\lambda_1 = r\lambda_2$

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \lambda_2} = \frac{(r\lambda_2 + \lambda_2)^2}{r\lambda_2^2} = \frac{(r+1)^2}{r}$$

Chỉ giữ lại những keypoint mà có:

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

Nguồn đề nghị $r = 10$.

1.5 Gán hướng cho keypoint

Gán một hướng thích hợp cho keypoint dựa trên đặc tính cục bộ của ảnh, vì vậy bất biến với phép quay ảnh.

Với mỗi ảnh được làm mờ $L(x,y)$ ở scale gần nhất mà keypoint được phát hiện, độ lớn gradient $m(x,y)$ và hướng $\theta(x,y)$ được tính:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) =$$

$$\tan^{-1}((L(x, y + 1) - L(x, y - 1))/(L(x + 1, y) - L(x - 1, y)))$$

Lược đồ hướng gồm 36 bins để biểu diễn hết 360° của các hướng gradient. Các điểm xung quanh keypoint được thêm vào lược đồ bằng độ lớn gradient và cửa sổ tròn trọng số Gaussian.

Tìm đỉnh cao nhất trong lược đồ, các đỉnh còn lại mà đạt 80% so với đỉnh cao nhất, thì sẽ tạo keypoint ứng với hướng này.

1.6 Miêu tả đặc trưng

Tính độ lớn và hướng gradient của mỗi điểm xung quanh keypoint. Hàm Gaussian được dùng để gán trọng số độ lớn cho mỗi điểm. Kích thước vùng xung quanh keypoint là 16×16 , và được chia thành 4×4 vùng con.

Vector miêu tả đặc trưng chứa giá trị của tất cả lược đồ hướng. Vùng con 4×4 của lược đồ biểu diễn 8 hướng cho mỗi bin. Số thành phần của vector đặc trưng cho mỗi keypoint là $4 \times 4 \times 8 = 128$.

2 Thuật toán phát hiện góc Harris

Tính đạo hàm theo x và y của ảnh

$$I_x = G_\sigma^x * I; I_y = G_\sigma^y * I$$

Tính tích tại mỗi pixel của đạo hàm I_x và I_y

$$I_{x2} = I_x * I_x; I_{y2} = I_y * I_y; I_{xy} = I_x * I_y$$

Áp bộ lọc Gaussian lên I_{x2} , I_{y2} và I_{xy}

$$I_{xx} = G_{\sigma'} * I_{x2}; I_{yy} = G_{\sigma'} * I_{y2}; I_{xy} = G_{\sigma'} * I_{xy}$$

Tại mỗi pixel:

$$H(x, y) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$$

Gọi:

$$A = I_{xx}$$

$$B = I_{xy}$$

$$C = I_{yy}$$

$$M = \det(H) - k * \text{Trace}(H)^2 > \text{ngưỡng}$$

Tính nonmax suppression

Tại mỗi pixel, giá trị $M(x,y)$ không lớn hơn các pixel xung quanh (3×3) thì loại pixel này.

3 Thuật toán phát hiện Blob

Xây dựng không gian tỉ lệ (scale space):

Từ ảnh ban đầu tạo ra các ảnh mờ với mức độ khác nhau. Sau đó thay đổi kích thước ảnh ban đầu xuống một nữa, và tiếp tục sinh ra các ảnh mờ. Quá trình cứ lặp lại như thế.

Để tạo ra các ảnh mờ có thể áp dụng bộ lọc Laplacian. Có thể xấp xỉ Laplacian bởi Difference of Gaussians.

Phát hiện các cực đại (cực tiểu) cục bộ: blob là các vị trí mà có cực đại cục bộ không chỉ trong cùng ảnh (8 pixel lân cận) mà còn với các pixel trong hai ảnh tỉ lệ lân cận (2×9 pixel lân cận), như vậy tổng cộng $8 + 18$ pixel lân cận được xem xét.

4 Bài tập

- Viết chương trình tìm đặc trưng SIFT.
- Viết chương trình phát hiện góc Harris
- Viết chương trình phát hiện Blob.

Tài liệu tham khảo

- [1] David G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, 60, 2 (2004), pp. 91-110.
- [2] C. Harris and M. Stephens (1988). "A combined corner and edge detector". Proceedings of the 4th Alvey Vision Conference. pp. 147–151.

[CTT451] - [Nhập môn Thị giác Máy tính]

Tháng 4/2013

SPLIT AND MERGE ALGORITHM

Bộ môn TGMT và KH Rô-bốt
Khoa Công nghệ thông tin
ĐH Khoa học tự nhiên TP HCM



MỤC LỤC

MỤC LỤC	1
1 Các thuật toán Split và Merge	3
2 Thuật toán WaterShed	3
3 Bài tập.....	5
Tài liệu tham khảo	6

1 Các thuật toán Split và Merge

Cho R biểu diễn toàn bộ vùng của ảnh, và một logic vị từ P.

- **Split** (chia nhỏ): với mỗi vùng R_i , mà $P(R_i) = \text{FALSE}$ thì chia nhỏ R_i . Chia nhỏ các vùng cho đến khi các vùng R_i mà $P(R_i) = \text{TRUE}$.
- **Merge** (nhóm lại): nhóm các vùng lân cận R_i và R_k nếu $P(R_j \cup R_k) = \text{TRUE}$.

Một số thuật toán thể hiện tư tưởng Split và Merge này như:

- Watershed
- Region splitting
- Region merging
- Graph-based segmentation
- Probabilistic aggregation

2 Thuật toán WaterShed

Hàm cho thuật toán Watershed như sau:

```
void cvWatershed(const CvArr* image, CvArr* markers);
```

Với:

image là ảnh màu 8 bit

maker là ảnh kênh đơn (IPL_DEPTH_32S) có cùng kích thước với image. Maker có thể có được từ mặt nạ nhị phân sử dụng cvFindContours() và cvDrawContours(). Tham khảo đoạn chương trình FindContours và DrawContours sau:

```
#include "cv.h"
#include "highgui.h"

int main( int argc, char** argv )
{
    IplImage* src;
    // the first command line parameter must be file name of binary
    // (black-n-white) image
    if( argc == 2 && (src=cvLoadImage(argv[1], 0))!= 0 )
    {
        IplImage* dst = cvCreateImage( cvGetSize(src), 8, 3 );
        CvMemStorage* storage = cvCreateMemStorage(0);
        CvSeq* contour = 0;

        cvThreshold( src, src, 1, 255, CV_THRESH_BINARY );
        cvNamedWindow( "Source", 1 );
```

```
cvShowImage( "Source", src );

cvFindContours( src, storage, &contour, sizeof(CvContour),
                CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE );
cvZero( dst );

for( ; contour != 0; contour = contour->h_next )
{
    CvScalar color = CV_RGB( rand()&255, rand()&255, rand()&255 );
    /* replace CV_FILLED with 1 to see the outlines */
    cvDrawContours( dst, contour, color, color, -1, CV_FILLED, 8
);
}

cvNamedWindow( "Components", 1 );
cvShowImage( "Components", dst );
cvWaitKey(0);
}
```

3 Bài tập

- Cài đặt một số thuật toán Split và Merge:
 - o Watershed
 - o Region splitting
 - o Region merging
 - o Graph-based segmentation
 - o Probabilistic aggregation

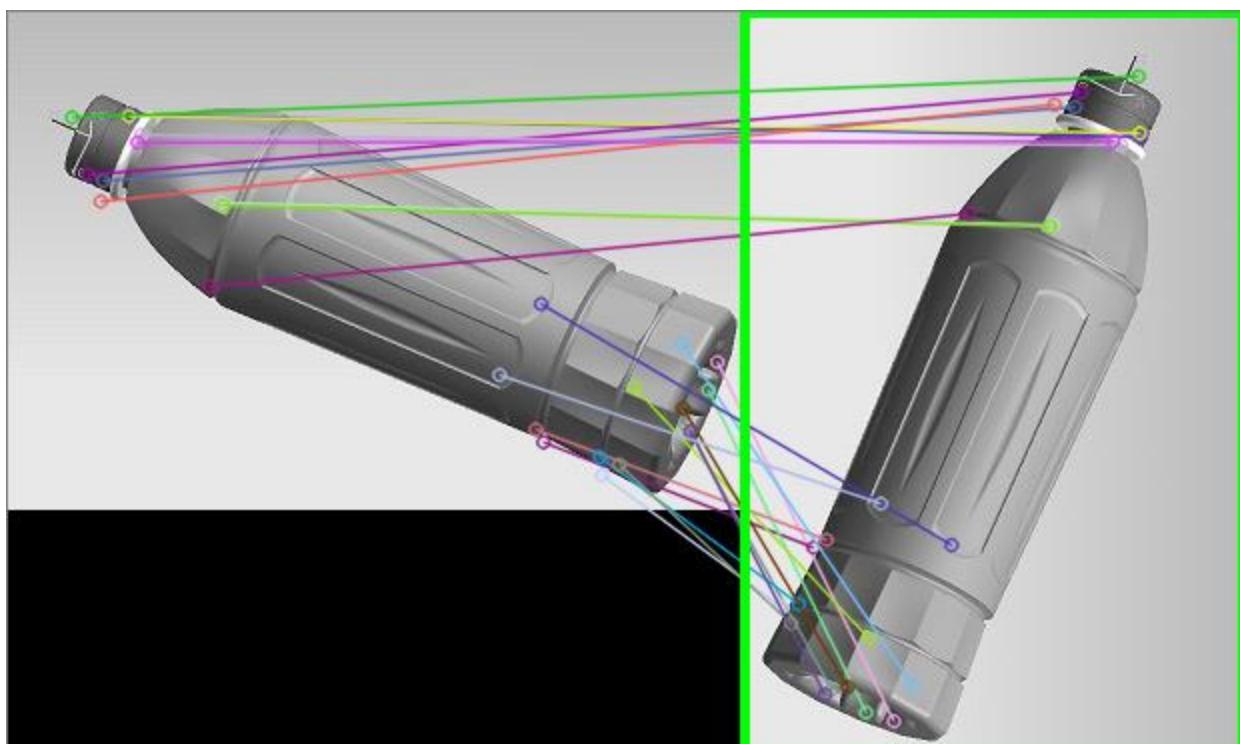
Tài liệu tham khảo

[1] Computer Vision: Algorithms and Applications, book draft by Richard Szeliski.

[CTT451] - [Nhập môn Thị giác Máy tính]

Tháng 5/2013

COMPUTATION HOMOGRAPHY



Bộ môn TGMT và KH Rô-bốt
Khoa Công nghệ thông tin
ĐH Khoa học tự nhiên TP HCM



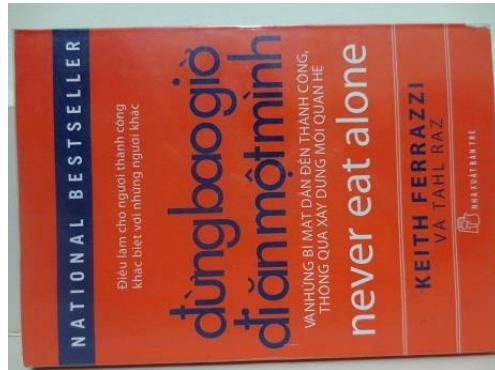
MỤC LỤC

MỤC LỤC	1
1 Ứng dụng Homography	3
2 Chương trình.....	3

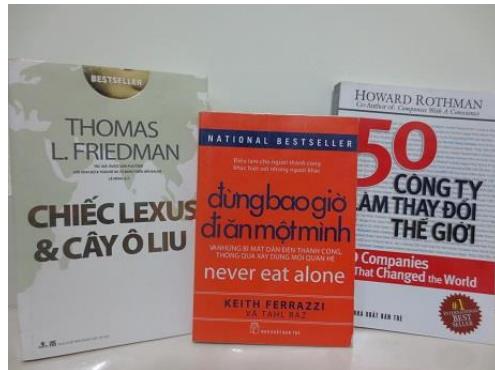
1 Ứng dụng Homography

Tìm một đối tượng đã biết trước

Cho đối tượng sau:



Tìm đối tượng trên trong hình sau:



Bước 1: phát hiện các keypoints sử dụng đặc trưng SURF

Bước 2: từ tập các keypoints được phát hiện trên, tính toán các vector đặc trưng (vector miêu tả).

Bước 3: đối sánh các vector miêu tả sử dụng FLANN matcher.

- Vẽ các đối sánh tốt nhất: là các đối sánh nhỏ hơn ngưỡng
- Xác định đối tượng trong ảnh

2 Chương trình

```
#include "stdio.h"
#include "iostream"

#include "opencv/cv.h"
#include "opencv/highgui.h"
```

```

#include "opencv2/nonfree/features2d.hpp"
#include "opencv2/legacy/legacy.hpp"
#include "opencv2/core/core.hpp"

using namespace cv;

/** @function main */
int main( int argc, char** argv )
{
    Mat img_object = imread( "C:\\\\OPENCV245\\\\Test\\\\3.jpg", CV_LOAD_IMAGE_GRAYSCALE );
    Mat img_scene = imread( "C:\\\\OPENCV245\\\\Test\\\\5.jpg", CV_LOAD_IMAGE_GRAYSCALE );

    if( !img_object.data || !img_scene.data )
    { std::cout<< " --(!) Error reading images " << std::endl; return -1; }

    //-- Step 1: Detect the keypoints using SURF Detector
    int minHessian = 400;
    //SURF surf (minHessian);
    SurfFeatureDetector detector(minHessian, 4, 2, true, false );

    std::vector<KeyPoint> keypoints_object, keypoints_scene;

    detector.detect( img_object, keypoints_object );
    detector.detect( img_scene, keypoints_scene );

    //-- Step 2: Calculate descriptors (feature vectors)
    SurfDescriptorExtractor extractor;

    Mat descriptors_object, descriptors_scene;

    extractor.compute( img_object, keypoints_object, descriptors_object );
    extractor.compute( img_scene, keypoints_scene, descriptors_scene );

    //-- Step 3: Matching descriptor vectors using FLANN matcher
    FlannBasedMatcher matcher;
    std::vector< DMatch > matches;
    matcher.match( descriptors_object, descriptors_scene, matches );

    double max_dist = 0; double min_dist = 100;

    //-- Quick calculation of max and min distances between keypoints
    for( int i = 0; i < descriptors_object.rows; i++ )
    { double dist = matches[i].distance;
      if( dist < min_dist ) min_dist = dist;
      if( dist > max_dist ) max_dist = dist;
    }

    printf("-- Max dist : %f \n", max_dist );
    printf("-- Min dist : %f \n", min_dist );

    //-- Draw only "good" matches (i.e. whose distance is less than 3*min_dist )
    std::vector< DMatch > good_matches;

    for( int i = 0; i < descriptors_object.rows; i++ )
    { if( matches[i].distance < 3*min_dist )
      { good_matches.push_back( matches[i] ); }
    }

```

```

}

Mat img_matches;
drawMatches( img_object, keypoints_object, img_scene, keypoints_scene,
    good_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
    vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );

<!-- Localize the object
std::vector&lt;Point2f&gt; obj;
std::vector&lt;Point2f&gt; scene;

for( int i = 0; i &lt; good_matches.size(); i++ )
{
    <!-- Get the keypoints from the good matches
    obj.push_back( keypoints_object[ good_matches[i].queryIdx ].pt );
    scene.push_back( keypoints_scene[ good_matches[i].trainIdx ].pt );
}

Mat H = findHomography( obj, scene, CV_RANSAC );

<!-- Get the corners from the image_1 ( the object to be "detected" )
std::vector&lt;Point2f&gt; obj_corners(4);
obj_corners[0] = cvPoint(0,0); obj_corners[1] = cvPoint( img_object.cols, 0 );
obj_corners[2] = cvPoint( img_object.cols, img_object.rows ); obj_corners[3] =
cvPoint( 0, img_object.rows );
std::vector&lt;Point2f&gt; scene_corners(4);

perspectiveTransform( obj_corners, scene_corners, H);

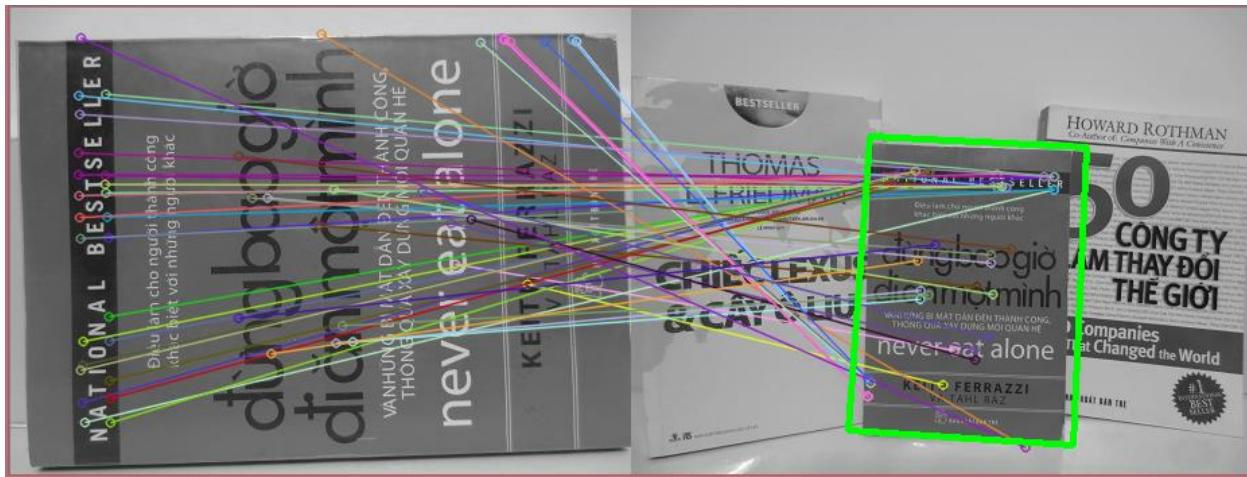
<!-- Draw lines between the corners (the mapped object in the scene - image_2 )
line( img_matches, scene_corners[0] + Point2f( img_object.cols, 0 ),
scene_corners[1] + Point2f( img_object.cols, 0 ), Scalar(0, 255, 0), 4 );
    line( img_matches, scene_corners[1] + Point2f( img_object.cols, 0 ),
scene_corners[2] + Point2f( img_object.cols, 0 ), Scalar( 0, 255, 0 ), 4 );
    line( img_matches, scene_corners[2] + Point2f( img_object.cols, 0 ),
scene_corners[3] + Point2f( img_object.cols, 0 ), Scalar( 0, 255, 0 ), 4 );
    line( img_matches, scene_corners[3] + Point2f( img_object.cols, 0 ),
scene_corners[0] + Point2f( img_object.cols, 0 ), Scalar( 0, 255, 0 ), 4 );

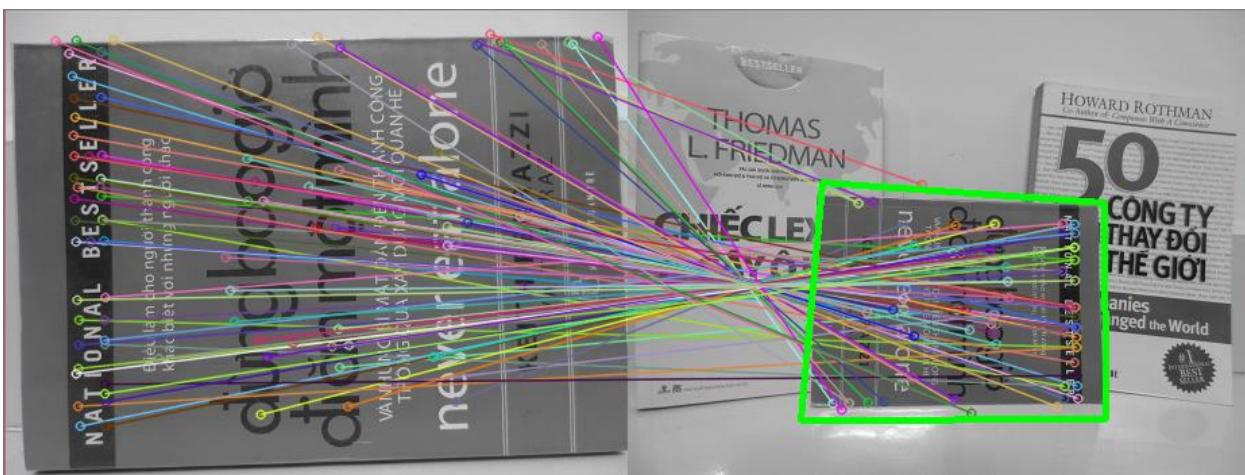
<!-- Show detected matches
imshow( "Good Matches &amp; Object detection", img_matches );

waitKey(0);
return 0;
}
</pre>

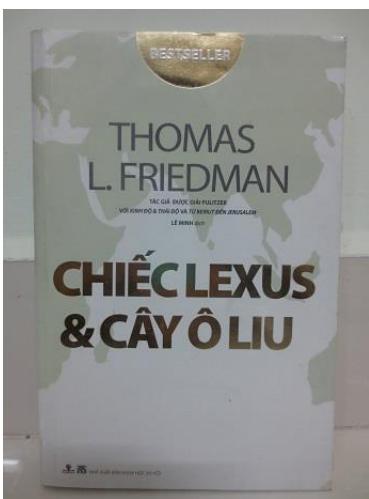
```

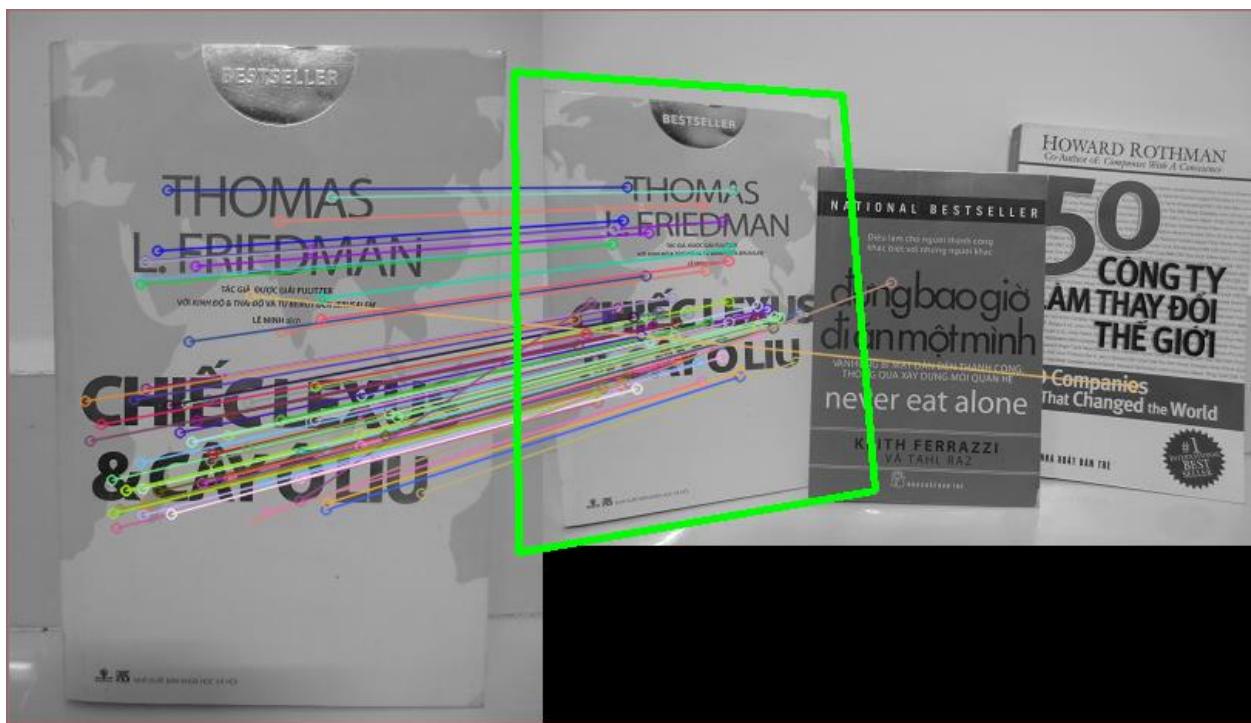
Chạy chương trình





Đối tượng thứ 2:





[CTT451] - [Nhập môn Thị giác Máy tính]

Tháng 5/2013

Kalman và Particle Filter

Bộ môn TGMT và KH Rô-bốt
Khoa Công nghệ thông tin
ĐH Khoa học tự nhiên TP HCM



MỤC LỤC

MỤC LỤC	1
1 Ví dụ sử dụng Kalman Filter:	3
2 Mean-Shift.....	6
3 Phát hiện và theo vết khuôn mặt trong video	6

1 Ví dụ sử dụng Kalman Filter:

Giả sử chúng ta có một điểm chuyển động xung quanh một vòng tròn, như một chiếc xe chạy trên đường đua. Chiếc xe chuyển động với vận tốc là một hằng số. Xác định vị trí chiếc xe sử dụng phương pháp theo vết.

```
#include "opencv2/video/tracking.hpp"
#include "opencv2/highgui/highgui.hpp"

#include <stdio.h>

using namespace cv;

static inline Point calcPoint(Point2f center, double R, double angle)
{
    return center + Point2f((float)cos(angle), (float)-sin(angle))*(float)R;
}

int main(int, char**)
{
    Mat img(500, 500, CV_8UC3);
    KalmanFilter KF(2, 1, 0);
    Mat state(2, 1, CV_32F); /* (phi, delta_phi) */
    Mat processNoise(2, 1, CV_32F);
    Mat measurement = Mat::zeros(1, 1, CV_32F);
    char code = (char)-1;

    for(;;)
    {
        randn( state, Scalar::all(0), Scalar::all(0.1) );
        KF.transitionMatrix = *(Mat<float>(2, 2) << 1, 1, 0, 1);

        setIdentity(KF.measurementMatrix);
        setIdentity(KF.processNoiseCov, Scalar::all(1e-5));
        setIdentity(KF.measurementNoiseCov, Scalar::all(1e-1));
        setIdentity(KF.errorCovPost, Scalar::all(1));

        randn(KF.statePost, Scalar::all(0), Scalar::all(0.1));

        for(;;)
        {
            Point2f center(img.cols*0.5f, img.rows*0.5f);
            float R = img.cols/3.f;
            double stateAngle = state.at<float>(0);
            Point statePt = calcPoint(center, R, stateAngle);

            Mat prediction = KF.predict();
            double predictAngle = prediction.at<float>(0);
            Point predictPt = calcPoint(center, R, predictAngle);

            randn( measurement, Scalar::all(0),
Scalar::all(KF.measurementNoiseCov.at<float>(0)));

            // generate measurement
            measurement += KF.measurementMatrix*state;

            double measAngle = measurement.at<float>(0);
```

```

        Point measPt = calcPoint(center, R, measAngle);

        // plot points
#define drawCross( center, color, d )      \
line( img, Point( center.x - d, center.y - d ),           \
Point( center.x + d, center.y + d ), color, 1, CV_AA, 0 ); \
line( img, Point( center.x + d, center.y - d ),           \
Point( center.x - d, center.y + d ), color, 1, CV_AA, 0 )

img = Scalar::all(0);
drawCross( statePt, Scalar(255,255,255), 3 );
drawCross( measPt, Scalar(0,0,255), 3 );
drawCross( predictPt, Scalar(0,255,0), 3 );
line( img, statePt, measPt, Scalar(0,0,255), 3, CV_AA, 0 );
line( img, statePt, predictPt, Scalar(0,255,255), 3, CV_AA, 0 );

if(theRNG().uniform(0,4) != 0
    KF.correct(measurement);

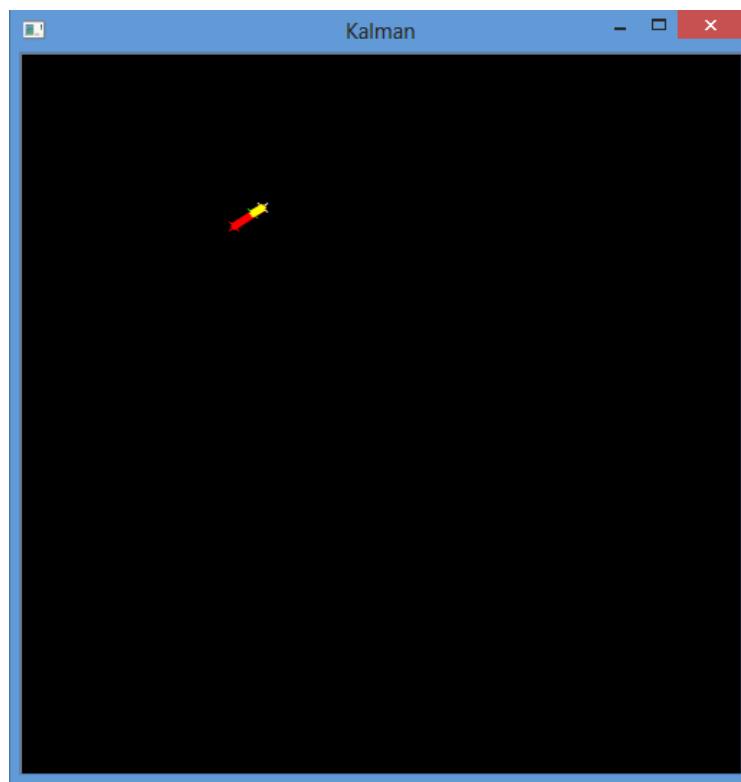
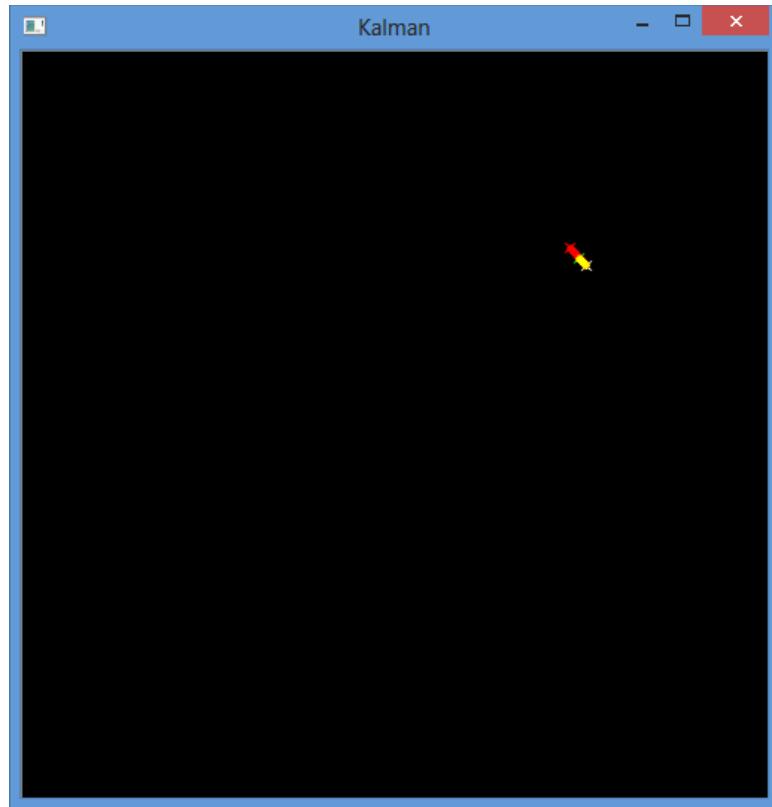
randn( processNoise, Scalar(0),
Scalar::all(sqrt(KF.processNoiseCov.at<float>(0, 0))));
state = KF.transitionMatrix*state + processNoise;

imshow( "Kalman", img );
code = (char)waitKey(100);

if( code > 0 )
    break;
}
if( code == 27 || code == 'q' || code == 'Q' )
    break;
}

return 0;
}

```



2 Mean-Shift

```
int cvMeanShift(  
    const CvArr* prob_image,  
    CvRect window,  
    CvTermCriteria criteria,  
    CvConnectedComp* comp  
) ;
```

prob_image biểu diễn vị trí mật độ có thể của dữ liệu.

window cửa sổ kernel ban đầu

criteria: quyết định số lần lặp của window có thể dừng

comp chứa tọa độ tìm kiếm của window.

3 Phát hiện và theo vết khuôn mặt trong video

```
#include "opencv/cv.h"  
#include "opencv/cxcore.h"  
#include "opencv/highgui.h"  
#include <iostream>  
#include <stdlib.h>  
#include <time.h>  
  
using namespace std;  
  
// Create memory for calculations  
static CvMemStorage* storage = 0;  
  
// Create a new Haar classifier  
static CvHaarClassifierCascade* cascade = 0;  
  
// Create a string that contains the cascade name  
const char* cascade_name = "haarcascade_frontalface.xml";  
/*      "haarcascade_profileface.xml";*/  
  
void detect_and_draw(IplImage* img);  
  
bool writeActive;  
IplImage* im;  
IplImage* imFace;  
IplImage* imR;  
IplImage* imG;  
IplImage* imB;
```

```

struct face
{
    CvRect rectangle;
    int dx, dy, confidence;
    float weight;
    CvHistogram* rHistogram;
    CvHistogram* gHistogram;
    CvHistogram* bHistogram;
};

vector<face> allFaces;

CvHistogram* getHistogram(IplImage* im, CvRect r)
{
    cvSetImageROI(im, r);
    int numBins = 64;
    float range[] = {0.0,255.0};
    float* ranges[] = {range};
    CvHistogram* h = cvCreateHist(1, &numBins, CV_HIST_ARRAY, ranges);
    cvCalcHist(&im, h);
    /*for(int binIter=0; binIter<numBins; binIter++)
    {
        cout<<cvQueryHistValue_1D(h, binIter)<< " ";
    }
    cout<<endl;*/
    cvResetImageROI(im);
    return h;
}

vector<face> getSamples(face f, int predX, int predY, int predW, int predH, int numSamples, float searchSTD, int wLimit, int hLimit)
{
    vector<face> samples;

    float u1,u2,u3,u4,n1,n2,n3,n4,probability,scale,rCorr,gCorr,bCorr,likelihood;
    int newWidth,newHeight;

    //generate random samples
    for(int randGenIter=0; randGenIter<numSamples; randGenIter++)
    {
        //generate two random uniformly distributed numbers
        u1 = ((float)rand())/(RAND_MAX);
        u2 = ((float)rand())/(RAND_MAX);
        u3 = ((float)rand())/(RAND_MAX);
        u4 = ((float)rand())/(RAND_MAX);
        //get normally distributed random numbers using box-muller transform (has
mean 0 and std 1)
        n1 = sqrt(-2*log(u1)) * cos(2*3.14159265359*u2);
        n2 = sqrt(-2*log(u1)) * sin(2*3.14159265359*u2);
        n3 = sqrt(-2*log(u3)) * sin(2*3.14159265359*u4);

        //probability = pow(2.71828,-0.5*n1*n1)/sqrt(2*3.14159265359) *
pow(2.71828,-0.5*n2*n2)/sqrt(2*3.14159265359);
        //probability *= pow(2.71828,-0.5*n3*n3)/sqrt(2*3.14159265359);

        //make std dev one third of face dimensions and mean at the predicted
position
    }
}

```

```

n1*=f.rectangle.width * searchSTD;
n1+=predX;
n2*=f.rectangle.height * searchSTD;
n2+=predY;

n3=1;
/*n3*=0.05;
n3+=1;
n3 = MIN(1.3, MAX(0.7,n3) );/**/

scale = n3;
newWidth = predW * scale;
//scale = n4;
newHeight = predH * scale;

if (n1>0 && n2>0 && n1<wLimit-newWidth && n2<hLimit-newHeight)//if
randomized position is on the image
{
    //declare a face at the location
    face newFace;
    newFace.rectangle = cvRect(n1,n2,newWidth,newHeight);
    newFace.rHistogram = getHistogram(imR, newFace.rectangle);
    newFace.gHistogram = getHistogram(imG, newFace.rectangle);
    newFace.bHistogram = getHistogram(imB, newFace.rectangle);
    newFace.dx=0;
    newFace.dy=0;

    //calculate likelihood / weight
    //cout<< " <<newFace.rectangle.x<< " <<newFace.rectangle.y<<
" <<newFace.rectangle.width<< " <<newFace.rectangle.height<<endl;
    //cout<< " <<allFaces.at(faceIter).rectangle.x<<
" <<allFaces.at(faceIter).rectangle.y<< " <<allFaces.at(faceIter).rectangle.width<<
" <<allFaces.at(faceIter).rectangle.height<<endl;
    rCorr = cvCompareHist(newFace.rHistogram, f.rHistogram,
CV_COMP_CORREL);
    gCorr = cvCompareHist(newFace.gHistogram, f.gHistogram,
CV_COMP_CORREL);
    bCorr = cvCompareHist(newFace.bHistogram, f.bHistogram,
CV_COMP_CORREL);

    likelihood = (rCorr*0.4 + gCorr*0.3 + bCorr*0.3);

    newFace.weight = pow(2.718281828, -16.0 * (1-likelihood));
    //cout<<newFace.weight<<endl;
    samples.push_back(newFace);
}
}

return samples;
}

vector<face> resample(vector<face> samples, face f, int wLimit, int hLimit)
{
    float totalWeight=0;
    for(int sampleIter=0; sampleIter<samples.size(); sampleIter++)

```

```

{
    //cout<<samples.at(sampleIter).weight<<endl;
    totalWeight+=samples.at(sampleIter).weight;
}
vector<face> resamples;
vector<face> allResamples;
int numSamplesToDraw;
for(int sampleIter=0; sampleIter<samples.size(); sampleIter++)
{
    resamples.clear();
    numSamplesToDraw = (int)((samples.at(sampleIter).weight/totalWeight) *
samples.size())+0.5;

    //predicted position
    int predX = samples.at(sampleIter).rectangle.x;
    int predY = samples.at(sampleIter).rectangle.y;

    resamples = getSamples(f, predX, predY,
samples.at(sampleIter).rectangle.width, samples.at(sampleIter).rectangle.height,
numSamplesToDraw, 0.1, wLimit, hLimit);
    //add resamples to the vector of all resamples
    for(int resampleIter=0; resampleIter<resamples.size(); resampleIter++)
    {
        allResamples.push_back(resamples.at(resampleIter));
    }
}
return allResamples;
}

void drawFaces()
{
    //copy the image and draw the faces
    cvCopy(im, imFace);

    CvPoint pt1, pt2;
    CvScalar rectColor;
    //draw the faces
    for(int faceIter = 0; faceIter < allFaces.size(); faceIter++ )
    {
        pt1.x = allFaces.at(faceIter).rectangle.x;
        pt2.x = pt1.x + allFaces.at(faceIter).rectangle.width;
        pt1.y = allFaces.at(faceIter).rectangle.y;
        pt2.y = pt1.y + allFaces.at(faceIter).rectangle.height;

        rectColor = cvScalar(0,0,0,0);
        cvRectangle( imFace, pt1, pt2, rectColor, 3, 8, 0 );
        rectColor = cvScalar(0,255,0,0);
        cvRectangle( imFace, pt1, pt2, rectColor, 1, 8, 0 );
    }

    cvShowImage("Faces",imFace);
}

int main( int argc, char** argv )
{
    //initialize random seed
    srand ( time(NULL) );
}

```

```

cout<<"wait..";
cvWaitKey(3000);
cout<<"go"<<endl;

// Load the HaarClassifierCascade
cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );
// Allocate the memory storage
storage = cvCreateMemStorage(0);

const char* filename = "data//ForrestGump.avi";
//CvCapture* capture = cvCreateFileCapture(filename);
CvCapture* capture = cvCaptureFromCAM(0);

if(!capture) cout << "No camera detected" << endl;
cvNamedWindow( "result", 1 );
IplImage* iplImg = cvQueryFrame( capture );
cvShowImage("result", iplImg);
cvWaitKey(0);

int i = cvGrabFrame(capture);

im = cvRetrieveFrame(capture);

CvVideoWriter* writer = cvCreateVideoWriter("out.mp4", CV_FOURCC('F','M','P','4'),
10, cvSize(im->width,im->height), 1);

imFace = cvCloneImage(im);
IplImage* imCopy = cvCloneImage(im);

//allocate some images used to extract a skin likelihood map
IplImage* imHSV = cvCreateImage(cvSize(im->width,im->height),IPL_DEPTH_8U, 3);
IplImage* skin = cvCreateImage(cvSize(im->width,im->height),IPL_DEPTH_8U, 1);

imR = cvCreateImage(cvSize(im->width,im->height),IPL_DEPTH_8U, 1);
imG = cvCreateImage(cvSize(im->width,im->height),IPL_DEPTH_8U, 1);
imB = cvCreateImage(cvSize(im->width,im->height),IPL_DEPTH_8U, 1);

IplImage* sampling = cvCreateImage(cvSize(im->width,im->height),IPL_DEPTH_32F, 1);

int frame = 0;

int keyPressed = -1;

writeActive=true;

while(i!=0 && (keyPressed== -1 || keyPressed=='f' || keyPressed=='w'))
{
    cvShowImage("Given",im);

    cvCvtColor(im, imHSV, CV_BGR2HSV);
    CvScalar pixRGB, pixHSV;
    float cb, cr;

    //separate into channels
    cvSetImageCOI(im,1);
    cvCopy(im,imB);
    cvSetImageCOI(im,2);
    cvCopy(im,imG);

```

```

cvSetImageCOI(im,3);
cvCopy(im,imR);
cvSetImageCOI(im,0);

if(keyPressed=='w')
{
    writeActive=!writeActive;
}

if (frame==0 || allFaces.size()==0 || keyPressed=='f') //detect faces
{
    // Clear the memory storage which was used before
    cvClearMemStorage( storage );

    // There can be more than one face in an image. So create a growable
sequence of faces.
    // Detect the objects and store them in the sequence
    CvSeq* faces = cvHaarDetectObjects( im, cascade, storage, 1.1, 2,
CV_HAAR_DO_CANNY_PRUNING, cvSize(40, 40) );

    for(int currentFace=0; currentFace<faces->total; currentFace++)
    {
        //get rectangle bounding first face
        CvRect* r = (CvRect *)cvGetSeqElem( faces, currentFace );

        face newFace;
        newFace.rectangle = *r;
        newFace.confidence = 2;
        newFace.dx = 0;
        newFace.dy = 0;

        //find the total amount of skin-colored pixels in the face
        float skinSum=0;
        for (int y=r->y; y<r->y+r->height; y++)
        {
            for (int x=r->x; x<r->x+r->width; x++)
            {
                pixRGB = cvGet2D(im,y,x);
                pixHSV = cvGet2D(imHSV,y,x);
                cb = 0.148*pixRGB.val[2] - 0.291*pixRGB.val[1]
+ 0.439*pixRGB.val[0] + 128;
                cr = 0.439*pixRGB.val[2] - 0.368*pixRGB.val[1]
- 0.071*pixRGB.val[0] + 128;
                if ( ( pixHSV.val[0]>245 || pixHSV.val[0]<25.5)
&& 140<=cr && cr<=165 && 140<=cb && cb<=195)
                {
                    skinSum++;
                }
            }
        }
        //if less than 30% skin, this face doesnt count
        if (skinSum / (r->width*r->height) < 0.3)
        {
            //break;
        }

        //check to see if this face is roughly matching an existing
face
    }
}

```

```

        bool matchesExisting=false;

        for(int faceIter = 0; faceIter < allFaces.size(); faceIter++ )
        {
            //find the width and height of the region of overlap
            int overlapWidth, overlapHeight;
            if ( newFace.rectangle.x <
allFaces.at(faceIter).rectangle.x)
            {
                overlapWidth = min( newFace.rectangle.x +
newFace.rectangle.width - allFaces.at(faceIter).rectangle.x,
allFaces.at(faceIter).rectangle.width);
            }else{
                overlapWidth = min(
allFaces.at(faceIter).rectangle.x + allFaces.at(faceIter).rectangle.width -
newFace.rectangle.x, newFace.rectangle.width);
            }
            if ( newFace.rectangle.y <
allFaces.at(faceIter).rectangle.y)
            {
                overlapHeight = min( newFace.rectangle.y +
newFace.rectangle.height - allFaces.at(faceIter).rectangle.y,
allFaces.at(faceIter).rectangle.height);
            }else{
                overlapHeight = min(
allFaces.at(faceIter).rectangle.y + allFaces.at(faceIter).rectangle.height -
newFace.rectangle.y, newFace.rectangle.height);
            }

            //if region of overlap is greater than 60% of larger
rectangle, then faces are the same
            if ( ((float)overlapWidth*overlapHeight)/(max(
newFace.rectangle.width*newFace.rectangle.height,
allFaces.at(faceIter).rectangle.width*allFaces.at(faceIter).rectangle.height) )>0.6)
            {
                matchesExisting = true;
                allFaces.at(faceIter).confidence = min( 4,
allFaces.at(faceIter).rectangle =
allFaces.at(faceIter).dx = newFace.dx;
allFaces.at(faceIter).dy = newFace.dy;
allFaces.at(faceIter).rHistogram =
allFaces.at(faceIter).gHistogram =
allFaces.at(faceIter).bHistogram =
break;
            }
        }

        if (!matchesExisting) //if its new, add it to our vector
        {
            newFace.rHistogram = getHistogram(imR,
newFace.rectangle);
            newFace.gHistogram = getHistogram(imG,
newFace.rectangle);
        }
    }
}

```

```

newFace.bHistogram = getHistogram(imB,
newFace.rectangle);
        allFaces.push_back(newFace);
    }
}

//subtract 1 from confidence of all faces
for(int faceIter = 0; faceIter < allFaces.size(); faceIter++ )
{
    allFaces.at(faceIter).confidence--;
    if (allFaces.at(faceIter).confidence < 1) //if confidence
gone, remove it
    {
        allFaces.erase(allFaces.begin()+faceIter,
allFaces.begin()+faceIter+1);
        faceIter--;
    }
}

if(allFaces.size()>0) //track faces
{
    cvSet(sampling,cvScalar(0));
    for(int faceIter = 0; faceIter < allFaces.size(); faceIter++ )
//first face only for now
    {
        //predicted position
        int predX = allFaces.at(faceIter).rectangle.x +
allFaces.at(faceIter).dx;
        int predY = allFaces.at(faceIter).rectangle.y +
allFaces.at(faceIter).dy;

        vector<face> samples = getSamples(allFaces.at(faceIter),
predX, predY, allFaces.at(faceIter).rectangle.width,
allFaces.at(faceIter).rectangle.height, 100, 0.2, im->width, im->height);

        //do importance resampling a number of times
        for(int resampling=0; resampling<3; resampling++)
        {
            samples = resample(samples, allFaces.at(faceIter), im-
>width, im->height);
        }

        int bestIdx=0;
        float bestWeight=0;

        //generate random samples
        for(int sampleIter=0; sampleIter<samples.size(); sampleIter++)
        {
            if (samples.at(sampleIter).weight > bestWeight)
            {
                bestWeight = samples.at(sampleIter).weight;
                bestIdx = sampleIter;
            }
            //cvSet2D(sampling,
samples.at(sampleIter).rectangle.y,samples.at(sampleIter).rectangle.x,
cvScalar(samples.at(sampleIter).weight));
        }
}

```

```

        //move to best sample
        allFaces.at(faceIter).dx = samples.at(bestIdx).rectangle.x -
allFaces.at(faceIter).rectangle.x;
        allFaces.at(faceIter).dy = samples.at(bestIdx).rectangle.y -
allFaces.at(faceIter).rectangle.y;
        allFaces.at(faceIter).rectangle =
samples.at(bestIdx).rectangle;
        /*cvCopyHist(samples.at(bestIdx).rHistogram,
&allFaces.at(faceIter).rHistogram);
        cvCopyHist(samples.at(bestIdx).gHistogram,
&allFaces.at(faceIter).gHistogram);
        cvCopyHist(samples.at(bestIdx).bHistogram,
&allFaces.at(faceIter).bHistogram);*/
    }
    drawFaces();
    //cvCvtColor(imFace, imCopy, CV_BGR2RGB);
    //if(writeActive)
    //cvWriteFrame(writer, imFace);
    //scaleAndShow(sampling,"Sampling");
    //cvWaitKey(1);
}

i = cvGrabFrame(capture);
im = cvRetrieveFrame(capture);
frame++;

keyPressed = cvWaitKey(100);
}

cvReleaseImage(&im);
cvReleaseImage(&imCopy);
cvReleaseImage(&imHSV);
cvReleaseImage(&skin);
cvReleaseCapture(&capture);
cvReleaseVideoWriter(&writer);
return 0;
}

```

