OpenCV

Basics

Basics

- OpenCV header file
- OpenCV namespace
- OpenCV basic structures
  - Primitive data types
  - Point_
  - Size_
  - Vec
  - Scalar_
  - Mat

OpenCV Header File

- #include <opencv2/opencv.hpp>
- .hpp is a convention for C++ language header files
- OpenCV has a modular structure that includes several shared or static libraries
- opencv.hpp is like an umbrella file that includes some of the basic modules:
  - core: core functionality
  - highgui: high-level GUI
  - imgproc: image processing
  - ……
- It is ok to manually include each individual module
  - #include <opencv2/core/core.hpp>

OpenCV Namespace

- All the OpenCV classes and functions are placed into the cv namespace. So in your code,
  - Use the **cv::** specifier, or
  - Use the **using namespace cv;** directive.
- Some of the current or future OpenCV names may conflict with C++ STL or other libraries. In this case, use the explicit namespace specifiers to resolve the name conflicts:
  - cv::log vs std::log
  - cv::max vs std::max
  - ……

OpenCV Basic Structures

- Primitive data types:
  - Unsigned char
  - Bool
  - Signed char
  - Unsigned short
  - Signed short
  - Int
  - Float
  - Double
  - Or a tuple of values of one of these types

OpenCV Basic Structures

- Point_, template class for 2D points
  - Members: x, y
  - Type aliases:

```
typedef Point_<int> Point2i;
typedef Point2i Point;
typedef Point_<float> Point2f;
typedef Point_<double> Point2d;
```

  - Examples:

```
Point2f a(0.3f, 0.f), b(0.f, 0.4f);
Point pt = (a + b)*10.f;
cout << pt.x << ", " << pt.y << endl;
```

```
pt1 = pt2 + pt3;
pt1 = pt2 - pt3;
pt1 = pt2 * a;
pt1 = a * pt2;
pt1 += pt2;
pt1 -= pt2;
pt1 *= a;
double value = norm(pt); // L2 norm
pt1 == pt2;
pt1 != pt2;
```

OpenCV Basic Structures

- Size_, template class for specifying the size of an image or rectangle.
  - Members: width, height
  - Type aliases:

```
typedef Size_<int> Size2i;
typedef Size2i Size;
typedef Size_<float> Size2f;
```

  - Example:
    - cv::Size sz(640, 480); // a VGA image size

OpenCV Basic Structures

- Vec, template class for short numerical vectors
  - Commonly used in OpenCV to describe pixel types of multi-channel arrays. See Mat for examples
  - Use operator[] to access the elements of Vec
  - Type aliases:

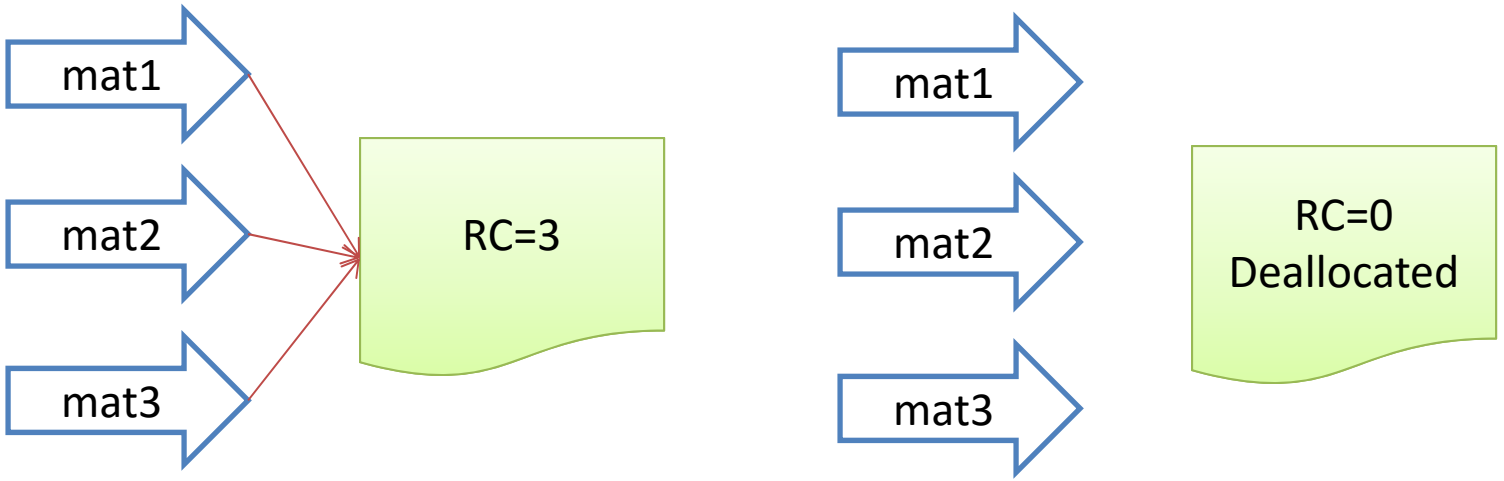
```
typedef Vec<uchar, 2> Vec2b;
typedef Vec<uchar, 3> Vec3b;
typedef Vec<uchar, 4> Vec4b;

typedef Vec<short, 2> Vec2s;
typedef Vec<short, 3> Vec3s;
typedef Vec<short, 4> Vec4s;

typedef Vec<int, 2> Vec2i;
typedef Vec<int, 3> Vec3i;
typedef Vec<int, 4> Vec4i;

typedef Vec<float, 2> Vec2f;
typedef Vec<float, 3> Vec3f;
typedef Vec<float, 4> Vec4f;
typedef Vec<float, 6> Vec6f;

typedef Vec<double, 2> Vec2d;
typedef Vec<double, 3> Vec3d;
typedef Vec<double, 4> Vec4d;
typedef Vec<double, 6> Vec6d;
```

OpenCV Basic Structures

- Scalar_, template class for a 4-element Vec
  - Widely used in OpenCV to pass pixel values
  - Type alias:
```
typedef Scalar_<double> Scalar;
```
  - Example:
    - cv::Scalar color(0, 0, 255); // Red. OpenCV's default is BGR

OpenCV Basic Structures

- Mat, OpenCV C++ n-dimensional dense array class
  - It is used to represent images and matrices.
  - It is a header that points to a block of data which is allocated either automatically or by the user.
  - It is managed via a reference-counting mechanism such that the array data is deallocated when no one points to it.



OpenCV Basic Structures

- Any primitive type can be defined by an identifier in the form:
  - CV_<bitDepth>{U|S|F}(C<numberOfChannels>)
  - CV_8U
  - CV_8UC2
  - CV_32FC4
  - ……

A Mapping of Type to Numbers in OpenCV

	C1	C2	C3	C4
CV_8U	0	8	16	24
CV_8S	1	9	17	25
CV_16U	2	10	18	26
CV_16S	3	11	19	27
CV_32S	4	12	20	28
CV_32F	5	13	21	29
CV_64F	6	14	22	30

http://ninghang.blogspot.com/2012/11/list-of-mat-type-in-opencv.html

OpenCV Mat Creation & Initialization

- Mat constructors mostly used:

```
Mat::Mat()

Mat::Mat(int rows, int cols, int type)

Mat::Mat(Size size, int type)

Mat::Mat(int rows, int cols, int type, const Scalar& s)

Mat::Mat(Size size, int type, const Scalar& s)
```

```
// make a 7x7 complex matrix filled with 1+3j.
Mat M(7,7,CV_32FC2,Scalar(1,3));
// and now turn M to a 100x60 15-channel 8-bit matrix.
// The old content will be deallocated
M.create(100,60,CV_8UC(15));
```

OpenCV Mat Creation & Initialization

```
// create a 100x100x100 8-bit array
int sz[] = {100, 100, 100};
Mat bigCube(3, sz, CV_8U, Scalar::all(0));
```

```
// create a double-precision identity martix and add it to M.
M += Mat::eye(M.rows, M.cols, CV_64F);
```

```
// create a 3x3 double-precision identity matrix
Mat M = (Mat_<double>(3,3) << 1, 0, 0, 0, 1, 0, 0, 0, 1);
```

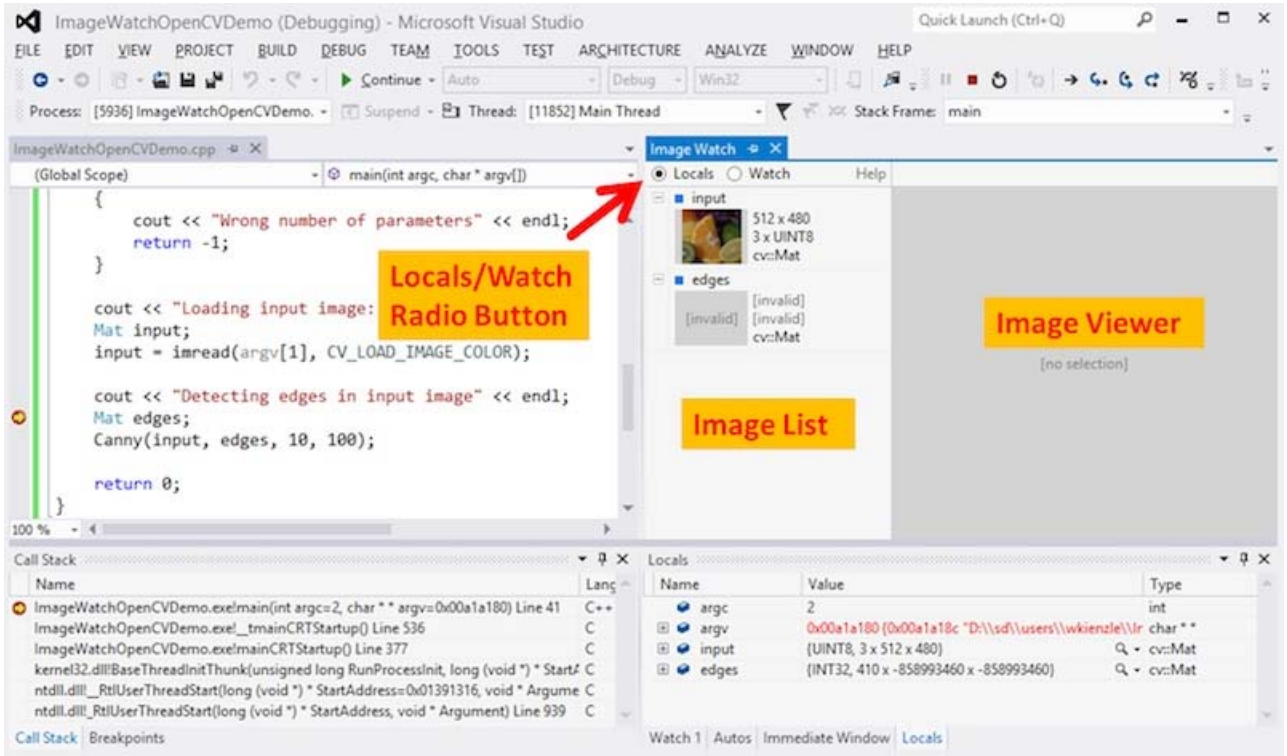
OpenCV treats matrices in row-major order, while MatLab is column-major.

OpenCV Mat

- Element access:
  - double elem = M.at<double>(row, col) * 6.0;
  - M.at<cv::Vec3b>(row, col) = cv::Vec3b(255,255,255);
- Row access:
  - M.row(3) = M.row(3) + M.row(5) * 3;
- Expressions:
  - Addition, subtraction, negation: `A+B, A-B, A+s, A-s, s+A, s-A, -A`
  - Scaling: `A*alpha`
  - Per-element multiplication and division: `A.mul(B), A/B, alpha/A`
  - Matrix multiplication: `A*B`
  - Transposition: `A.t()` (means A^T)

Optional for VS User: Image Watch

- Image Watch is a Visual Studio plug-in to visualize in-memory images while debugging an application.
- http://docs.opencv.org/3.1.0/d4/d14/tutorial_windows_visual_studio_image_watch.html



Example

```
/*
  OpenCV Basics
*/

// Include this if using Visual Studio
//#include <stdafx.h>

#include <opencv2/opencv.hpp>
int main(int argc, char* argv[])
{
    // Point_
    std::cout << std::endl << "Point_" << std::endl;
    cv::Point2f a(0.5f, 0.f), b(0.f, 0.6f);
    cv::Point pt = (a + b) * 10.f;
    std::cout << "a = (" << a.x << ", " << a.y << ")" << std::endl;
    std::cout << "pt = " << pt << std::endl;
    if (a == b)
        std::cout << "a==b" << std::endl;
    else
        std::cout << "a!=b" << std::endl;

    // Size_
    std::cout << std::endl << "Size_" << std::endl;
    cv::Mat image = // Read an image.
    cv::imread("C:/sw/opencv/sources/opencv/samples/data/lena.jpg");
    if (image.empty()) {
        std::cout << "Hey! Can't read the image!" << std::endl;
        system("PAUSE");
        return EXIT_FAILURE;
    }
    cv::Size sz(image.cols, image.rows);
    std::cout << "sz = " << sz << ", area = " << sz.area() << std::endl;
```

Example

```
// Vec
std::cout << std::endl << "Vec" << std::endl;
cv::Vec3d vx(1, 0, 0), vy(0, 1, 0), v, vz(0, 0, 0);
v = vx.dot(vy);    // Dot product
vz[2] = 1; // Modify the 3rd element to be 1
std::cout << "v = " << v << std::endl;
std::cout << "vz = " << vz << std::endl;

// Scalar_
std::cout << std::endl << "Scalar_" << std::endl;
cv::Scalar redScalar(0, 0, 255);
cv::rectangle(image, pt, pt * 80, redScalar, 3);
std::cout << "See the red rectangle on the output image." << std::endl;

// Mat
std::cout << std::endl << "Mat" << std::endl;
cv::Scalar blueScalar(255, 0, 0);
int typeImage = image.type();
cv::Mat doodle(sz, typeImage, blueScalar);
std::cout << "Image Type: " << typeImage
    << " (Look up the type mapping number in the lecture slides)" << std::endl;
if (typeImage == CV_8UC3)
    std::cout << "Image is an 8 bit unsigned char 3 channel matrix." << std::endl;
else
    std::cout << "Image is NOT an 8 bit unsigned char 3 channel matrix." << std::endl;
```

Example

```
    cv::Vec3b greenVec(0, 255, 0);
    // use row & col to access an element
    for (int row = 100; row < 200; row++)
        for (int col = 100; col < 200; col++)
            doodle.at<cv::Vec3b>(row, col) = greenVec;

    cv::Vec3b orangeVec(0, 127, 255);
    // use Point to access an element
    for (int y = 200; y < doodle.rows / 2; y++)
        for (int x = 200; x < doodle.cols / 2; x++)
            doodle.at<cv::Vec3b>(cv::Point(x, y)) = orangeVec;

    cv::Mat imageDoodled;
    imageDoodled = image + doodle;
    // This will not work as expected because .row() only forms a header.
    imageDoodled.row(300) = doodle.row(1);
    // This is recommended.
    doodle.row(1).copyTo(imageDoodled.row(300));

    cv::namedWindow("image");
    cv::namedWindow("doodle");
    cv::namedWindow("imageDoodled");
    // Show the image in the window
    cv::imshow("image", image);
    cv::imshow("doodle", doodle);
    cv::imshow("imageDoodled", imageDoodled);
    // Wait until a keypress
    cv::waitKey(0);

    return EXIT_SUCCESS;
}
```