

# Multimedia (Lab 01)

Spring, 2018

Department of Software

Yong Ju Jung (정용주)

# Homework

- Try to find multimedia-related projects in CS(or EE, ECE) departments in Stanford Univ., MIT, UC Berkeley, EPFL.
- You can also find latest awesome papers in CVPR (International Conference on Computer Vision and Pattern Recognition). Read it!
- And make a report.

# [Lab01-1] Introduction to OpenCV

- Install Visual C++ and OpenCV library in your computer
- Load an image and display it on your screen

# What is OpenCV

- OpenCV (Open Source Computer Vision Library: <http://opencv.org>) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms.
- Online documentation :  
<http://docs.opencv.org/3.4.0/>
- Tutorial :  
[https://docs.opencv.org/3.4.0/d9/df8/tutorial\\_root.html](https://docs.opencv.org/3.4.0/d9/df8/tutorial_root.html)

# OpenCV

- Install Visual C++ and OpenCV library in your computer
  - OpenCV3.4 – vc15 (Visual Studio2017), vc14 (VS2015)
  - OpenCV3.2 – vc14 (VS2015)
  - OpenCV3.1 – vc14 (VS2015), vc12 (VS2013)
  - OpenCV3.0 – vc12, vc11 (VS2012)
- If you are using x86 computer, you have to install OpenCV3.0 prebuild or you have to build the source code by yourself.

- OpenCV has a **modular structure**, which means that the package includes several shared or static libraries. The following modules are available:
  - **Core functionality** - a compact module defining basic data structures, including the dense multi-dimensional array 'Mat' and basic functions used by all other modules.
  - **Image processing** - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
  - **video** - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
  - **calib3d** - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
  - **features2d** - salient feature detectors, descriptors, and descriptor matchers.
  - **objdetect** - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
  - **highgui** - an easy-to-use interface to simple UI capabilities.
  - **videoio** - an easy-to-use interface to video capturing and video codecs.
  - **gpu** - GPU-accelerated algorithms from different OpenCV modules.
  - ... some other helper modules

# API Concepts

- **cv Namespace**

- All the OpenCV classes and functions are placed into the cv namespace. Therefore, to access this functionality from your code, use the **cv::** specifier or using namespace cv; directive:

- 1 #include "opencv2/core.hpp"
- 2 ...
- 3 **cv::**Mat H = **cv::**findHomography(points1, points2, CV\_RANSAC, 5);
- 4 ...

- or :

- 1 #include "opencv2/core.hpp"
- 2 **using namespace cv;**
- 3 ...
- 4 Mat H = findHomography(points1, points2, CV\_RANSAC, 5 );
- 5 ...

- **Automatic Memory Management**

- OpenCV handles all the memory automatically.
- First of all, `std::vector`, `Mat`, and other data structures used by the functions and methods have destructors that deallocate the underlying memory buffers when needed. This means that the destructors do not always deallocate the buffers as in case of `Mat`. They take into account possible data sharing. A destructor *decrements the reference counter* associated with the matrix data buffer. The buffer is deallocated *if and only if the reference counter reaches zero*, that is, when no other structures refer to the same buffer.
- Similarly, when a `Mat` instance is copied, no actual data is really copied. Instead, the reference counter is incremented to memorize that there is another owner of the same data.
- There is also the **`Mat::clone`** method that creates a full copy of the matrix data. See the example in the next slide:



## • Automatic Allocation of the Output Data

- OpenCV deallocates the memory automatically, as well as automatically allocates the memory for output function parameters most of the time.
- The size and type of the output arrays are determined from the size and type of input arrays.

```

1  #include "opencv2/imgproc.hpp"
2  #include "opencv2/highgui.hpp"
3
4  using namespace cv;
5
6  int main(int, char**)
7  {
8      VideoCapture cap(0);
9      if(!cap.isOpened()) return -1;
10
11     Mat frame, edges;
12     namedWindow("edges",1);
13     for(;;)
14     {
15         cap >> frame;
16         cvtColor(frame, edges, COLOR_BGR2GRAY);
17         GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
18         Canny(edges, edges, 0, 30, 3);
19         imshow("edges", edges);
20         if(waitKey(30) >= 0) break;
21     }
22     return 0;
23 }

```

- A limited fixed set of primitive data types
- That is, array elements should have one of the following types:
  - 8-bit unsigned integer (uchar)
  - 8-bit signed integer (schar)
  - 16-bit unsigned integer (ushort)
  - 16-bit signed integer (short)
  - 32-bit signed integer (int)
  - 32-bit floating-point number (float)
  - 64-bit floating-point number (double)

```
1 enum { CV_8U=0, CV_8S=1, CV_16U=2, CV_16S=3, CV_32S=4, CV_32F=5, CV_64F=6 };
```

- Examples of array data construction

```

1 Mat mtx(3, 3, CV_32F); // make a 3x3 floating-point matrix
2 Mat cmtx(10, 1, CV_64FC2); // make a 10x1 2-channel floating-point
3                               // matrix (10-element complex vector)
4 Mat img(Size(1920, 1080), CV_8UC3); // make a 3-channel (color) image
5                               // of 1920 columns and 1080 rows.
6 Mat grayscale(image.size(), CV_MAKETYPE(image.depth(), 1)); // make a 1-channel image of
7                               // the same size and same
8                               // channel type as img

```

# OpenCV Tutorials

- [https://docs.opencv.org/3.4.0/d9/df8/tutorial\\_root.html](https://docs.opencv.org/3.4.0/d9/df8/tutorial_root.html)
- **Introduction to OpenCV**
  - You will learn how to setup OpenCV on your computer!
- **The Core Functionality (core module)**
  - Here you will learn the about the basic building blocks of the library. A must read and know for understanding how to manipulate the images on a pixel level.

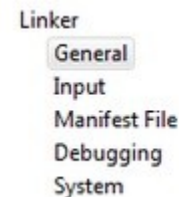
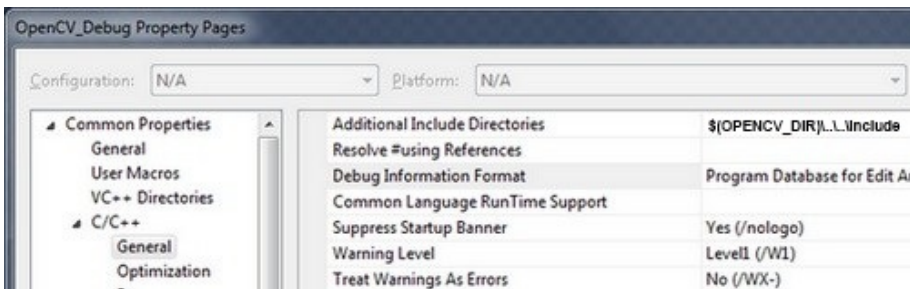
# How to setup OpenCV

## • Installation in Windows

- *Author:* Bernát Gábor
- You will learn how to setup OpenCV in your Windows Operating System!

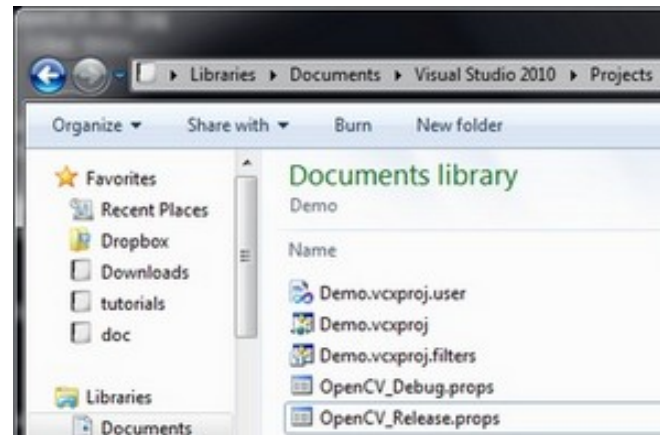
## • How to build applications with OpenCV inside the "Microsoft Visual Studio"

- *Author:* Bernát Gábor
- You will learn what steps you need to perform in order to use the OpenCV library inside a new Microsoft Visual Studio project.
- **opencv\_world340.dll**

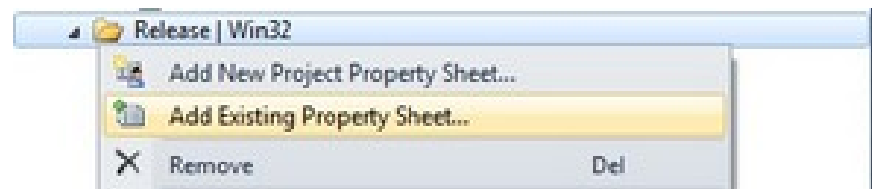


Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	\$(OPENCV_DIR)\lib
Link Library Dependencies	Yes

- You can find your property sheets inside your projects directory. At this point, it is a wise decision to back them up into some special directory, to always have them at hand in the future, whenever you create an OpenCV project. Note that for Visual Studio 2010 the file extension is *props*, while for 2008 this is *vsprops*.



- Next time when you make a new OpenCV project just use the "Add Existing Property Sheet..." menu entry inside the Property Manager to easily add the OpenCV build rules.



# cv:Mat

- In the below image you can see that the mirror of the car is nothing more than a matrix containing all the intensity values of the pixel points.
- All images inside a computer world are numerical matrices and other information describing the matrix itself.
- *OpenCV* is a computer vision library whose main focus is to process and manipulate this information.
- Therefore, the first thing you need to be familiar with is how OpenCV stores and handles images.



- *Mat* is basically a class with two data parts:
  - The **matrix header** (containing information such as the size of the matrix, the method used for storing, at which address is the matrix stored, and so on)
  - and a **pointer** to the matrix containing the pixel values (taking any dimensionality depending on the method chosen for storing).
  - The matrix header size is constant, however the size of the matrix itself may vary from image to image and usually is larger by orders of magnitude.
- You need to know about *Mat* is that you no longer need to manually allocate its memory and release it as soon as you do not need it.
- Most of the OpenCV functions will allocate its output data automatically.



- OpenCV uses a reference counting system.
- The idea is that each *Mat* object has its own header, however the matrix may be shared between two instances of them by having their matrix pointers point to the same address.
- Moreover, the copy operators **will only copy the headers and the pointer to the large matrix, not the data itself.**

```
Mat A, C;
A = imread(argv[1], IMREAD_COLOR); // creates just the header parts
                                     // here we'll know the method used (allocate matrix)

Mat B(A);                          // Use the copy constructor

C = A;                             // Assignment operator
```

- All the above objects, in the end, point to the same single data matrix. Their headers are different, however, and making a modification using any of them will affect all the other ones as well.
- To create a region of interest (*ROI*) in an image you just create a new header with the new boundaries:

```
Mat D (A, Rect(10, 10, 100, 100) ); // using a rectangle
Mat E = A(Range::all(), Range(1,3)); // using row and column boundaries
```

- When the counter reaches zero the matrix too is freed. Sometimes you will want to copy the matrix itself too, so OpenCV provides the [Mat::clone\(\)](#) and [Mat::copyTo\(\)](#) functions.

```
Mat F = A.clone();  
Mat G;  
A.copyTo(G);
```

# Practice

- [Lab01-2]
  - Load an image (using [cv::imread](#) )
  - Create a named OpenCV window (using [cv::namedWindow](#) )
  - Display an image in the OpenCV window (using [cv::imshow](#) )

[http://docs.opencv.org/3.2.0/db/deb/tutorial\\_display\\_image.html](http://docs.opencv.org/3.2.0/db/deb/tutorial_display_image.html)

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui/highgui.hpp>

#include <iostream>
#include <string>

using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
    string imageName("../data/HappyFish.jpg"); // by default
    if( argc > 1)
    {
        imageName = argv[1];
    }

    Mat image;

    image = imread(imageName.c_str(), IMREAD_COLOR); // Read the file
    if( image.empty() ) // Check for invalid input
    {
        cout << "Could not open or find the image" << std::endl ;
        return -1;
    }

    namedWindow( "Display window", WINDOW_AUTOSIZE ); // Create a window for display.
    imshow( "Display window", image ); // Show our image inside it.

    waitKey(0); // Wait for a keystroke in the window
    return 0;
}
```