LAB 7

Fall 2010, BESE- 13 & 14

Image Processing with OpenCV

Objective

The purpose of today's lab is to introduce you to the Open Source Computer Vision Library (OpenCV) developed by Intel®. It gives basic overview about writing a program in OpenCV, using primitive datatypes defined in opencv and manipulating basic image operation. By the end of this lab you should be comfortable enough to read an image, performing basic transformations and writing processed image to disk.

NOTE: Exercises given in this lab are supposed to be practiced within the LAB and are not required to be submitted as a LAB Report.

Tasks for Today

1. Introduction to OpenCV

OpenCV is an Open Source Computer Vision Library from Intel®. Like MATLAB it also offers image processing operation. Comparative to MATLAB it is free, open source, and optimized for real-time applications. It offers a wide range of features:

a. Image manipulation (copying, converting, resizing, destroying image etc).
b. Image processing operations (intensity transforms, filtering, edge/corner detection, morphological operations, image histograms etc).
c. Video processing (tracking, frame handling, motion analysis, optical flow etc).
d. Optimized routines to perform algebraic operations on matrices and vectors.
e. Dynamic data structures (lists, queues, graphs, trees etc).
f. Structural analysis (labeling, components connectivity, contour processing, image moments, template matching, hough transform).
g. Object Recognition and Segmentation.
h. Camera calibration
i. Human Computer Interaction (HCI) and Artificial Intelligence (e.g. neural networks, policy search, curve/line fitting, polygonal approximation etc).

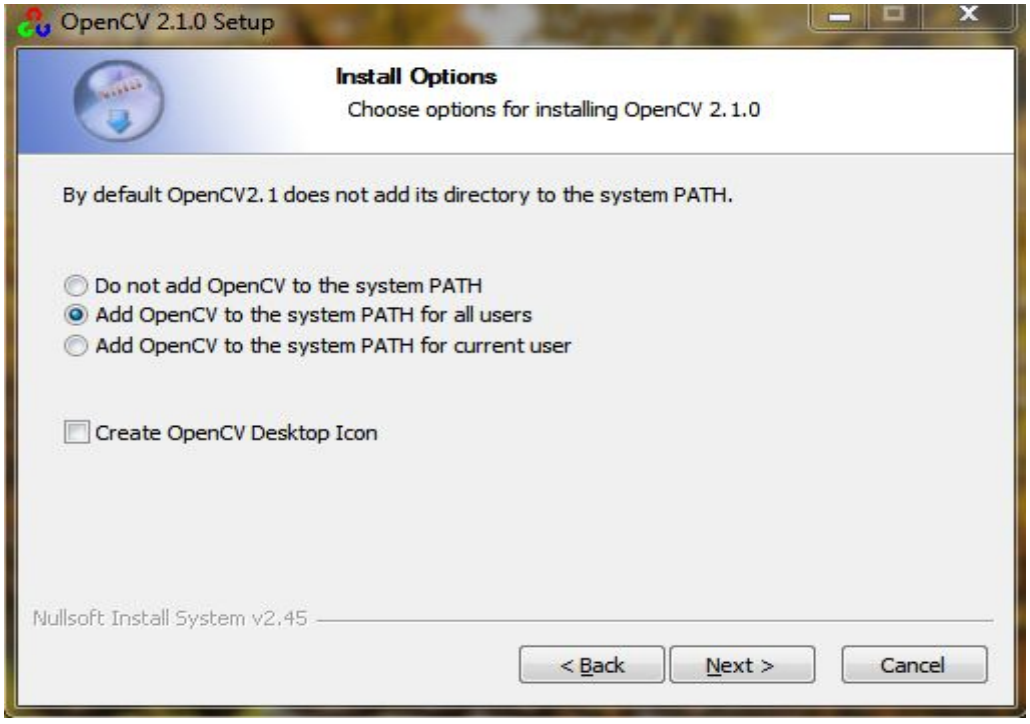
2. Working with OpenCV in Microsoft Visual C++ 2008. NET

For Microsoft dot NET framework OpenCV offers integration only with Visual C++ dot NET (use EmguCV for Microsoft Visual C# dot NET). The functionality of OpenCV is defined and distributed among several modules, mentioned as below:

- **cv** – contains image processing, and camera calibration methods. Also, it includes computational geometry functions.
- **cvaux** – contains obsolete and auxiliary (experimental). The code for face recognition is also defined here.
- **cxcore** – wraps basic datatypes definitions (e.g. data structures for image, point, and rectangles). Also, it contains linear algebra and statistics methods and the error handles.
- **highgui** – defines GUI functions and I/O interfaces.
- **ml** – contains machine learning interfaces.
- **cvcam** – contains interfaces for camera.

2.1. Guidelines for Installing OpenCV for Microsoft Visual C++ 2008 dot NET:

Download and install opencv v2.1 for Microsoft Visual C++ 2008 dot NET from http://opencv.willowgarage.com/wiki/Welcome. During the installation you will be asked *"if you want to add OpenCV to System PATH"*, as shown below:

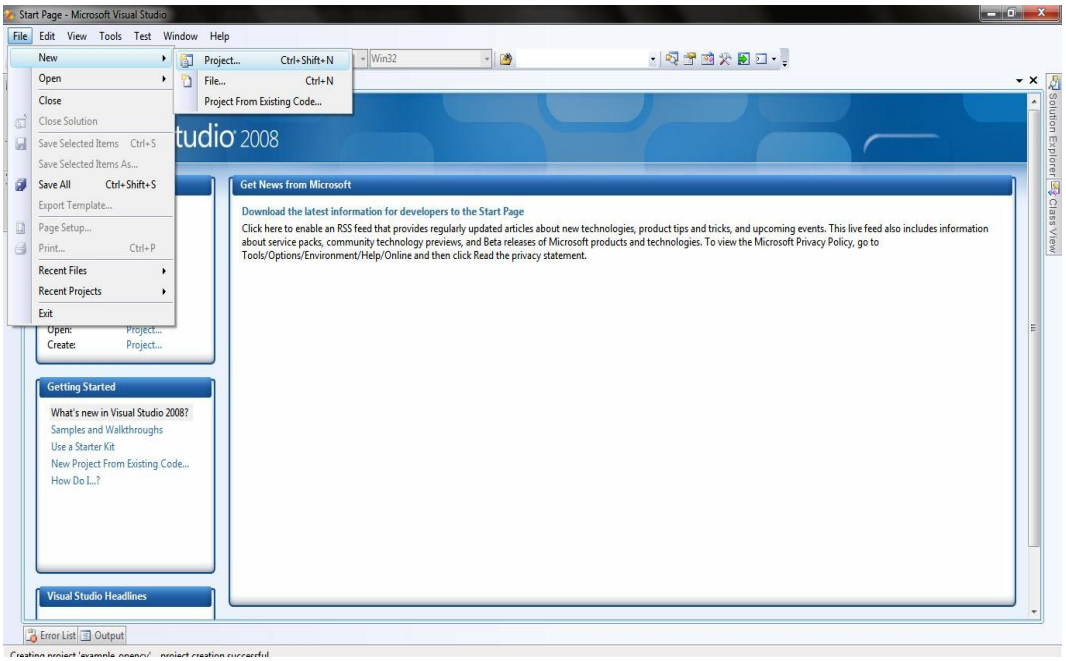


Select Add OpenCV to system PATH for "*all users*" Or *"current user"* as per your requirements. Click Next to proceed and finish the installation successfully.

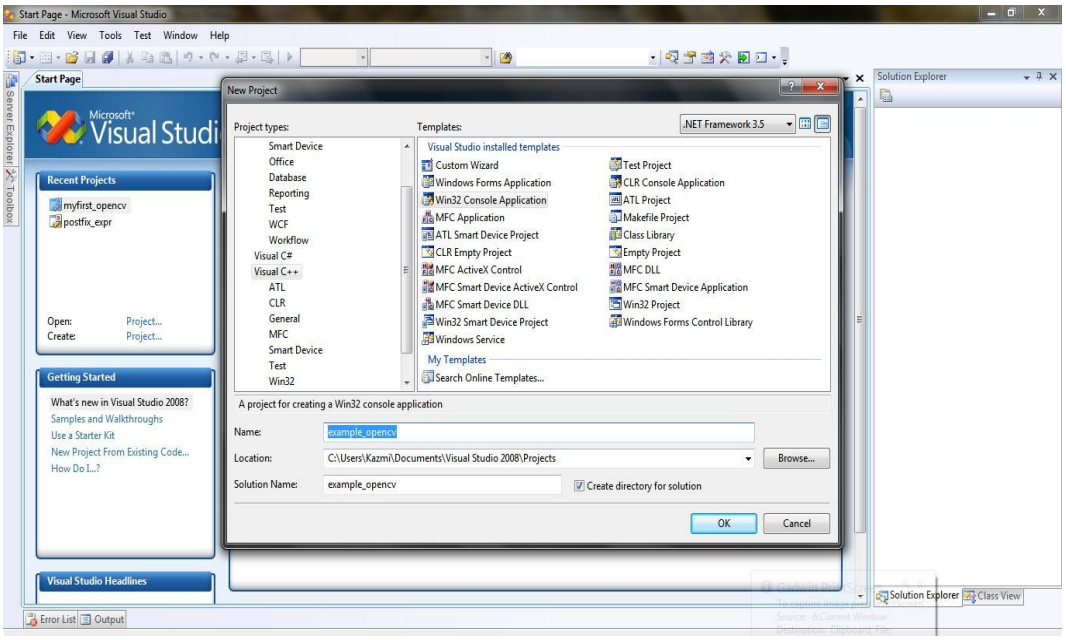
2.2. Configuring OpenCV for Microsoft Visual C++ 2008 dot NET:

After the OpenCV has been successfully installed and all the system PATHS are set, follow the steps as listed below:

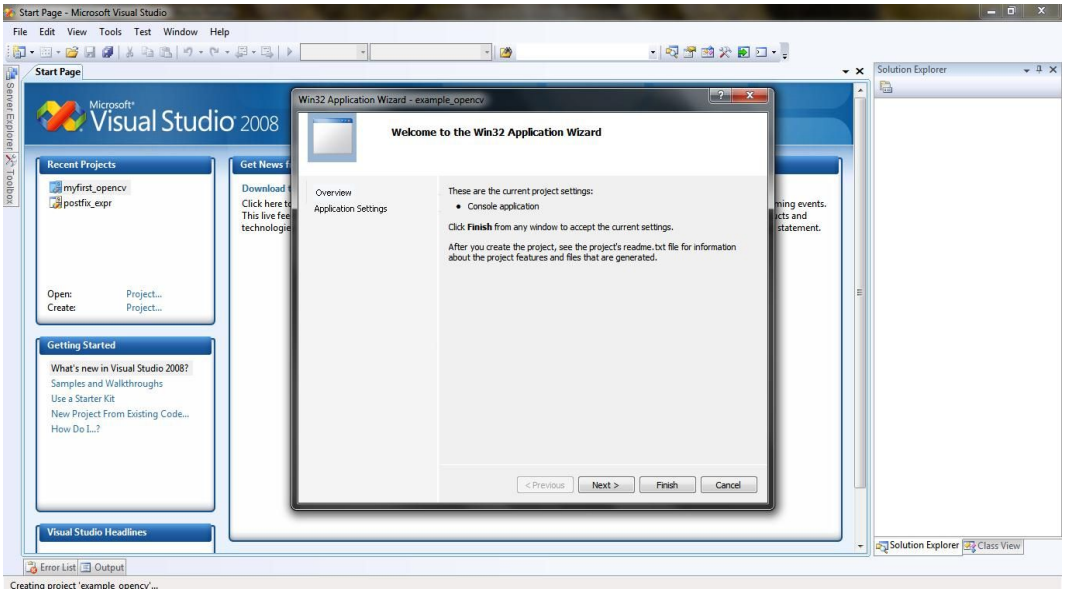
[Step – 1]: Create a New Project: go to File → New → Project.



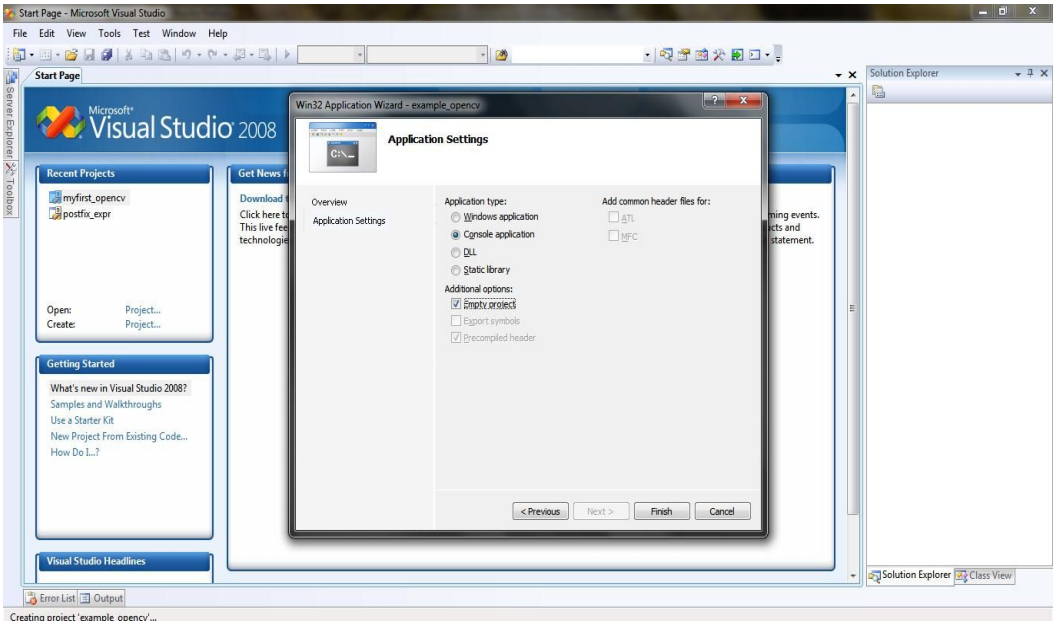
[Step – 2]: Select *Visual C++* for *Project Types* and *Win32 Console Application* under Templates. Give some name to your project (in our case it's *example_opencv*).



[Step – 3]: Finalize the current project settings by clicking *next* button.



[Step – 4]: Select *application type* as *console application* and check *empty project* under *additional options*, as shown below:



[Step – 5]: Click *finish* to finalize all setting and creating an empty project (i.e. with no source and header files).

[Step – 6]: Configure VC++ Directories: goto Tools → Options → Projects and Solutions → VC++ Directories.

- Includes Directories… add: 'C:\OpenCV2.1\include\opencv'

- Library Directories… add: 'C:\OpenCV2.1\lib'
- Source Directories... add: 'C:\OpenCV2.1\src\cv; C:\OpenCV2.1\src\cvaux; C:\OpenCV2.1\src\cxcore; C:\OpenCV2.1\src\highgui; C:\OpenCV2.1\src\ml;'

[Step – 7]: Configure Project Properties: Projects → Properties → Configuration Properties → Linker → Input → Additional Dependencies…

- **for debug builds..** add: 'cv210d.lib; cxcore210d.lib; highgui210d.lib;'
- **for release builds..** add: 'cv210.lib; cxcore210.lib; highgui210.lib;'

2.3. Primitive DataTypes in OpenCV

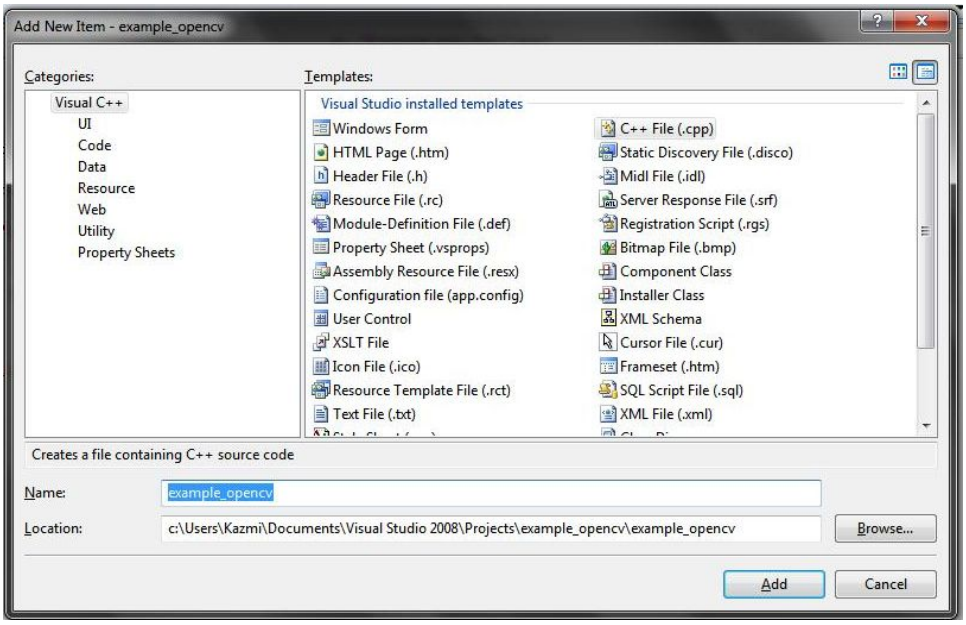
In OpenCV, apart from C dataypes, there are several other types those are all simple structures. They are primitive in the sense that they are particular used for image and image's data manipulation.

a. **CvPoint**… is a simple structure with two integers type members x and y.
b. **CvPoint2D32f**… is same as CvPoint class except here x and y are floating point numbers.
c. **CvPoint3D32f**… is a type with 3 floating type members i.e. x, y, and z.
d. **CvSize**… same as CvPoint but its members are height and width of type integer.
e. **CvSize2D32f**… same as CvSize except here height and width are of type float.
f. **CvRect**… is a child class of CvPoint and CvSize; it has four data members x, y, width and height.
g. **CvScalar**… is a type with four double-precision numbers (can be used if memory is not an issue or used to represent 1D, 2D or 3D numbers).

Structure	Contains	Represents
CvPoint	int x, y	Point in image
CvPoint2D32f	float x, y	Points in $\Re^2$
CvPoint3D32f	float x, y, z	Points in $\Re^3$
CvSize	int width, height	Size of image
CvRect	int x, y, width, height	Portion of image
CvScalar	double val[4]	RGBA value

2.4. Writing a Sample Program in OpenCV

In Visual C++ 2008 dot NET, find the solution explorer of right side of the screen showing *resources*, *header* and *source files* that are part of the current project. Right click the *source files* folder with a mouse and select Add → New Item…. Select the *C++ File (.cpp)* under *Templates* and give any suitable name to this file, in this case *example_opencv*, as shown below:



Write the following sample code that will read an image from the specified path and display it:

```
#include <iostream>
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>

int main(int argn, char *args[]) {

      //load the image from current project's directory
      char img_path[] = "img6.jpg";
      IplImage *img = cvLoadImage(img_path);

      if (img == NULL)
            cout << "Couldn't read an image from " << img_path;

      //load the image from current project's directory
      If plImage *img = cvLoadImage("img6.jpg");

      //create a window in which image will be displayed
      cvNamedWindow("main_win", CV_WINDOW_AUTOSIZE);

      //set position of the created window for image
      cvMoveWindow("main_win", 100, 100);

      //show image from current project's directory
```

```
            cvShowImage("main_win", img);

            //wait until user press a key
            cvWaitKey(0);

            //leave the hold of occupied resourse(s)
            cvDestroyWindow("main_win");
            cvReleaseImage(& img);

            return 0;
}
```

In the above program IplImage is an image data structure that holds the image data and all the information about it.

IplImage Structure Definition…

```
typedef struct _IplImage {

      int nSize; /* size of iplImage struct */
      int ID; /* image header version */
      int nChannels;
      int alphaChannel;
      int depth; /* pixel depth in bits */
      char colorModel[4];
      char channelSeq[4];
      int dataOrder;
      int origin;
      int align; /* 4- or 8-byte align */
      int width;
      int height;
      struct _IplROI *roi; /* pointer to ROI if any */
      struct _IplImage *maskROI; /*pointer to mask ROI if any */
      void *imageId; /* use of the application */
      struct _IplTileInfo *tileInfo; /* information on tiling */
      int imageSize; /* useful size in bytes */
      char *imageData; /* pointer to aligned image */
      int widthStep; /* size of aligned line in bytes */
      int BorderMode[4]; /* top, bottom, left, right border mode */
      int BorderConst[4]; /* constants for border */
      char *imageDataOrigin; /* ptr to full, nonaligned image */

} IplImage;
```

Loading an image… cvLoadImage()

cvLoadImage() is a function that reads an image from a specified path, allocates memory for image data structure and return the pointer to that image data structure i.e. IplImage*, its definition is as follows:

```
IplImage* cvLoadImage( const char* filename,
                       int iscolor = CV_LOAD_IMAGE_COLOR);
```

where *filename* is the constant string which is indeed path of the image file together with its name. cvLoadImage() can read a wide variety of image formats, including BMP, DIB, JPEG, JPE, PNG, PBM, PGM, PPM, SR, RAS, and TIFF. The second

argument iscolor can be set to one of the several values. By default it is CV_LOAD_IMAGE_COLOR which will force the image to be stored in 3 channels with 8-bits per channel. CV_LOAD_IMAGE_ANYCOLOR will read the image from disk as it is stored in the file. CV_LOAD_IMAGE_GRAYSCALE will convert the image to a single channel image i.e. grayscale.

Displaying an image… cvShowImage()

cvShowImage() function takes two arguments to display an image. First argument is the window in which image is to be displayed and second argument is the image which is to be displayed.

```
void cvShowImage(const char* name, const CvArr* image);
```

Creating a Window… cvNamedWindow()

cvNamedWindow() is an OpenCV function defined in highgui library and it takes two arguments. First argument is name of the window which will hold image and second specifies if the image window is re-sizeable by user or not. By default its CV_WINDOW_AUTOSIZE i.e. window will automatically resize itself if the new image is loaded.

```
int cvNamedWindow(const char* name, int flags = CV_WINDOW_AUTOSIZE);
```

Set 2nd argument to 0 to allow user for changing the size of the image window.

Moving a Window… cvMoveWindow()

cvMoveWindow() moves the specified window to the provided pixel co-ordinates on the screen. It is also defined in highgui library. It takes three arguments: first argument is name of the window to be moved whereas second and third arguments specify the position of window on screen coordinates.

```
int cvMoveWindow(const char* name, int x, int y);
```

Resizing a Window… cvResizeWindow()

cvResizeWindow() resizes the specified window to the given width and height.

```
void cvResizeWindow(const char* name, int width, int height);
```

Allocating and Releasing an Image… cvCreateImage, cvReleaseImage()

cvCreateImage() creates an image header and allocates image data.

```
IplImage* cvCreateImage(CvSize size, int depth, int channels);
```

where *size* is an image's width and height, *depth* is the depth of an image e.g. IPL_DEPTH_8U, IPL_DEPTH_16U, IPL_DEPTH_32F etc. Channels specify the number of channels (e.g. RGBA) per pixel.

cvReleaseImage() can be used to free the allocated memory. It takes one argument that is address (&) of the image for which memory is to be de-allocated.

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

Releasing Resources… cvDestroyWindow()

cvDestroyWindow() closes the specified window and de-allocates any associated memory usage.

```
void cvDestroyWindow(const char* name);
```

Also, one can use cvDestroyAllWindows() to close all the opened window for current application and de-allocating any associated memory use.

Saving an Image… cvSaveImage()

cvSaveImage() writes the image provided as IplImage or CvArr data structure to the disk on a specified path.

```
int cvSaveImage(const char* filename, const CvArr* image);
```

1st argument is the filename together with its path and 2nd argument is an image that is to be written onto disk.

Exercise 1:

Read any image from the disk, convert it to grayscale image and save the converted image to disk.

Reading Image's Pixel Value… cvGet2D()
cvGet2D()… retrieve image's pixel value at location (x, y) from a specified image.

```
CvScalar cvGet2D(const CvArr *arr, int idx0, int idx1);
```

It takes an instance of an image structure (IplImage) or a matrix (CvArr) as first argument. 2nd and 3rd arguments are the pixel co-ordinates whose value is to be retrieved (read).
But this way of accessing the image pixels is not preferable because it is computationally expensive. So alternatively image data for a GRAYSCALE image can be accessed as:

```
uchar pix_val = img->imageData[r * img->widthStep + c];
```

where r and c specify the corresponding row and column index number.

Setting Image's Pixel Value… cvSet2D()
cvSet2D()… sets the image's pixel value for the specified pixel coordinate.

```
void cvSet2D(const CvArr *arr, int idx0, int idx1, CvScalar value);
```

It takes an instance of an image structure (IplImage) or a matrix (CvArr) as first argument. 2nd and 3rd arguments are the pixel co-ordinates for which the new value to be set is defined by CvScalar i.e. fourth argument.
Alternatively, image's pixel value for a GRAYSCALE image can be set as:

```
img->imageData[r * img->widthStep + c] = pix_val;
```

where pix_val is a value [0…255] that is to be set and r and c specify the corresponding row and column index number respectively.

Exercise 2:

Read any image from the disk, convert it to grayscale image, apply inverse transform on it and display both the original as well as transformed image.