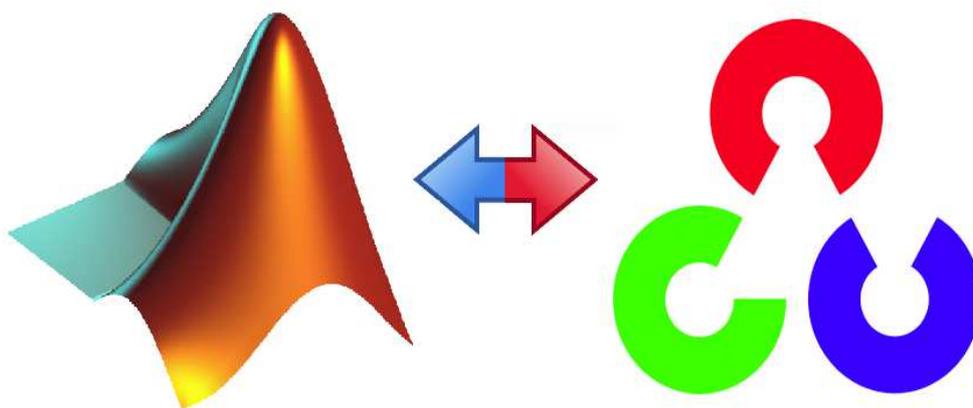


Connecting MATLAB with OpenCV



The following guide is also provided by the following document

[A guide to connecting Matlab with OpenCV](#)

This webpage provides a short guide to connecting Matlab with OpenCV. The main reason to adopt such a connection is the efficiency

Matlab provides a MEX environment in order to write C functions instead of M-files. Recall that MEX (Matlab-EXecutable) files are dynamically linked subroutines from C/C++ code (or Fortran code) that, when compiled, can be run from within Matlab like M-files. Hence, MEX environment offers a way to call your custom C/C++ routines as if they were Matlab built-in function. A detailed MEX Guide is offered by MathWorks [HERE](#)

The following description presupposes that, except for Matlab, OpenCV is installed in your system. Note that connection has been verified in a 64-bit machine with Win7, OpenCV2.1 and 32-bit Matlab R2009b. If you have installed a 64-bit Matlab version, try to use OpenCV 2.3 (or 2.3.1) that offers a prebuilt library for both 32-bit and 64-bit systems. Otherwise you could re-build the libraries from source-code using the proper generator (32-bit or 64 bit). Note that 32-bit Matlab works well in 64-bit machines, so I would suggest this version. (I think that you cannot build .mexw64 files using 32-bit static libraries of OpenCV.)

In order to be able to compile any C/C++, you have to setup MEX environment and choose the appropriate compiler with the command

```
>> mex -setup
```

Note that I used the compiler (cl) of Visual C++ 2008 express edition (it works fine). I provide below two examples of MEX-files that use OpenCV functions. The first example is the simplest case where no input/output arguments are required. For example, image loading is called from within the function and the MEX file returns nothing (it just displays the result). The second case is more interesting since input and output arguments are Matlab variables. The [Sun Peng's types convertor](#) that provides bidirectional conversion between C/C++ (OpenCV)types and matlab matrices is used by this case.

Example 1: A simple MEX file without input/output arguments that uses simple OpenCV modules

```

/*****
 This is a simple MEX file. It just calls openCV functions for loading,
 smothing and diaplaying the results
 *****/

#include <stdio.h>
#include <stdlib.h>
#include <opencv/cv.h>
#include <opencv/highgui.h>

#include "mex.h"

int smoothImage(char* filename){

//load the image
IplImage* img = cvLoadImage(filename);
if(!img){
printf("Cannot load image file: %s\n",filename);
return -1;
}

//create another image
IplImage* img_smooth = cvCloneImage(img);

//smooth the image img and save the result to img_smooth
cvSmooth(img, img_smooth, CV_GAUSSIAN, 5, 5);

// create windows to show images
cvNamedWindow("Original Image", CV_WINDOW_AUTOSIZE);
cvMoveWindow("Original Image", 100, 100);
cvNamedWindow("Smoothed Image", CV_WINDOW_AUTOSIZE);
cvMoveWindow("Smoothed Image", 400, 400);

// show the images
cvShowImage("Original Image", img );
cvShowImage("Smoothed Image", img_smooth );

// wait for a key
cvWaitKey(0);

//destroy windows
cvDestroyWindow("Original Image");
cvDestroyWindow("Smoothed Image");

//release images
cvReleaseImage(& img);
cvReleaseImage(& img_smooth);

return 0;
};

void mexFunction(int nlhs, mxArray* plhs[], int nrhs, const mxArray* prhs[]){

if (nrhs != 0)
{
mexErrMsgTxt("Do not give input arguments.");
}
if (nlhs != 0)
{
mexErrMsgTxt("Do not give output arguments.");
}

char *name = "cameraman.png";

```

```
smoothImage ( name ) ;
}

```

Let us suppose that the above code is saved to [displayImage.cpp](#) file. Since this file makes use of OpenCV header files, the compiler must be informed with the appropriate include directories. Moreover, in order to produce the executable file, the necessary libraries should be given.

There are two ways to define the required compilation flags:

1. by pre-editing the MEXOPTS.BAT file
2. by defining paths and libraries with the `mex` command

By editing the MEXOPTS.BAT file

When you choose a compiler with `mex -setup` an appropriate batch file is created containing all required settings the compilation needs. This is the file `mexopts.bat` and in order to find it, just type in Matlab

```
>> fullfile(prefdir, 'mexopts.bat')
```

In my machine, this file is in

```
C:\users\\AppData\Roaming\MathWorks\MATLAB\R2009b\mexopts.bat
```

Edit the above file and add follow the next steps:

1. set the OpenCV path based on your installation, i.e.

```
SET OCVDIR=C:\OpenCV2.1
```

2. Find the `INCLUDE` variable definition (`SET INCLUDE`) and add the path that contains the OpenCV include folder (the folder that contains the header files `cv.h`, `cxcore.h` etc)

```
%OCVDIR%\include\opencv
```

```
%OCVDIR%\include
```

3. Find the `LIB` variable definition (`SET LIB`) and add the path that contains the OpenCV libraries (*.lib files)

```
%OCVDIR%\lib
```

4. Find the `LINKFLAGS` variable definition (`SET LINKFLAGS`) and add the standard .lib files of OpenCV (here of OpenCV 2.1)

```
cv210.lib
```

```
cxcore210.lib
```

```
highgui210.lib
```

```
cxcore210.lib
```

After that, just run

```
>> mex displayImage.cpp
```

and the file `displayImage.mexw32` will be created. Then, you can call the Matlab function `diplayImage` without arguments and it will show the image that is loaded within the source (here the image [cameraman.png](#)).

By defining paths and libs with the `mex` command

Alternatively, you can define the paths and libraries anytime you call the `mex` command. For example, say that your source file needs the header files `cv.h`, `highgui.h` and `cxcore.h` as above. Then you need to define the include directory using the flag `-I` and the three `.lib` files.

```
>> OCVRoot = C:\OpenCV2.1;
>> IPath = ['-I',fullfile(OCVRoot,'include')];
>> LPath = fullfile(OCVRoot, 'lib');
>> lib1 = fullfile(LPath,'cv210d.lib');
>> lib2 = fullfile(LPath,'cxcore210d.lib');
>> lib3 = fullfile(LPath,'highgui210d.lib');

>> mex('displayImage.cpp', Ipath, lib1, lib2, lib3);
```

Then, you can show the image in Matlab as described above.

Example 2: A MEX file with input/output arguments that uses OpenCV

I consider the above example of image smoothing but now the input and output images are matlab variables. A Gaussian filter is adopted, the size of which is given with two more input parameters (height, width). As I mentioned above, I make use of Sun Peng's type convertor to switch between `mxArray` and `IplImage` structures or other C/C++ data types (find [HERE](#) only the appropriate files).

```
/*
*****
This is a simple MEX file that accepts as inputs an image and the
size of a Gaussian filter(two parameters). Then it applies the filter
to the image by calling the OpenCV function cvSmooth and returns the
filtered image to a matlab variable.
*****
*/

#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <opencv/cxcore.h>

#ifndef HAS_OPENCV
#define HAS_OPENCV
#endif

#include "mex.h"
#include "mc_convert.h"
#include "mc_convert.cpp"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
                 const mxArray *prhs[]) {

    //Read Matlab image and load it to an IplImage struct
    IplImage* inputImg = mxArr_to_new_IplImage(prhs[0]);

    //Read the filter parameters
    double filterHeight, filterWidth;
    mat_to_scalar (prhs[1], &filterHeight);
    mat_to_scalar (prhs[2], &filterWidth);

    //smooth the input image and save the result to outputImg
    IplImage* outputImg = cvCloneImage(inputImg);
```

```
cvSmooth(inputImg, outputImg, CV_GAUSSIAN, filterWidth, filterHeight);

//Return output image to mxArray (Matlab matrix)
plhs[0] = IplImage_to_new_mxArr(outputImg);
cvReleaseImage(&inputImg);
cvReleaseImage(&outputImg);
}
```

Note that Matlab-to-OpenCV convertor needs to define the symbol name `HAS_OPENCV` to the C preprocessor. I define it in the source code, but one can either add the switch `-DHAS_OPENCV` to `mex` command or edit the `COMPFLAGS` definition in `MEXOPTS.BAT` (find the `SET COMPFLAGS` line) and add the `/DHAS_OPENCV` flag.

Given that the source is saved to the file [smoothImage.cpp](#), you just need to compile it and run it. The command

```
>> mex smoothImage.cpp
```

will create the file `smoothImage.mexw32`. Then, you can run the Matlab function as follows

```
>> im = imread('cameraman.png');
>> filterHeight = 7;
>> filterWidth = 7;
>> outImage = smoothImage(im, filterHeight, filterWidht);
```

Good Luck!

Contact

For any bugs, questions or help, please contact the author.

[Georgios Evangelidis](#),

e-mail: `evagelid at ceid dot upatras dot gr`