# Lecture 1: [Introduction to OpenCV and SetUp](http://www.cvl.isy.liu.se/education/graduate/opencv/Lecture1_History.pdf)

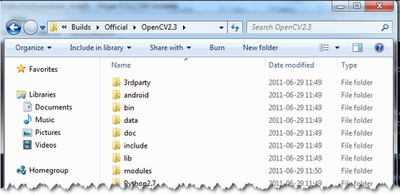
Installation in Windows[¶](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html#installation-in-windows)

The description here was tested on Windows 7 SP1. Nevertheless, it should also work on any other relatively modern version of Windows OS. If you encounter errors after following the steps described below, feel free send me email to [jxr041100@gmail.com](mailto:jxr041100@gmail.com) . We’ll do our best to help you out.

To use the OpenCV library you have two options: [Installation by Using the Pre-built Libraries](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html#windows-install-prebuild) or [Installation by Making Your Own Libraries from the Source Files](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html#cpptutwindowsmakeown). While the first one is easier to complete, it only works if you are coding with the latest Microsoft Visual Studio IDE and doesn’t take advantage of the most advanced technologies we integrate into our library.

Installation by Using the Pre-built Libraries[¶](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html#installation-by-using-the-pre-built-libraries)

1. Launch a web browser of choice and go to our [page on Sourceforge](http://sourceforge.net/projects/opencvlibrary/files/opencv-win/).
2. Choose a build you want to use and download it.
3. Make sure you have admin rights. Unpack the self-extracting archive.
4. You can check the installation at the chosen path as you can see below.



1. To finalize the installation go to the [Set the OpenCV enviroment variable and add it to the systems path](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html#windowssetpathandenviromentvariable) section.

Installation by Making Your Own Libraries from the Source Files[¶](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html#installation-by-making-your-own-libraries-from-the-source-files)

You may find the content of this tutorial also inside the following videos: [Part 1](https://www.youtube.com/watch?v=NnovZ1cTlMs) and [Part 2](https://www.youtube.com/watch?v=qGNWMcfWwPU), hosted on YouTube.

Warning

These videos above are long-obsolete and contain inaccurate information. Be careful, since solutions described in those videos are no longer supported and may even break your install.

If you are building your own libraries you can take the source files from our [Git repository](https://github.com/Itseez/opencv.git).

Building the OpenCV library from scratch requires a couple of tools installed beforehand:

* An IDE of choice (preferably), or just a CC++ compiler that will actually make the binary files. Here we will use the [Microsoft Visual Studio](https://www.microsoft.com/visualstudio/en-us). However, you can use any other IDE that has a valid CC++ compiler.
* [CMake](http://www.cmake.org/cmake/resources/software.html), which is a neat tool to make the project files (for your chosen IDE) from the OpenCV source files. It will also allow an easy configuration of the OpenCV build files, in order to make binary files that fits exactly to your needs.
* Git to acquire the OpenCV source files. A good tool for this is [TortoiseGit](http://code.google.com/p/tortoisegit/wiki/Download). Alternatively, you can just download an archived version of the source files from our [page on Sourceforge](http://sourceforge.net/projects/opencvlibrary/files/opencv-win/)

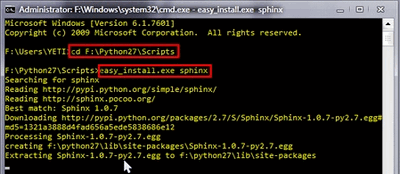
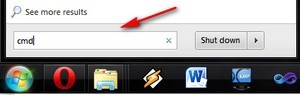
OpenCV may come in multiple flavors. There is a “core” section that will work on its own. Nevertheless, there is a couple of tools, libraries made by 3rd parties that offer services of which the OpenCV may take advantage. These will improve its capabilities in many ways. In order to use any of them, you need to download and install them on your system.

* The [Python libraries](http://www.python.org/downloads/) are required to build the Python interface of OpenCV. For now use the version 2.7.*x*. This is also a must if you want to build the OpenCV documentation.
* [Numpy](http://numpy.scipy.org/) is a scientific computing package for Python. Required for the Python interface.
* [Intel © Threading Building Blocks (TBB)](http://threadingbuildingblocks.org/file.php?fid=77) is used inside OpenCV for parallel code snippets. Using this will make sure that the OpenCV library will take advantage of all the cores you have in your systems CPU.
* [Intel © Integrated Performance Primitives (IPP)](http://software.intel.com/en-us/articles/intel-ipp/) may be used to improve the performance of color conversion, Haar training and DFT functions of the OpenCV library. Watch out, since this isn’t a free service.
* OpenCV offers a somewhat fancier and more useful graphical user interface, than the default one by using the [Qt framework](http://qt.nokia.com/downloads). For a quick overview of what this has to offer look into the documentations highgui module, under the Qt New Functions section. Version 4.6 or later of the framework is required.
* [Eigen](http://eigen.tuxfamily.org/index.php?title=Main_Page#Download) is a C++ template library for linear algebra.
* The latest [CUDA Toolkit](http://developer.nvidia.com/cuda-downloads) will allow you to use the power lying inside your GPU. This will drastically improve performance for some algorithms (e.g the HOG descriptor). Getting more and more of our algorithms to work on the GPUs is a constant effort of the OpenCV team.
* [OpenEXR](http://www.openexr.com/downloads.html) source files are required for the library to work with this high dynamic range (HDR) image file format.
* The [OpenNI Framework](http://www.openni.org/) contains a set of open source APIs that provide support for natural interaction with devices via methods such as voice command recognition, hand gestures and body motion tracking.
* [Miktex](http://miktex.org/2.9/setup) is the best [TEX](https://secure.wikimedia.org/wikipedia/en/wiki/TeX) implementation on the Windows OS. It is required to build the OpenCV documentation.
* [Sphinx](http://sphinx.pocoo.org/) is a python documentation generator and is the tool that will actually create the OpenCV documentation. This on its own requires a couple of tools installed, We will cover this in depth at the [How to Install Sphinx](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html#hereinstallsphinx) section.

Now we will describe the steps to follow for a full build (using all the above frameworks, tools and libraries). If you do not need the support for some of these you can just freely skip this section.

Building the library[¶](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html#building-the-library)

1. Make sure you have a working IDE with a valid compiler. In case of the Microsoft Visual Studio just install it and make sure it starts up.
2. Install [CMake](http://www.cmake.org/cmake/resources/software.html). Simply follow the wizard, no need to add it to the path. The default install options are OK.
3. Download and install an up-to-date version of msysgit from its [official site](http://code.google.com/p/msysgit/downloads/list). There is also the portable version, which you need only to unpack to get access to the console version of Git. Supposing that for some of us it could be quite enough.
4. Install [TortoiseGit](http://code.google.com/p/tortoisegit/wiki/Download). Choose the 32 or 64 bit version according to the type of OS you work in. While installing, locate your msysgit (if it doesn’t do that automatically). Follow the wizard – the default options are OK for the most part.
5. Choose a directory in your file system, where you will download the OpenCV libraries to. I recommend creating a new one that has short path and no special charachters in it, for example D:/OpenCV. For this tutorial I’ll suggest you do so. If you use your own path and know, what you’re doing – it’s OK.
   1. Clone the repository to the selected directory. After clicking Clone button, a window will appear where you can select from what repository you want to download source files (<https://github.com/Itseez/opencv.git>) and to what directory (D:/OpenCV).
   2. Push the OK button and be patient as the repository is quite a heavy download. It will take some time depending on your Internet connection.
6. In this section I will cover installing the 3rd party libraries.
   1. Download the [Python libraries](http://www.python.org/downloads/) and install it with the default options. You will need a couple other python extensions. Luckily installing all these may be automated by a nice tool called [Setuptools](http://pypi.python.org/pypi/setuptools#downloads). Download and install again.
   2. Installing Sphinx is easy once you have installed Setuptools. This contains a little application that will automatically connect to the python databases and download the latest version of many python scripts. Start up a command window (enter cmd into the windows start menu and press enter) and use the CD command to navigate to your Python folders Script sub-folder. Here just pass to the easy\_install.exe as argument the name of the program you want to install. Add the sphinx argument.

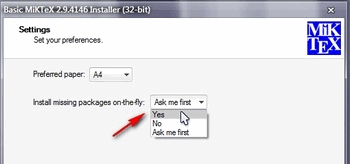


Note

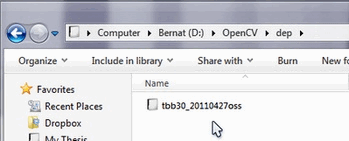
The CD navigation command works only inside a drive. For example if you are somewhere in the C: drive you cannot use it this to go to another drive (like for example D:). To do so you first need to change drives letters. For this simply enter the command D:. Then you can use the CD to navigate to specific folder inside the drive. Bonus tip: you can clear the screen by using the CLS command.

This will also install its prerequisites [Jinja2](http://jinja.pocoo.org/docs/) and [Pygments](http://pygments.org/).

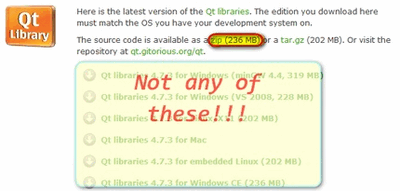
* 1. The easiest way to install [Numpy](http://numpy.scipy.org/) is to just download its binaries from the [sourceforga page](http://sourceforge.net/projects/numpy/files/NumPy/). Make sure your download and install exactly the binary for your python version (so for version 2.7).
  2. Download the [Miktex](http://miktex.org/2.9/setup) and install it. Again just follow the wizard. At the fourth step make sure you select for the “Install missing packages on-the-fly” the Yes option, as you can see on the image below. Again this will take quite some time so be patient.



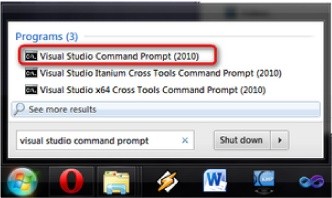
* 1. For the [Intel © Threading Building Blocks (TBB)](http://threadingbuildingblocks.org/file.php?fid=77) download the source files and extract it inside a directory on your system. For example let there be D:/OpenCV/dep. For installing the [Intel © Integrated Performance Primitives (IPP)](http://software.intel.com/en-us/articles/intel-ipp/) the story is the same. For exctracting the archives I recommend using the [7-Zip](http://www.7-zip.org/) application.



* 1. In case of the [Eigen](http://eigen.tuxfamily.org/index.php?title=Main_Page#Download) library it is again a case of download and extract to the D:/OpenCV/dep directory.
  2. Same as above with [OpenEXR](http://www.openexr.com/downloads.html).
  3. For the [OpenNI Framework](http://www.openni.org/) you need to install both the [development build](http://www.openni.org/downloadfiles/opennimodules/openni-binaries/21-stable) and the [PrimeSensor Module](http://www.openni.org/downloadfiles/opennimodules/openni-compliant-hardware-binaries/32-stable).
  4. For the CUDA you need again two modules: the latest [CUDA Toolkit](http://developer.nvidia.com/cuda-downloads) and the CUDA Tools SDK. Download and install both of them with a complete option by using the 32 or 64 bit setups according to your OS.
  5. In case of the [Qt framework](http://qt.nokia.com/downloads) you need to build yourself the binary files (unless you use the Microsoft Visual Studio 2008 with 32 bit compiler). To do this go to the [Qt Downloads](http://qt.nokia.com/downloads) page. Download the source files (not the installers!!!):



Extract it into a nice and short named directory like D:/OpenCV/dep/qt/ . Then you need to build it. Start up a Visual Studio Command Prompt (2010) by using the start menu search (or navigate through the start menu All Programs ‣ Microsoft Visual Studio 2010 ‣ Visual Studio Tools ‣ Visual Studio Command Prompt (2010)).



Now navigate to the extracted folder and enter inside it by using this console window. You should have a folder containing files like Install, Make and so on. Use the dir command to list files inside your current directory. Once arrived at this directory enter the following command:

configure.exe -release -no-webkit -no-phonon -no-phonon-backend -no-script -no-scripttools

-no-qt3support -no-multimedia -no-ltcg

Completing this will take around 10-20 minutes. Then enter the next command that will take a lot longer (can easily take even more than a full hour):

nmake

After this set the Qt enviroment variables using the following command on Windows 7:

setx -m QTDIR D:/OpenCV/dep/qt/qt-everywhere-opensource-src-4.7.3

Also, add the built binary files path to the system path by using the [Path Editor](http://www.redfernplace.com/software-projects/patheditor/). In our case this is D:/OpenCV/dep/qt/qt-everywhere-opensource-src-4.7.3/bin.

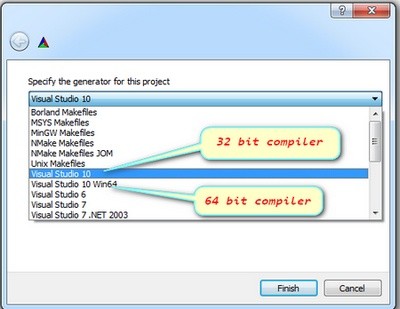
Note

If you plan on doing Qt application development you can also install at this point the Qt Visual Studio Add-in. After this you can make and build Qt applications without using the Qt Creator. Everything is nicely integrated into Visual Studio.

1. Now start the CMake (cmake-gui). You may again enter it in the start menu search or get it from the All Programs ‣ CMake 2.8 ‣ CMake (cmake-gui). First, select the directory for the source files of the OpenCV library (1). Then, specify a directory where you will build the binary files for OpenCV (2).



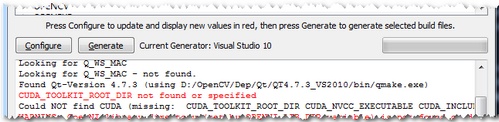
Press the Configure button to specify the compiler (and IDE) you want to use. Note that in case you can choose between different compilers for making either 64 bit or 32 bit libraries. Select the one you use in your application development.



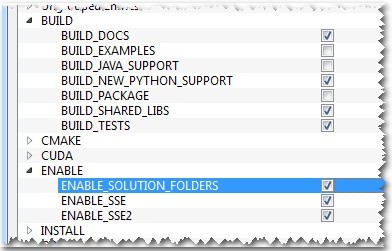
CMake will start out and based on your system variables will try to automatically locate as many packages as possible. You can modify the packages to use for the build in the WITH ‣ WITH\_X menu points (where X is the package abbreviation). Here are a list of current packages you can turn on or off:



Select all the packages you want to use and press again the Configure button. For an easier overview of the build options make sure the Grouped option under the binary directory selection is turned on. For some of the packages CMake may not find all of the required files or directories. In case of these CMake will throw an error in its output window (located at the bottom of the GUI) and set its field values, to not found constants. For example:

Constant for not found packages

For these you need to manually set the queried directories or files path. After this press again the Configure button to see if the value entered by you was accepted or not. Do this until all entries are good and you cannot see errors in the field/value or the output part of the GUI. Now I want to emphasize an option that you will definitely love: ENABLE ‣ ENABLE\_SOLUTION\_FOLDERS. OpenCV will create many-many projects and turning this option will make sure that they are categorized inside directories in the Solution Explorer. It is a must have feature, if you ask me.



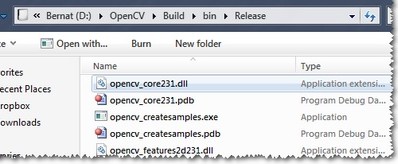
Furthermore, you need to select what part of OpenCV you want to build.

* 1. BUILD\_DOCS -> It creates two projects for building the documentation of OpenCV (there will be a separate project for building the HTML and the PDF files). Note that these aren’t built together with the solution. You need to make an explicit build project command on these to do so.
  2. BUILD\_EXAMPLES -> OpenCV comes with many example applications from which you may learn most of the libraries capabilities. This will also come handy to easily try out if OpenCV is fully functional on your computer.
  3. BUILD\_PACKAGE -> Prior to version 2.3 with this you could build a project that will build an OpenCV installer. With this you can easily install your OpenCV flavor on other systems. For the latest source files of OpenCV it generates a new project that simply creates zip archive with OpenCV sources.
  4. BUILD\_SHARED\_LIBS -> With this you can control to build DLL files (when turned on) or static library files (\*.lib) otherwise.
  5. BUILD\_TESTS -> Each module of OpenCV has a test project assigned to it. Building these test projects is also a good way to try out, that the modules work just as expected on your system too.
  6. BUILD\_PERF\_TESTS -> There are also performance tests for many OpenCV functions. If you’re concerned about performance, build them and run.
  7. BUILD\_opencv\_python -> Self-explanatory. Create the binaries to use OpenCV from the Python language.

Press again the Configure button and ensure no errors are reported. If this is the case you can tell CMake to create the project files by pushing the Generate button. Go to the build directory and open the created **OpenCV** solution. Depending on just how much of the above options you have selected the solution may contain quite a lot of projects so be tolerant on the IDE at the startup. Now you need to build both the Release and the Debug binaries. Use the drop-down menu on your IDE to change to another of these after building for one of them.

Look here for changing the Build Type

In the end you can observe the built binary files inside the bin directory:



For the documentation you need to explicitly issue the build commands on the doc project for the PDF files and on the doc\_html for the HTML ones. Each of these will call Sphinx to do all the hard work. You can find the generated documentation inside the Build/Doc/\_html for the HTML pages and within the Build/Doc the PDF manuals.

The Documentation Projects

To collect the header and the binary files, that you will use during your own projects, into a separate directory (simillary to how the pre-built binaries ship) you need to explicitely build the Install project.

The Install Project

This will create an Install directory inside the Build one collecting all the built binaries into a single place. Use this only after you built both the Release and Debug versions.

To test your build just go into the Build/bin/Debug or Build/bin/Release directory and start a couple of applications like the contours.exe. If they run, you are done. Otherwise, something definitely went awfully wrong. In this case you should contact us at our [Q&A forum](http://answers.opencv.org/). If everything is okay the contours.exe output should resemble the following image (if built with Qt support):



Note

If you use the GPU module (CUDA libraries) make sure you also upgrade to the latest drivers of your GPU. Error messages containing invalid entries in (or cannot find) the nvcuda.dll are caused mostly by old video card drivers. For testing the GPU (if built) run the performance\_gpu.exe sample application.

Set the OpenCV enviroment variable and add it to the systems path[¶](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html#set-the-opencv-enviroment-variable-and-add-it-to-the-systems-path)

First we set an enviroment variable to make easier our work. This will hold the build directory of our OpenCV library that we use in our projects. Start up a command window and enter:

setx -m OPENCV\_DIR D:\OpenCV\Build\x86\vc10 (suggested for Visual Studio 2010 - 32 bit Windows)

setx -m OPENCV\_DIR D:\OpenCV\Build\x64\vc10 (suggested for Visual Studio 2010 - 64 bit Windows)

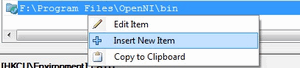
setx -m OPENCV\_DIR D:\OpenCV\Build\x86\vc11 (suggested for Visual Studio 2012 - 32 bit Windows)

setx -m OPENCV\_DIR D:\OpenCV\Build\x64\vc11 (suggested for Visual Studio 2012 - 64 bit Windows)

Here the directory is where you have your OpenCV binaries (extracted or built). You can have different platform (e.g. x64 instead of x86) or compiler type, so substitute appropriate value. Inside this you should have two folders called lib and bin. The -m should be added if you wish to make the settings computer wise, instead of user wise.

If you built static libraries then you are done. Otherwise, you need to add the bin folders path to the systems path. This is cause you will use the OpenCV library in form of “Dynamic-link libraries” (also known as **DLL**). Inside these are stored all the algorithms and information the OpenCV library contains. The operating system will load them only on demand, during runtime. However, to do this he needs to know where they are. The systems **PATH** contains a list of folders where DLLs can be found. Add the OpenCV library path to this and the OS will know where to look if he ever needs the OpenCV binaries. Otherwise, you will need to copy the used DLLs right beside the applications executable file (exe) for the OS to find it, which is highly unpleasent if you work on many projects. To do this start up again the [Path Editor](http://www.redfernplace.com/software-projects/patheditor/) and add the following new entry (right click in the application to bring up the menu):

%OPENCV\_DIR%\bin

Add the entry.

Save it to the registry and you are done. If you ever change the location of your build directories or want to try out your applicaton with a different build all you will need to do is to update the OPENCV\_DIR variable via the setx command inside a command window.

Now you can continue reading the tutorials with the [How to build applications with OpenCV inside the Microsoft Visual Studio](http://docs.opencv.org/doc/tutorials/introduction/windows_visual_studio_Opencv/windows_visual_studio_Opencv.html#windows-visual-studio-how-to) section. There you will find out how to use the OpenCV library in your own projects with the help of the Microsoft Visual Studio IDE.

[1]. OpenCV tutorial

# Lecture 2: Background subtraction, blob analysis and tracking with OpenCV

OpenCV with Images/Video

## Introduction to background subtraction

* Background subtraction (BS) is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras.
* As the name suggests, BS calculates the foreground mask performing a subtraction between the current frame and a background model, containing the static part of the scene or, more in general, everything that can be considered as background given the characteristics of the observed scene. 
* Background modeling consists of two main steps:
  1. Background Initialization;
  2. Background Update.

In the first step, an initial model of the background is computed, while in the second step that model is updated in order to adapt to possible changes in the scene.

* In this tutorial we will learn how to perform BS by using OpenCV. As input, we will use data coming from the publicly available data set [Background Models Challenge (BMC)](http://bmc.univ-bpclermont.fr/) .

## Goals

In this tutorial you will learn how to:

1. Read data from videos by using [**cv::VideoCapture**](http://docs.opencv.org/master/d8/dfe/classcv_1_1VideoCapture.html) or image sequences by using [**cv::imread**](http://docs.opencv.org/master/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56) ;
2. Create and update the background model by using [**cv::BackgroundSubtractor**](http://docs.opencv.org/master/d7/df6/classcv_1_1BackgroundSubtractor.html) class;
3. Get and show the foreground mask by using [**cv::imshow**](http://docs.opencv.org/master/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563) ;
4. Save the output by using [**cv::imwrite**](http://docs.opencv.org/master/d4/da8/group__imgcodecs.html#gabbc7ef1aa2edfaa87772f1202d67e0ce) to quantitatively evaluate the results.

## Code

In the following you can find the source code. We will let the user chose to process either a video file or a sequence of images.

Two different methods are used to generate two foreground masks:

1. [**cv::bgsegm::BackgroundSubtractorMOG**](http://docs.opencv.org/master/d6/da7/classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html)
2. [**cv::BackgroundSubtractorMOG2**](http://docs.opencv.org/master/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html)

The results as well as the input data are shown on the screen. The source file can be downloaded.

## Explanation

We discuss the main parts of the above code:

1. First, three Mat objects are allocated to store the current frame and two foreground masks, obtained by using two different BS algorithms.

Mat frame; //current frame

Mat fgMaskMOG; //fg mask generated by MOG method

Mat fgMaskMOG2; //fg mask fg mask generated by MOG2 method

1. Two [**cv::BackgroundSubtractor**](http://docs.opencv.org/master/d7/df6/classcv_1_1BackgroundSubtractor.html) objects will be used to generate the foreground masks. In this example, default parameters are used, but it is also possible to declare specific parameters in the create function.

Ptr<BackgroundSubtractor> pMOG; //MOG Background subtractor

Ptr<BackgroundSubtractor> pMOG2; //MOG2 Background subtractor

//create Background Subtractor objects

pMOG = [createBackgroundSubtractorMOG](http://docs.opencv.org/master/d6/d17/group__cudabgsegm.html#ga6742648a654585013544eaa300f77b57)(); //MOG approach

pMOG2 = [createBackgroundSubtractorMOG2](http://docs.opencv.org/master/de/de1/group__video__motion.html#ga2beb2dee7a073809ccec60f145b6b29c)(); //MOG2 approach

1. The command line arguments are analysed. The user can chose between two options:
   * video files (by choosing the option -vid);
   * image sequences (by choosing the option -img).

if(strcmp(argv[1], "-vid") == 0) {

//input data coming from a video

processVideo(argv[2]);

}

else if(strcmp(argv[1], "-img") == 0) {

//input data coming from a sequence of images

processImages(argv[2]);

}

1. Suppose you want to process a video file. The video is read until the end is reached or the user presses the button 'q' or the button 'ESC'.

while( (char)keyboard != 'q' && (char)keyboard != 27 ){

//read the current frame

if(!capture.[read](http://docs.opencv.org/master/d8/dfe/classcv_1_1VideoCapture.html#a473055e77dd7faa4d26d686226b292c1)(frame)) {

cerr << "Unable to read next frame." << endl;

cerr << "Exiting..." << endl;

exit(EXIT\_FAILURE);

}

1. Every frame is used both for calculating the foreground mask and for updating the background. If you want to change the learning rate used for updating the background model, it is possible to set a specific learning rate by passing a third parameter to the 'apply' method.

//update the background model

pMOG->apply(frame, fgMaskMOG);

pMOG2->apply(frame, fgMaskMOG2);

1. The current frame number can be extracted from the [**cv::VideoCapture**](http://docs.opencv.org/master/d8/dfe/classcv_1_1VideoCapture.html) object and stamped in the top left corner of the current frame. A white rectangle is used to highlight the black colored frame number.

//get the frame number and write it on the current frame

stringstream ss;

[rectangle](http://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html#ga07d2f74cadcf8e305e810ce8eed13bc9)(frame, [cv::Point](http://docs.opencv.org/master/db/d4e/classcv_1_1Point__.html)(10, 2), [cv::Point](http://docs.opencv.org/master/db/d4e/classcv_1_1Point__.html)(100,20),

[cv::Scalar](http://docs.opencv.org/master/d1/da0/classcv_1_1Scalar__.html)(255,255,255), -1);

ss << capture.[get](http://docs.opencv.org/master/d8/dfe/classcv_1_1VideoCapture.html#aeb1644641842e6b104f244f049648f94)([CAP\_PROP\_POS\_FRAMES](http://docs.opencv.org/master/dd/de7/group__videoio.html#gga04495204613753c836e07d31c575bf3ba6223452891755166a4fd5173ea257068));

string frameNumberString = ss.str();

[putText](http://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html#ga5126f47f883d730f633d74f07456c576)(frame, frameNumberString.c\_str(), [cv::Point](http://docs.opencv.org/master/dc/d84/group__core__basic.html#ga1e83eafb2d26b3c93f09e8338bcab192)(15, 15),

[FONT\_HERSHEY\_SIMPLEX](http://docs.opencv.org/master/d0/de1/group__core.html#gga0f9314ea6e35f99bb23f29567fc16e11afff8b973668df2e4028dddc5274310c9), 0.5 , [cv::Scalar](http://docs.opencv.org/master/dc/d84/group__core__basic.html#ga599fe92e910c027be274233eccad7beb)(0,0,0));

1. We are ready to show the current input frame and the results.

//show the current frame and the fg masks

[imshow](http://docs.opencv.org/master/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563)("Frame", frame);

[imshow](http://docs.opencv.org/master/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563)("FG Mask MOG", fgMaskMOG);

[imshow](http://docs.opencv.org/master/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563)("FG Mask MOG 2", fgMaskMOG2);

1. The same operations listed above can be performed using a sequence of images as input. The processImage function is called and, instead of using a [**cv::VideoCapture**](http://docs.opencv.org/master/d8/dfe/classcv_1_1VideoCapture.html) object, the images are read by using [**cv::imread**](http://docs.opencv.org/master/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56) , after individuating the correct path for the next frame to read.

//read the first file of the sequence

frame = [imread](http://docs.opencv.org/master/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56)(fistFrameFilename);

if(!frame.data){

//error in opening the first image

cerr << "Unable to open first image frame: " << fistFrameFilename << endl;

exit(EXIT\_FAILURE);

}

...

//search for the next image in the sequence

ostringstream oss;

oss << (frameNumber + 1);

string nextFrameNumberString = oss.str();

string nextFrameFilename = prefix + nextFrameNumberString + suffix;

//read the next frame

frame = [imread](http://docs.opencv.org/master/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56)(nextFrameFilename);

if(!frame.data){

//error in opening the next image in the sequence

cerr << "Unable to open image frame: " << nextFrameFilename << endl;

exit(EXIT\_FAILURE);

}

//update the path of the current frame

fn.assign(nextFrameFilename);

Note that this example works only on image sequences in which the filename format is <n>.png, where n is the frame number (e.g., 7.png).

## Results

* Given the following input parameters:

-vid Video\_001.avi

The output of the program will look as the following:

* The video file Video\_001.avi is part of the [Background Models Challenge (BMC)](http://bmc.univ-bpclermont.fr/) data set and it can be downloaded from the following link [Video\_001](http://bmc.univ-bpclermont.fr/sites/default/files/videos/evaluation/Video_001.zip) (about 32 MB).

## Evaluation

To quantitatively evaluate the results obtained, we need to:

* Save the output images;
* Have the ground truth images for the chosen sequence.

In order to save the output images, we can use [**cv::imwrite**](http://docs.opencv.org/master/d4/da8/group__imgcodecs.html#gabbc7ef1aa2edfaa87772f1202d67e0ce) . Adding the following code allows for saving the foreground masks.

string imageToSave = "output\_MOG\_" + frameNumberString + ".png";

bool saved = [imwrite](http://docs.opencv.org/master/d4/da8/group__imgcodecs.html#gabbc7ef1aa2edfaa87772f1202d67e0ce)(imageToSave, fgMaskMOG);

if(!saved) {

cerr << "Unable to save " << imageToSave << endl;

}

Once we have collected the result images, we can compare them with the ground truth data. There exist several publicly available sequences for background subtraction that come with ground truth data. If you decide to use the [Background Models Challenge (BMC)](http://bmc.univ-bpclermont.fr/), then the result images can be used as input for the [BMC Wizard](http://bmc.univ-bpclermont.fr/?q=node/7). The wizard can compute different measures about the accuracy of the results.

# Lecture 3: Object detection