Installation Cheat Sheet 1 - OpenCV 3 and C++
Using Windows 10 + Visual Studio 2013 or 2015 (Community Edition) + precompiled binaries

Click here to go to the YouTube video for this Cheat Sheet

GitHub page with all Cheat Sheets and code

If you found this Cheat Sheet helpful please consider supporting me on Patreon

Note: **Bold blue** indicates something that will change depending on your version of OpenCV and/or Visual Studio

1a) Download and install Visual Studio **2013** or **2015** Community Edition (with Exception stated in point **1b)**
(yes, both are free, and choosing all default options will work fine)
1b) Exception: As of OpenCV **3.0.0**, the pre-compiled binaries are only available for Visual Studio **2013** (not **2015**), therefore **if you are using OpenCV 3.0.0, you will have to use Visual Studio 2013 for the remainder of this cheat sheet / video**. If you are using any version of OpenCV **after 3.0.0** then either Visual Studio **2013** or **2015** will work.

2) Download the latest version of OpenCV, for example OpenCV **3.0.0** as of when this is being written

3a) Make a folder "C:\OpenCV-**X.X.X**" for your version of OpenCV, ex. "C:\OpenCV-**3.0.0**"
3b) Double-click on the completed OpenCV download executable, then set "Extract to:" to your "C:\OpenCV-**X.X.X**" directory

4a) Add the **bin** directory for your version of OpenCV and Visual Studio to the operating system PATH.
4b) Note: In the OpenCV directories:
"vc**12**" => Visual Studio **2013**
"vc**14**" => Visual Studio **2015**
For example if you are using OpenCV **3.0.0** and Visual Studio **2013** add the following to your PATH:
C:\OpenCV-**3.0.0**\opencv\build\x86\vc12\bin
4c) Pull up a Command Prompt and verify the bin directory is now in PATH, then reboot

5) From my MicrocontrollersAndMore GitHub page decide which example you are going to use:
CannyStill.cpp (uses a still image)
CannyWebcam.cpp (uses a webcam)
RedBallTracker.cpp (tracks a red ball, uses a webcam)

If you are going through this for the first time I suggest CannyStill.cpp

6a) Start Visual Studio, make a new project
6b) Choose Visual C++, Win32 Console Application, name as you prefer, ex "CannyStill1", set preferred location, uncheck "Create directory for solution" and "Add to source control", choose OK
6c) On the **“Welcome to the Win32 Application Wizard”** screen choose Next
6d) On the **"Application Settings"** screen, uncheck "Precompiled Header" and "Security Development", check

"Empty Project", and verify "Console application" radio button is checked, then choose Finish

7a) Right click in Solution Explorer, choose Add -> New Item
7b) Choose "C++ File", name the C++ file as preferred, ex. "CannyStill1.cpp", choose "Add"
7c) Copy/paste the entire code from your chosen example into the .cpp file
(At this point Visual Studio will underline many of the lines of code with red because we have not yet informed Visual Studio as to the location of OpenCV, subsequent steps will resolve this)

8) If you are using an example with a still image (i.e. CannyStill.cpp), copy any JPEG image into the project directory and rename it "image.jpg". You can use the "image.jpg" from my MicrocontrollersAndMore GitHub page if you would like to see the same results as in the video (if you are using a webcam example then this step does not apply).

9) In VS go to:
Project -> Properties -> Configuration Properties -> VC++ Directories -> Include Directories
add the **include** directory for your version of OpenCV, ex "C:\OpenCV-**3.0.0**\opencv\build\include"

10) In VS go to:
Project -> Properties -> Configuration Properties -> VC++ Directories -> Library Directories:
add the **library** directory for your version of OpenCV, ex "C:\OpenCV-**3.0.0**\opencv\build\x86\vc12\lib"

11) If you currently *do not* have Windows 10 configured to allow viewing / editing of file extensions, go to: right-click on Start -> Control Panel -> View by: Large icons -> File Explorer Options -> View tab -> uncheck "Hide extensions for known file types" (if you already have viewing file extensions enabled then skip this step).

12) In File Explorer (not within Visual Studio), navigate to the **lib** directory, ex
C:\OpenCV-**3.0.0**\opencv\build\x86\vc12\lib
In the lib directory you will find the debug libs (ending with a 'd'), for example if you are using OpenCV 3.0.0 the debug libs are the following:

opencv_ts**300**d.lib
opencv_world**300**d.lib

Note the **300** is referring to OpenCV **3.0.0** and this will vary by OpenCV version, also which libraries are present may vary with OpenCV version.

Copy/paste the name of each debug lib in the lib directory into the following location in Visual Studio:
Project -> Properties -> Configuration Properties -> Linker -> Input -> Additional Dependencies

13) In the Visual Studio toolbar, verify that "Solution Configurations" and "Solution Platforms" are set to "Debug" and "Win32", respectively

14) Run the program, either without debugging (choose Debug, then the hollow green arrow, or press Ctrl+F5) or with debugging (solid green arrow or press F5)

15) Once you get the **debug** build working, if it is too slow and you would like to try the **release** build, at the top of Visual Studio change "Solution Configurations" to "Release", then repeat steps 9 through 12, making the applicable changes for a **release** configuration.

As of OpenCV **3.0.0**, when using the precompiled binaries as described in this document, the Include directory (step 9) and the Library directory (step 10) are the same for **debug** or **release**, i.e. use the same Include and Library directory paths as above if using OpenCV **3.0.0**). This may change in a future version of OpenCV so be sure to check this if you are using a version of OpenCV after **3.0.0**.

For the .lib files themselves (step 12), when configuring for a **release** build, make sure to use the .lib files without the 'd' just before the .lib, i.e. if you are using OpenCV **3.0.0**, use the following .libs:

opencv_ts**300**.lib
opencv_world**300**.lib

Once you have the **release** build configured, you can switch back and forth between the **debug** build and the **release** build simply by changing "Solution Configurations" back and forth between "Debug" and "Release".