Connecting OpenCV with Cmake to Unity

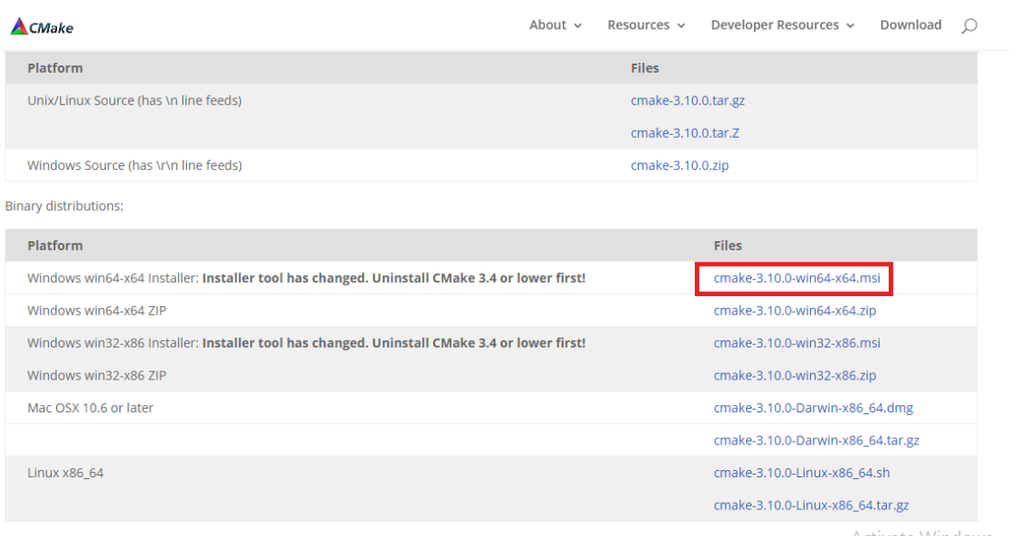
Hiba Jamal

February 2019

Since I came across many tutorials, some buggy while others incomplete to fulfill my purposes. So I'll put the entire process down, alongside the code that I used so that you don't have to go through the agony I did when I ventured out to do this at first. So let's get into it.

1 Install Cmake

Head over to `https://cmake.org/download/` and download the **installer** for the latest version of CMake. Example:



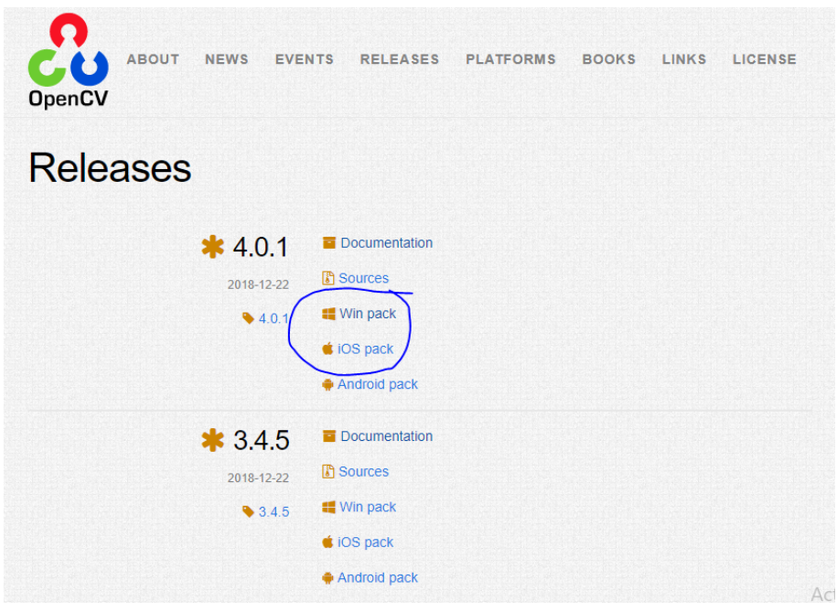
Remember to "Add CMake to system PATH" during installation.

2 Install OpenCV

Firstly download these source files, extract them in the same directory for ease later down this process:

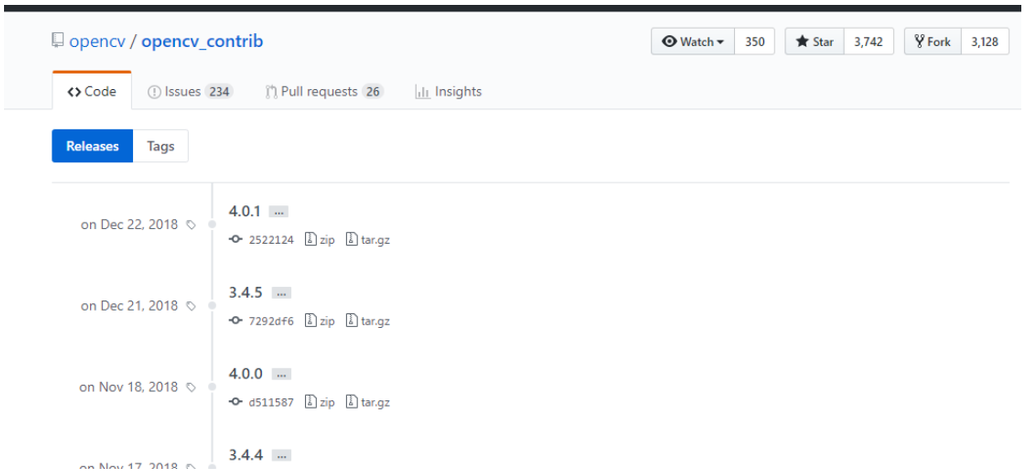
- `https://opencv.org/releases.html`

  Go for the self-extracting archive according to your machine. The latest version on this uptil now is 4.0.1, download the version you need:



- `https://github.com/opencv/opencv_contrib/releases`

Matching whatever you downloaded in the last step, download the same zip file from this link, in my case, it would be 4.0.1.



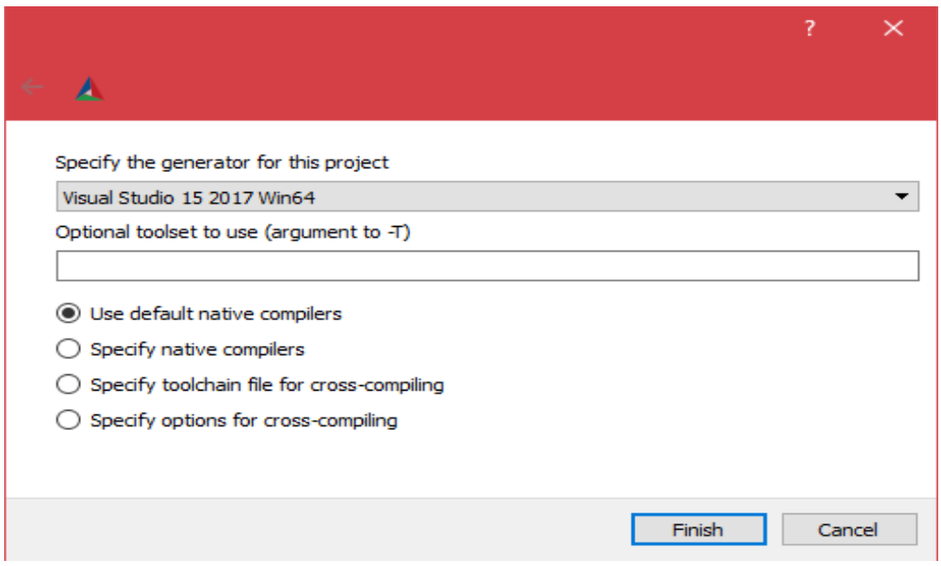
3 Create a Video Studio project using CMake

Now open up CMake (cmake-gui). Say I extracted both opencv and opencv_contrib in my C drive. In the "Where is the source code:" I would give the path of where I extracted opencv/sources, whereas in the "Where to build binaries:" space I would provide a path to where I want the resulting cmake generated files to be stored. For ease, it could just be a folder inside the opencv folder itself! For example, mine is called 'cmake_compilation'.

You can either create this folder yourself or provide the path (eg. C:/opencv/cmake_compilation) where it will create the 'cmake_compilation' folder itself - after showing you a dialog box asking for your permission.

Now click **Configure**! (This is the part where it shall ask for your permission to create the folder you've provided - if you have already created the folder yourself, forgive me for making you read all the bs in this bracket.)

Once you click **Configure** you will be prompted to select a generator for the project, SELECT THE Visual Studio {Version} **Win64** option, wallahi if you forget the Win64 and just select your Visual Studio {Version} you will regret it later down this process. Oh yes and keep the default compilers selected.

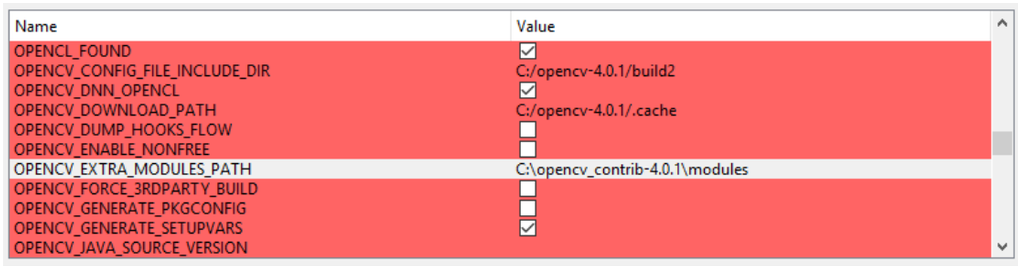


CMake will now start checking your system directories and start generating the appropriate files needed. You can see the progress in the white space under the Configure option.

3.1 Additional changes to CMake configuration according to what you may need (only for C and its variants in this case!)

Once configuration is done you will see a whole red area filled with items alongside checkboxes. Now CMake has already accounted for most things that you might need but here are a few more to check off:

- Check "INSTALL_C_EXAMPLES"
- Now look for **OPENCV_EXTRA_MODULES_PATH**, and provide the path for the extra ocv modules in our opencv_contrib folder:



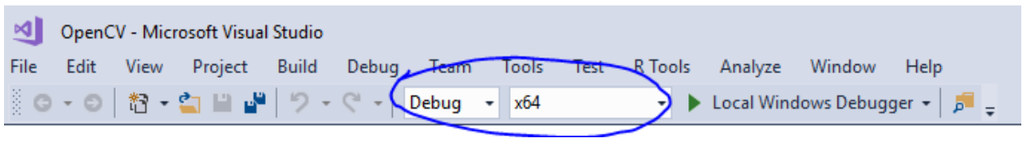
- Click Configure once again!
- If by chance you are on Windows 10, opencv_saliency module fails to build, so go ahead and uncheck **BUILD_opencv_saliency**.
- Click Configure again.

4 Compile OpenCV

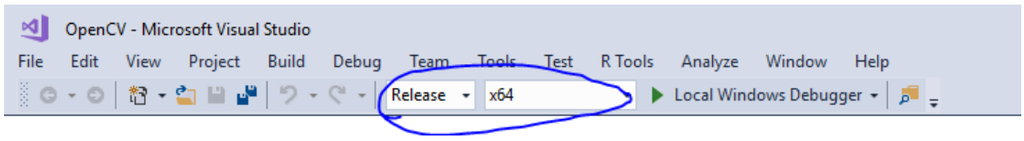
Once Configuration is complete (without any errors you hope), you should proceed to click generate so your Visual Studio .sln will be ready!

Now once the Visual Studio project is open, do not click any of the various folders, just proceed building the project **with the following specifics** (also remember this part takes a great lot of time, so find an episode to watch or just practice being patient):

- In the Solution Configurations option, set it to **Debug**, and keep the Solution Platform to **x64** (since our OpenCV version is also most likely 64 bit). Now build.



- Once the build is successfully completed, now set the Solution Configuration to **Release**, and again make sure the Solution Platform is **x64**. Now build.



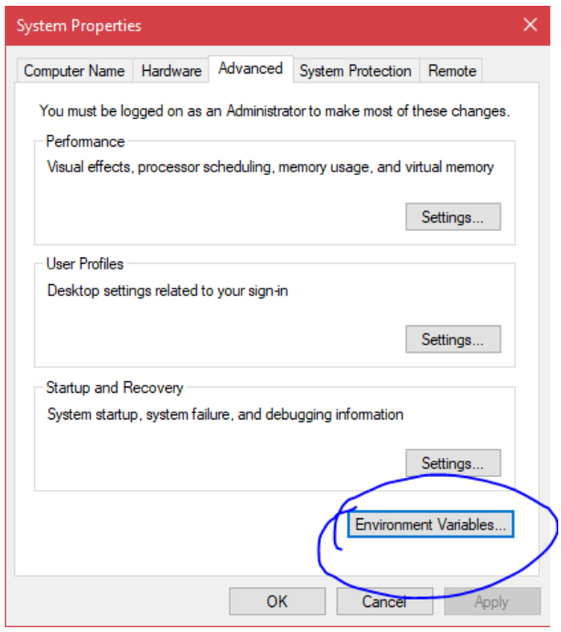
5 Updating System Variables

Before we head to setting environment variables, make another directory perhaps called **opencv4.0.1** and place it simply in the same directory you extracted the above folders to - I've placed it inside the opencv directory itself, hence my path is C:\OpenCV \OCV 4.0.1. In this folder place the following:

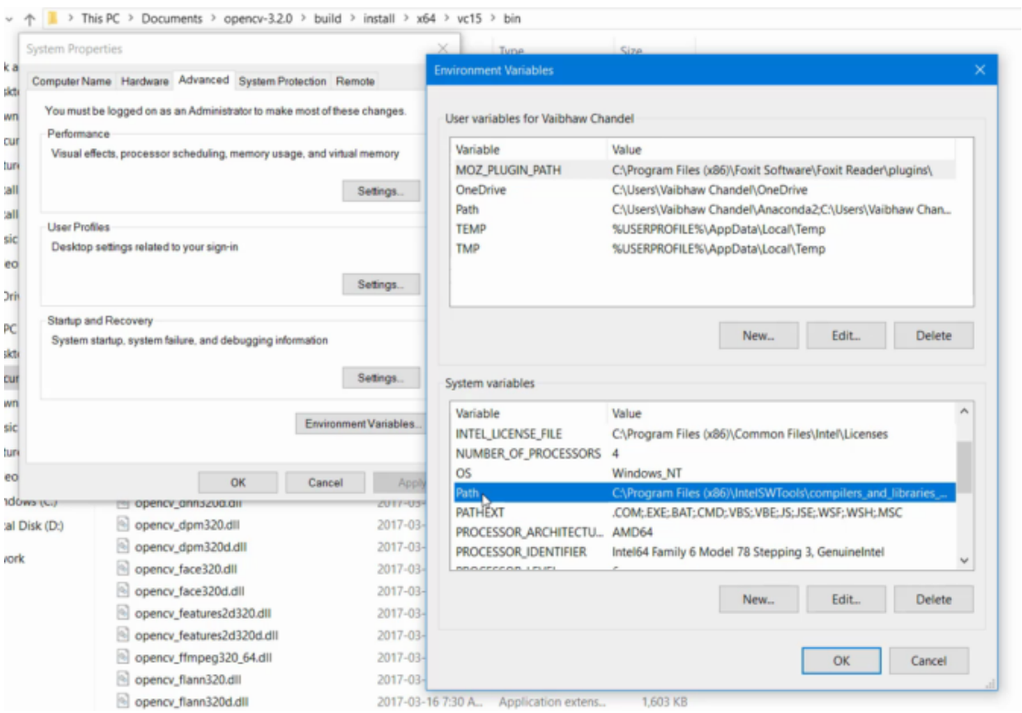
- In the original extraction of opencv, navigate to opencv/build and copy the **include** folder and paste it in this new folder.
- Now in your cmake_compilation folder (or your variation of it), copy the **bin** and **lib** folders from it and paste it in this folder too!

You now must have 3 folders inside this new folder, namely, include, bin, and lib.

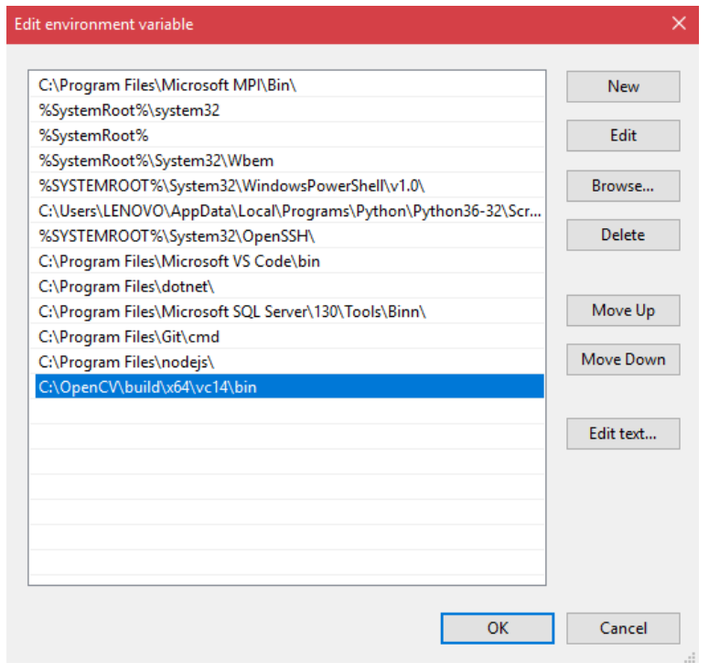
Now we move towards setting the system variables, head over to System Properties:



- Under System Variables, Select Path and click edit.



Click New, and give path to {wherever you unpacked opencv}\build \x64 \vc14 \bin and click Ok. Depending upon where you have kept opencv folder and what version of Visual Studio you used to compile OpenCV, this path would be different. For me, it was:



Click Ok to save.

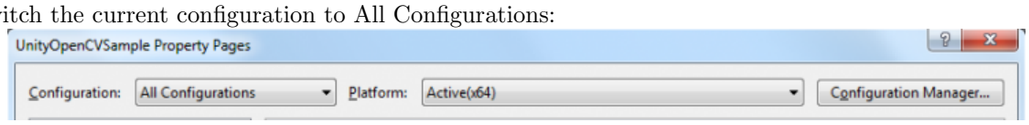
- Under "System variables", add a new entry called OPENCV_DIR and provide the path to the new folder we created in the last step (for me it was C:\OpenCV \OCV 4.0.1)
- Finally, in the User variables section above the system variables, find the PATH entry, and click edit. At the end, add %OPENCV_DIR%\bin.

6 Make a Project to finally be able to run OCV

Now create a new C++ project and add an empty .cpp file to it. **Make sure the platform is x64**.

6.1 Open project properties:

Switch the current configuration to All Configurations:



Now make the following changes in project properties:

- Configuration Properties >General >Target Extension = .dll
  Under Project defaults change Configuration Type to the same.
- C/C++ >General >Additional Include Directories = $(OPENCV_DIR)\include
  (Remember we defined OPENCV_DIR as a system user variable, pointing to where we copied the OpenCV .dll's, .lib's and .hpp's (includes))
- **Change configuration from All Configurations to Debug**

  Now in Linker >General >Additional Library Directories = $(OPENCV_DIR) \lib \Debug

  In Linker >Input >Additional Dependencies =
  opencv_core401d.lib;opencv_highgui401d.lib;opencv_objdetect401d.lib;opencv_videoio401d.lib;opencv_imgproc401d.lib
- **Change configuration from All Configurations to Release**

  Now in Linker >General >Additional Library Directories = $(OPENCV_DIR) \lib \Release

  In Linker >Input >Additional Dependencies =
  opencv_core401.lib;opencv_highgui401.lib;opencv_objdetect401.lib; opencv_videoio401.lib;opencv_imgproc401.lib

Now test your setup by adding the following lines:

```
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
```

And build this in both Debug and Release versions.

7 Pass facial detection to Unity

7.1 Create the dll file:

Copy the following code in the link provided to the main cpp file in your project:
`https://drive.google.com/file/d/1g5ubU26AGVTsiEvGhDqHiUheyo609PPb/view?usp=sharing`

Build this in Release mode (x64 of course), and you will have a .dll file ready made inside the folder containing your project. Hold on to that.

7.2 Create your Unity project:

Create a simple Unity project and place a 3D object on your screen. Attach no components to it, just place it on your screen.

Now head over to your Assets/Plugins folder and paste the .dll file that you attained in your last step. Along with this other OpenCV dlls will also be needed, but I'm not concerned with filtering out those for my purpose right now. So just head over to the folder you created where you placed your include, bin and lib folders to (for me that would be C: \OpenCV \OCV 4.0.1), and go into bin \Release. Copy all the dlls from there into your Assets/Plugins folder as well!

Now copy the cascade classifier .xml. Following is a link to doownload the lbp frontal face cascade xml file:
`https://drive.google.com/file/d/1o4FVk2CJ4FBJI3sV8V8L6a47AryYhCNX/view?usp=sharing`

Copy this to your root folder containing your Unity project. For example, mine would be C: \Users \LENOVO \Documents \try1 since my Unity project is saved in My Documents and is called try1 - creative name I know.

7.2.1 Unity scripts:

Create a new script called OpenCVFaceDetection, and follow the link below and copy the code to the generated class:
`https://drive.google.com/file/d/1rIC_Nxs91uNP-15vOqKm7nJVzRm3jB6C/view?usp=sharing`
For the explanation for this next bit, here's an excerpt from the base tutorial this tutorial is based on: `http://thomasmountainborn.com/2017/03/05/unity-and-opencv-part-three-passing-detection-data-to-unity/`
Ps. the Update() function is of the code provided above.

> The important bit happens in Update(): in an `unsafe` block, we call OpenCVInterop.Detect(), and pass the `fixed` pointer of an array of CvCircle. This means that the C++ OpenCV code will write the detected faces directly into this struct array we defined in C#, without the need for performance heavy copies from unmanaged space into managed space. This is a good trick to know for any C++ interop you may have to do in the future.
>
> Because we don't know how many faces will be detected, we create the array at a predefined size, and ask our C++ code to tell us how many faces were actually detected using a by ref integer. We also pass the array size to C++ to prevent buffer overflows.
>
> In case you are not familiar with the two above keywords, `unsafe` simply allows you to use pointers in C#, and `fixed` tells the compiler that the given variable has to stay at its assigned position in memory, and is not allowed to be moved around by the garbage collector – otherwise the C++ code could inadvertently be writing to a different bit of memory entirely, corrupting the application.
>
> This same procedure can be used to pass an array of pixels between OpenCV and Unity without having to copy it, allowing you to display video footage from OpenCV within Unity, or passing a WebcamTexture stream to OpenCV for processing. That is beyond the scope of this part, however.
>
> In order to be able to use `unsafe`, we need to add a file called "mcs.rsp" to the root asset folder, and add the line "-unsafe" to it. (in versions before 5.5 you may need to use either smcs.rsp for .NET 2.0 subset, or gmcs.rsp for the full .NET 2.0). This file is an instruction to the compiler to allow unsafe code.
>
> While this will let Unity compile your scripts, Visual Studio will still complain when you try to debug with an `unsafe` block – normally you add a flag in the project properties, but Visual Studio Tools for Unity blocks access to those. To be able to debug, you will have edit the .csproj (root project folder) manually, and set the two `<AllowUnsafeBlocks>false</AllowUnsafeBlocks>` lines to true. You will have to do this after every script change since the .csproj is recreated by Unity after every compile, so it'll be useful to comment out the `unsafe` lines when you're working on something else in the project.

The msc.rsp file is in the link provided:
https://drive.google.com/open?id=1FRYG8UajTHdnQVzhRyGVDNxXdiJJrzzA
As stated above, keep this in your assets folder.

Create another script called PositionAtFaceScreenSpace, take it from the link below, it sends the face position to other Unity scripts:
https://drive.google.com/file/d/1RRAFfn7TNyLPH3pv1iR4rajk1hsw1Tw6/view?usp=sharing

Save this to your root assets folder as well.

Finally, FINALLY, drag the scripts (OpenCVFaceDetection and PositionAtFaceScreenSpace) and place them onto the 3D object you placed on the screen. Once all this is in place, enter Play Mode.

8 Credits

https://www.learnopencv.com/install-opencv3-on-windows/
http://thomasmountainborn.com/2016/09/11/unity-and-opencv-part-one-install/
http://thomasmountainborn.com/2016/09/12/unity-and-opencv-part-two-project-setup/
http://thomasmountainborn.com/2017/03/05/unity-and-opencv-part-three-passing-detection-data-to-unity/
And of course StackExchange.

This tutorial is a combination of everything I found in the tutorials above, it has added everything I struggled with additionally. But other than that, all is owed to the above mentioned links.