Module 0.3.2

Getting started Windows (using VS GUI)

Satya Mallick, Ph.D.

June 19, 2017

Table of Contents

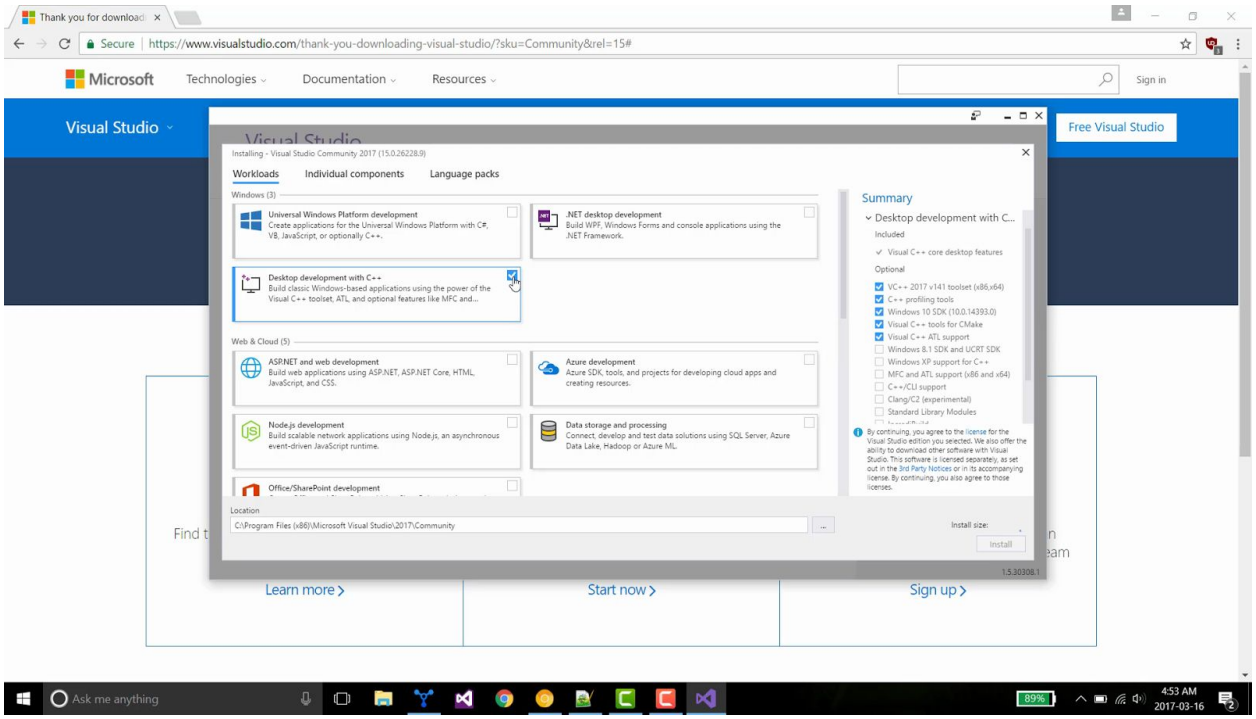
Installing OpenCV on Windows

In this tutorial, we will learn how to use Visual Studio GUI to build libraries and create projects. Visual Studio GUI can be confusing if you are not much familiar with Debug/Release mode and platform options such as x86/x64/Win32. If you would like to use command line or face issues related to linking error, please check Module 0.3.1.

We have used Windows Power Shell to run commands. Alternatively, you can use the command prompt too.

Step 1: Install Visual Studio

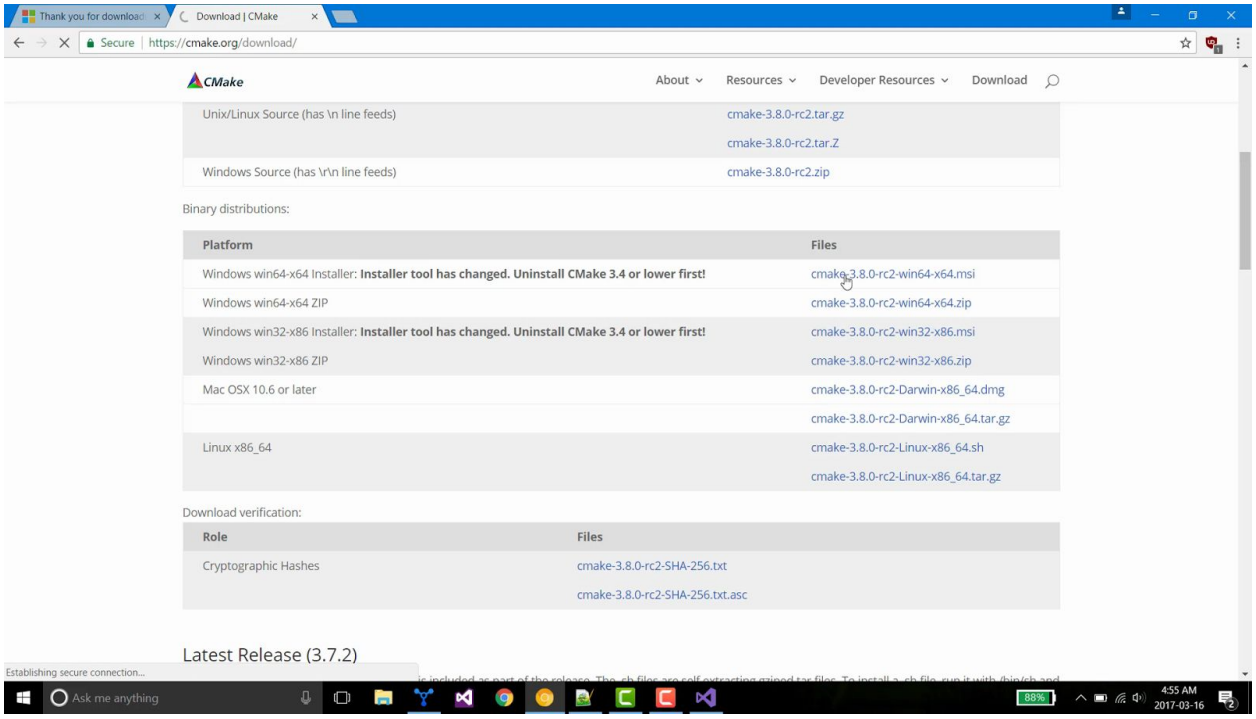
Download and install Visual Studio 2015 community edition. Make sure that you select “Desktop Development with C++” while installing Visual Studio.

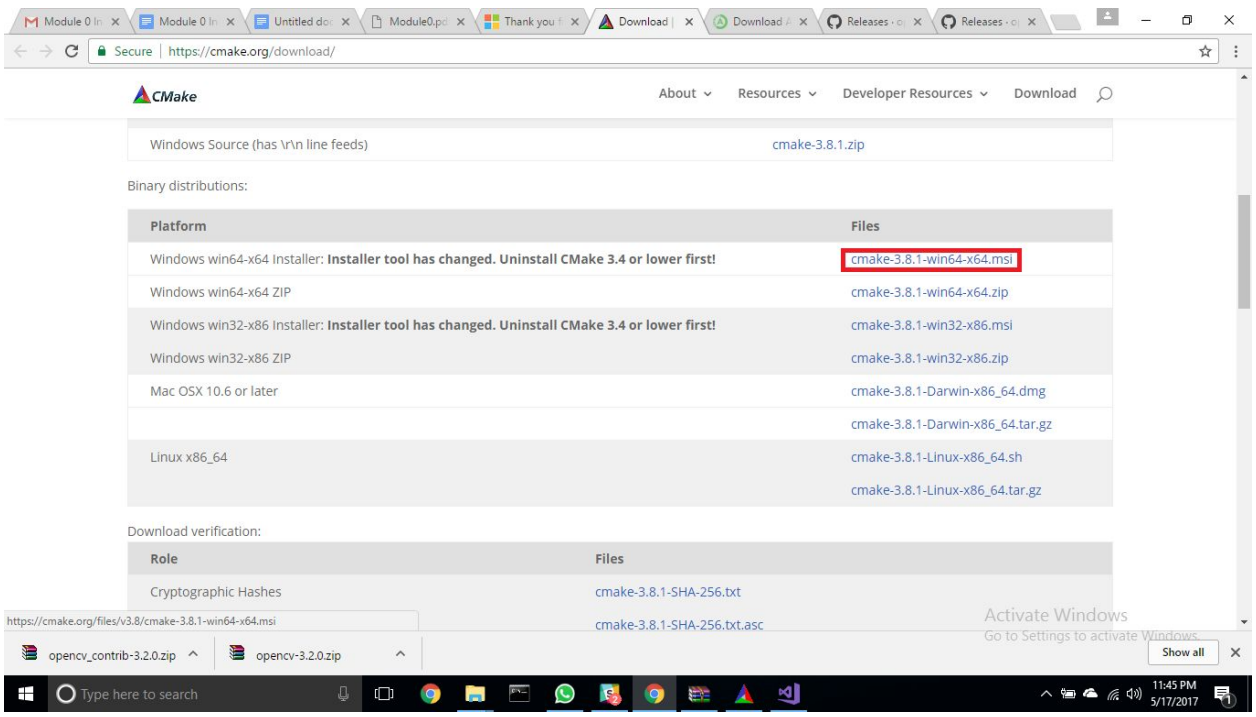


Note: Visual Studio 2017 fails to compile Dlib. We switched back to Visual Studio 2015. In a few screenshots you will see Visual Studio 15 2017.

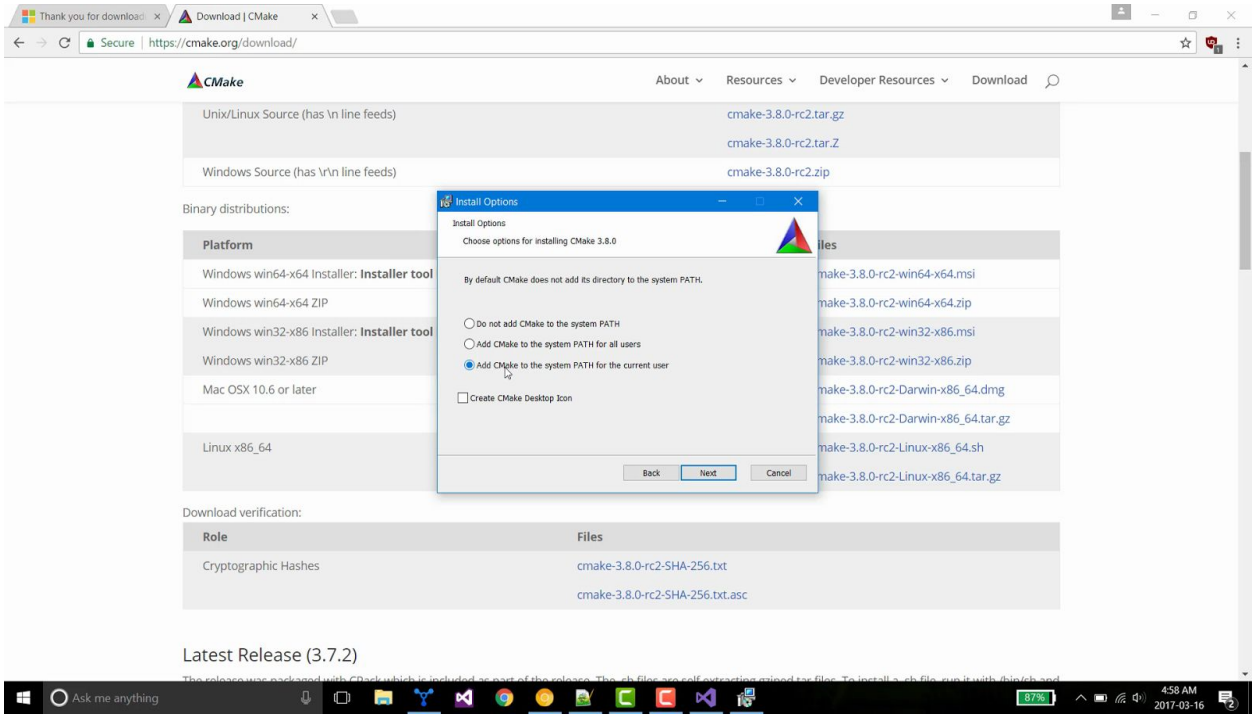
Step 2: Install CMake

Download and install CMake 3.8.2 from https://cmake.org/download/



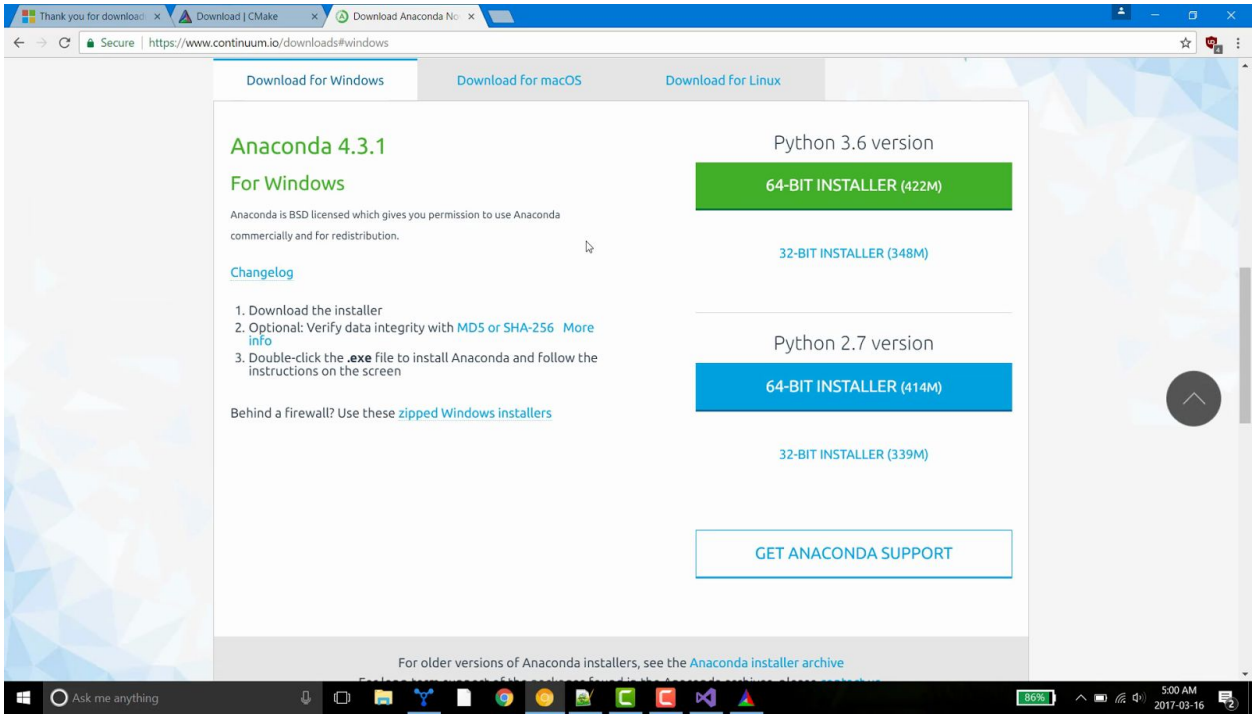


During installation select “Add CMake to system PATH”



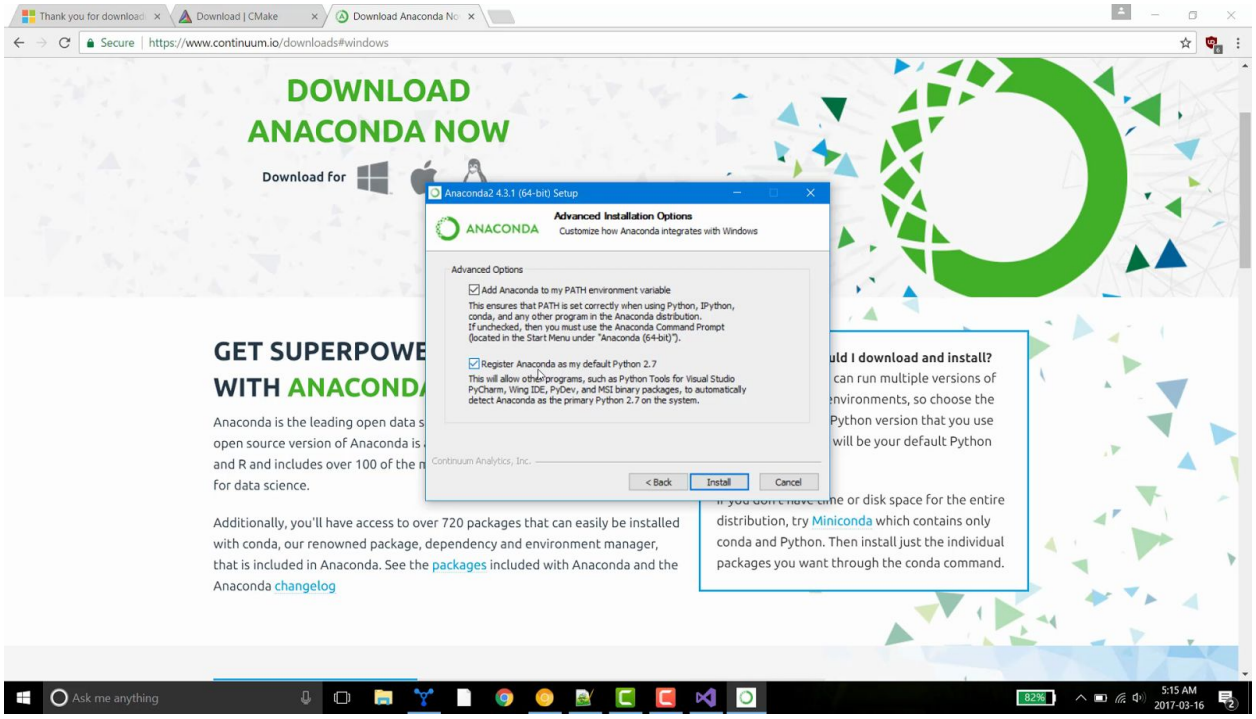
Step 3: Install Anaconda (a python distribution)

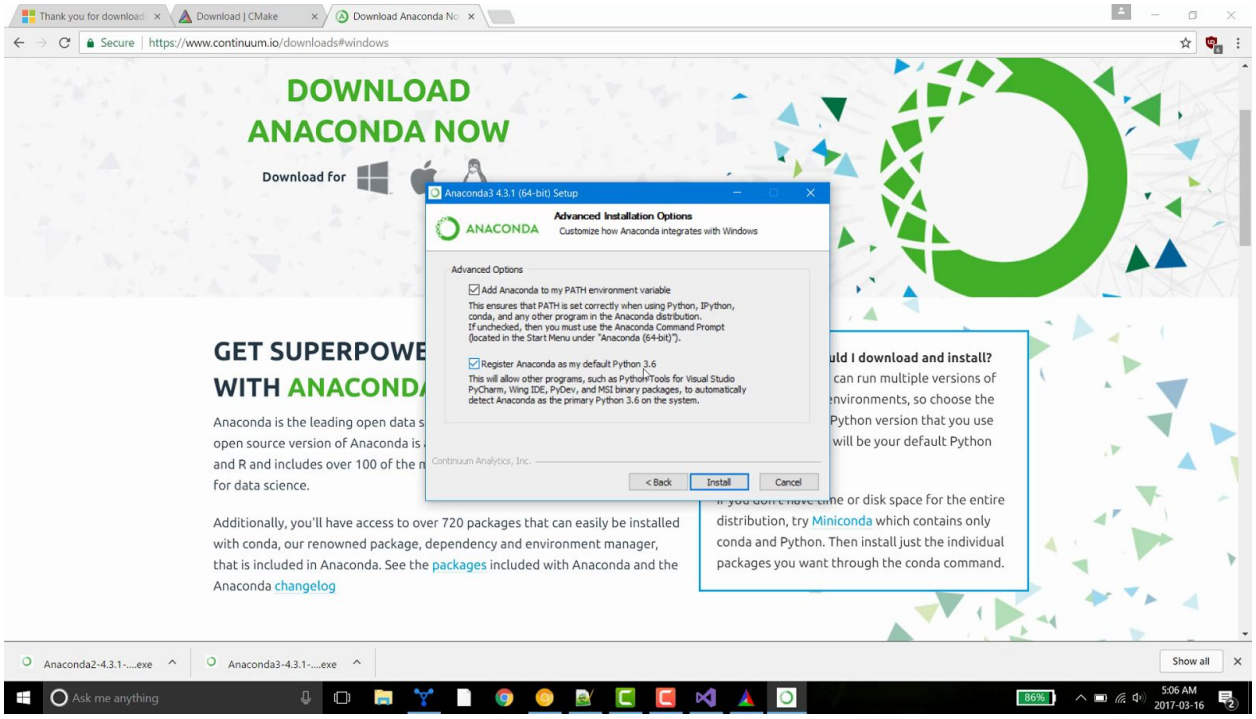
Download and install Anaconda 64-bit version from https://www.continuum.io/downloads. You can install Anaconda 2 or Anaconda 3, or both.



While installing Anaconda, make sure that you check both options:

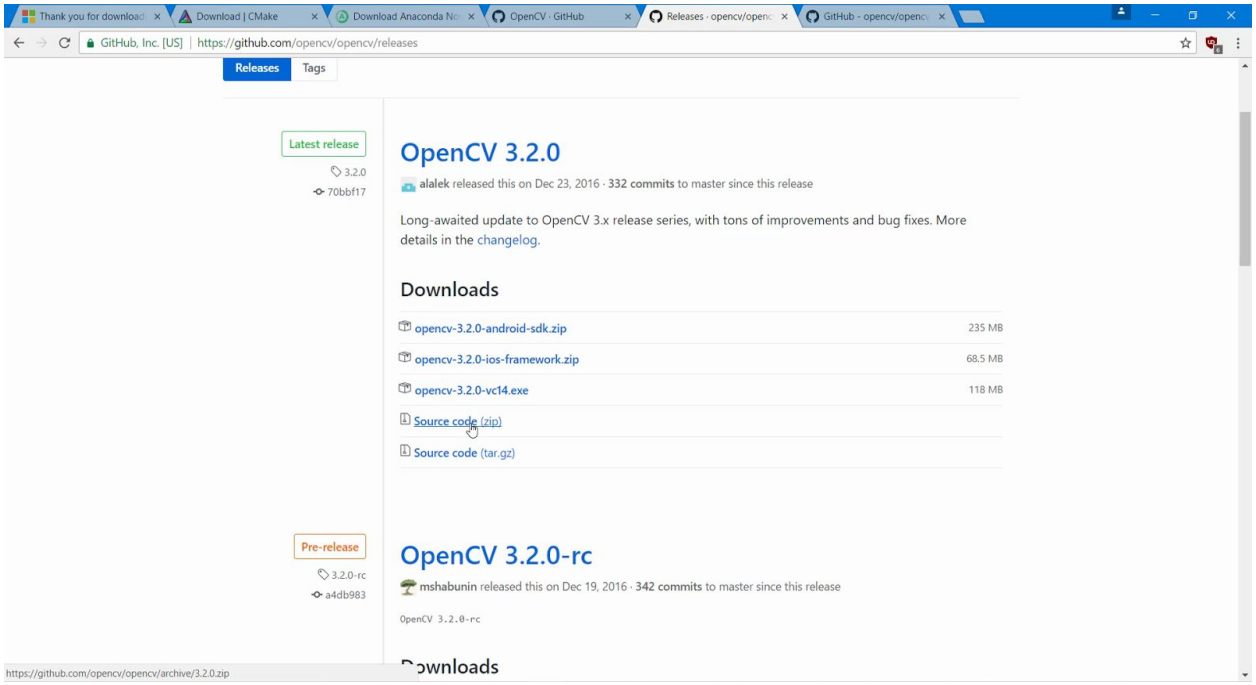
a) Add Anaconda to my PATH environment variable
b) Register Anaconda as my default Python



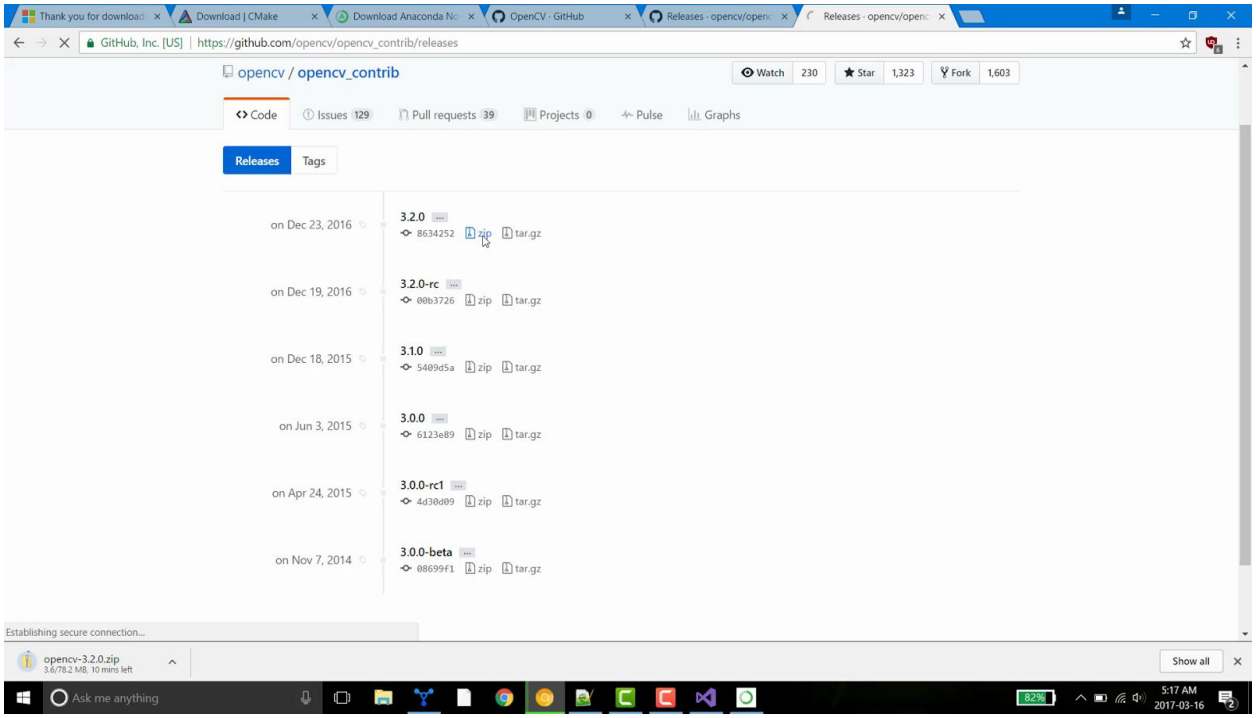


Step 4: Download and extract opencv-3.2.0 and opencv_contrib-3.2.0

Go to https://github.com/opencv/opencv/releases and download opencv-3.2.0 source code zip



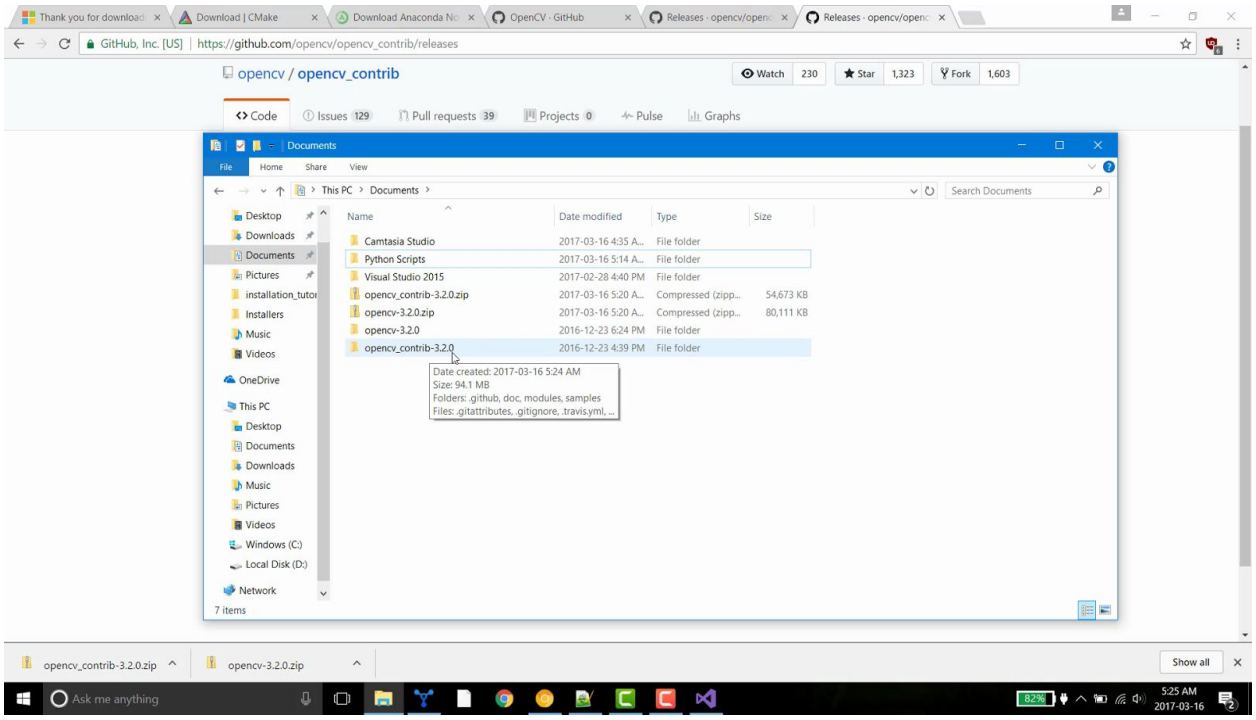
Go to https://github.com/opencv/opencv_contrib/releases and download opencv_contrib-3.2.0 source code zip



Extract both zip files. Although you can keep opencv and opencv_contrib folders anywhere, I suggest that you should keep both in the same directory. I have placed these two folders in “My Documents” directory.

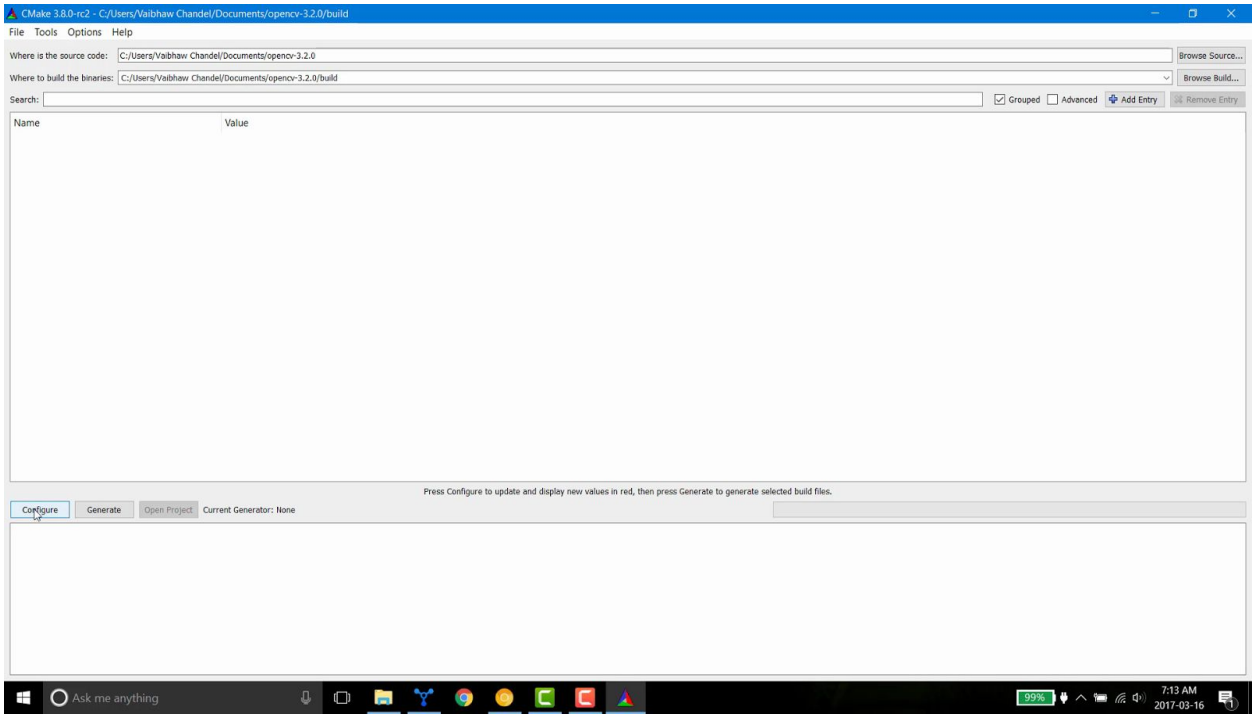
From hereon we will refer to the path to opencv-3.2.0 folder as **OPENCV_PATH**. In my case OPENCV_PATH is **C:/Users/Vaibhaw Chandel/Documents/opencv-3.2.0**

Depending upon where you have kept opencv-3.2.0 folder, this path would be different.



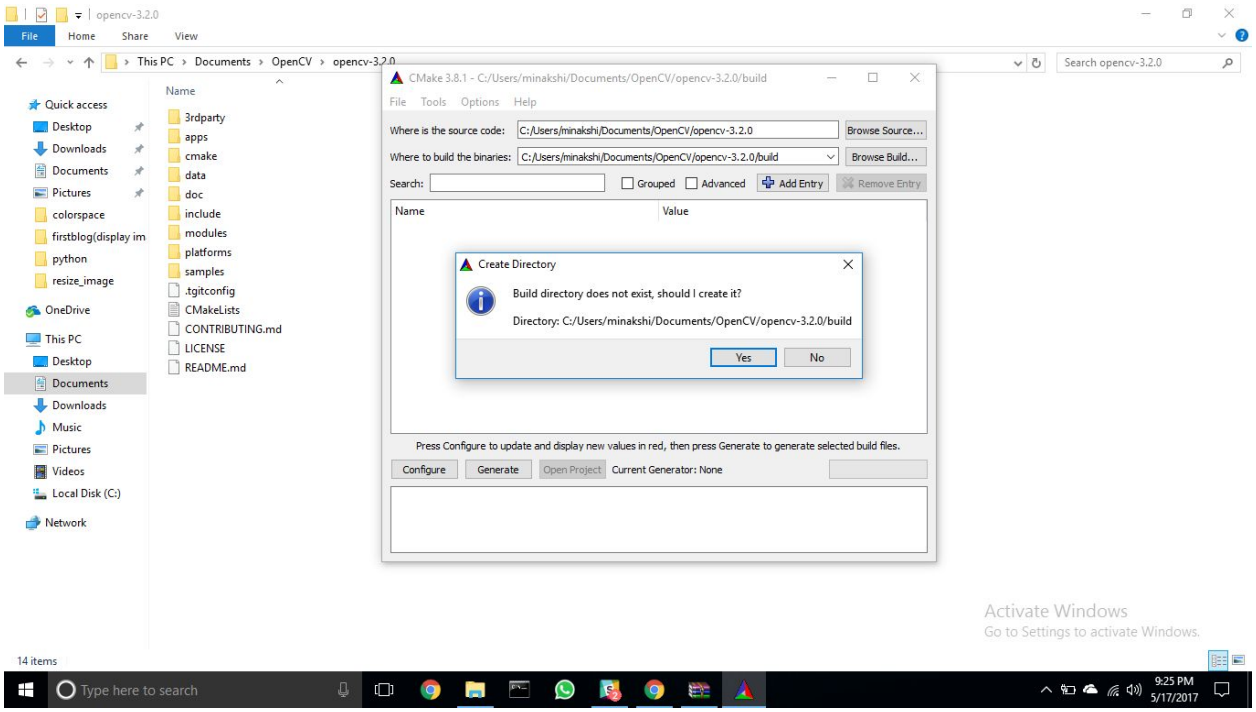
Step 5: Generate Visual Studio project using CMAKE

Run Cmake, in box “Where is the source code” write value of **OPENCV_PATH** (which is path to opencv-3.2.0 folder) and path to build directory. We will choose build directory as **OPENCV_PATH**/build

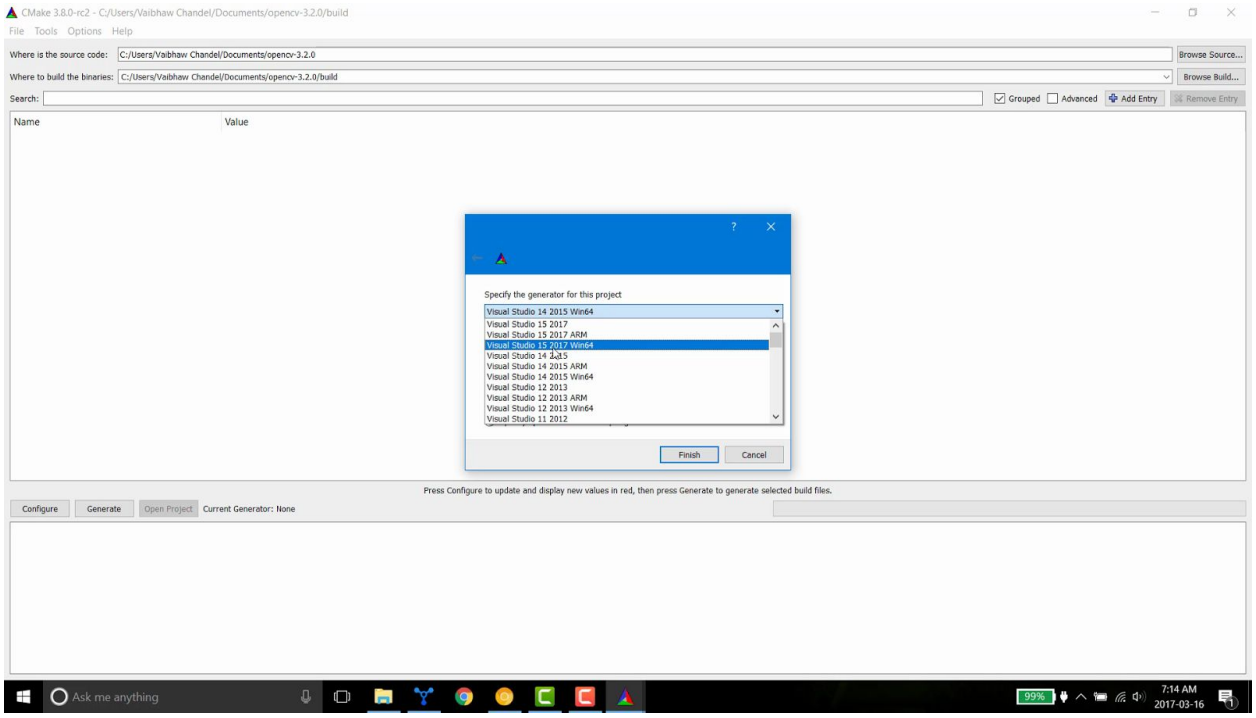


Now click configure.

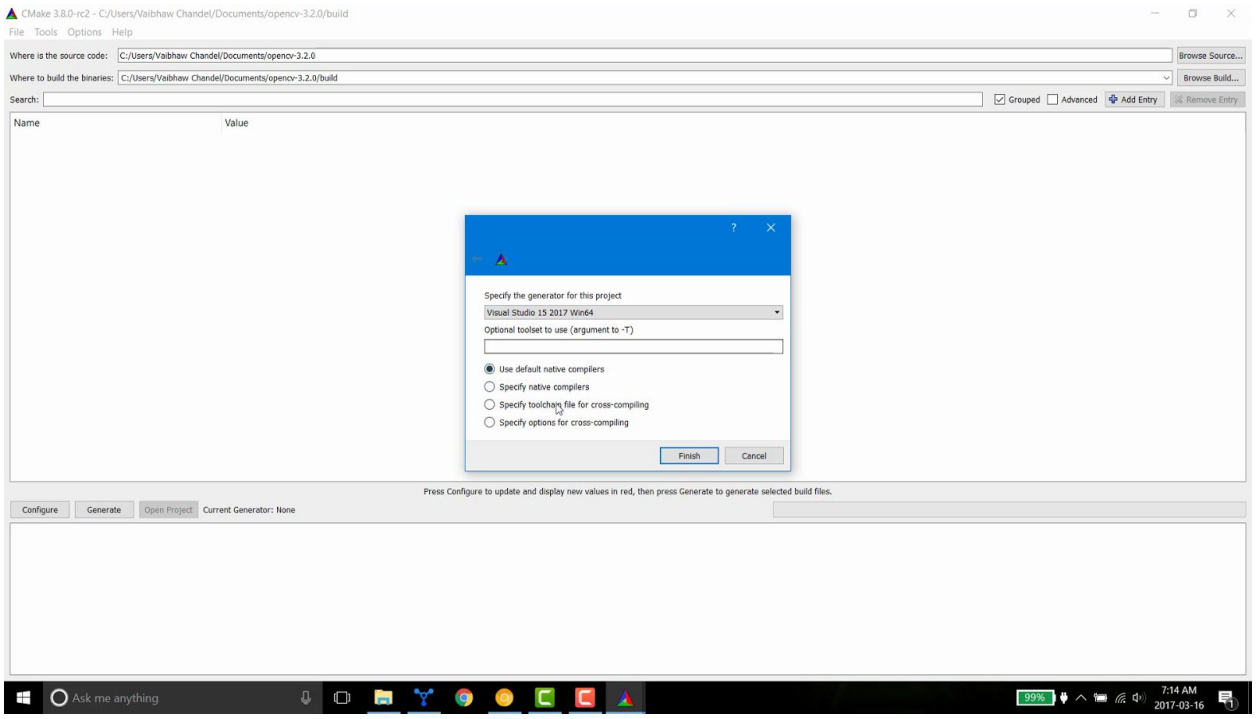
It will ask you for permission to create the build folder. Just click Yes.



It will also ask you to choose a compiler. Since we are using Visual Studio 2015 we will select **Visual Studio 14 2015 Win64**.



Click finish and in next window keep the default parameters checked.

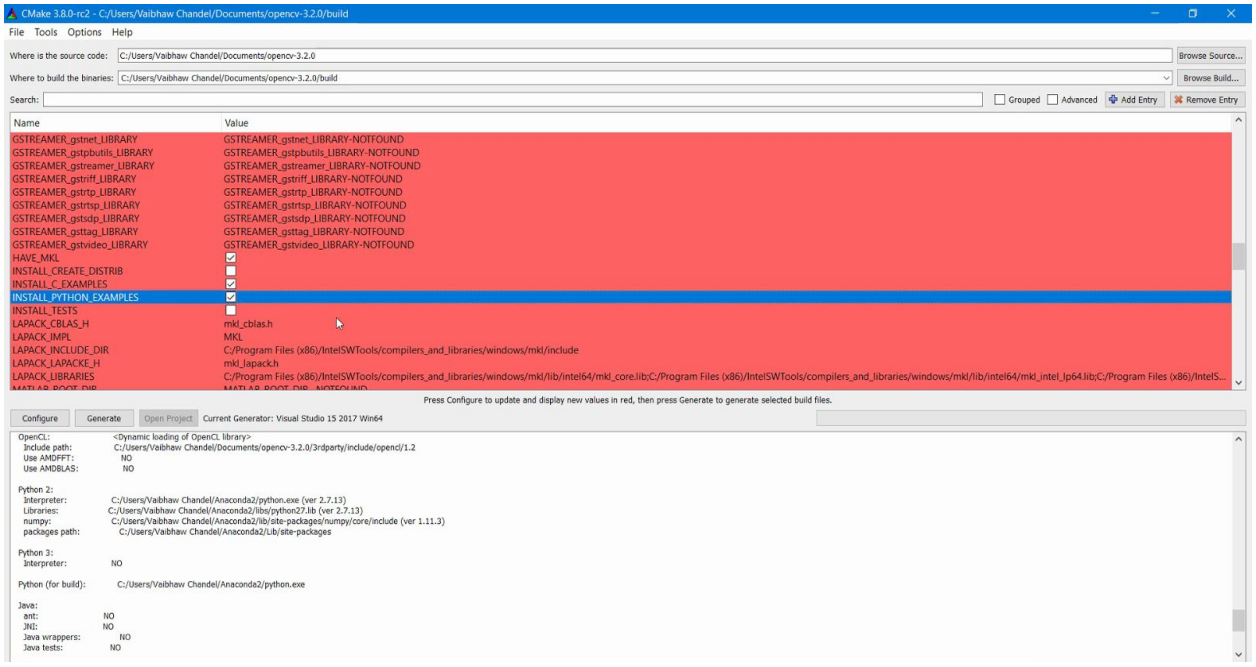


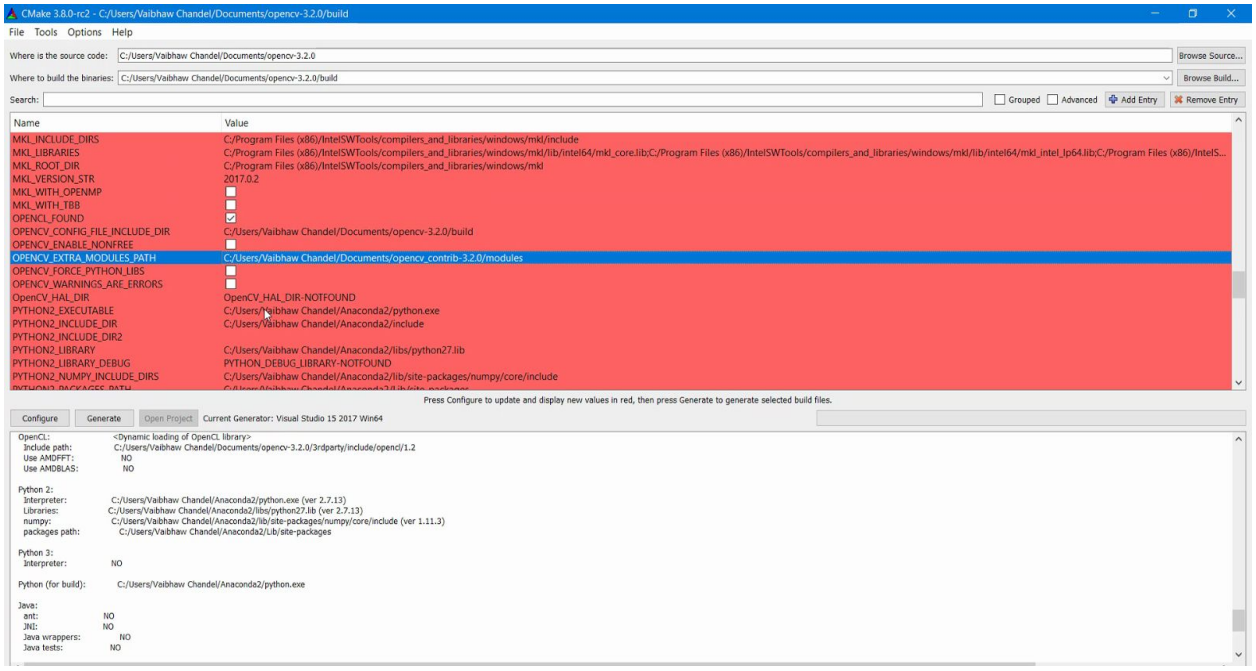
Click finish. Now CMake will look in system directories and generate makefiles.

Step 5.1 : Additional changes to CMake config

We will make few changes in the default configuration generated by CMake.

a) Check “INSTALL_C_EXAMPLES” and “INSTALL_PYTHON_EXAMPLES”
b) In flag “OPENCV_EXTRA_MODULES_PATH”, give path of modules directory within opencv_contrib-3.2.0. In our case we have kept opencv_contrib-3.2.0 in Documents folder so path is “C:/Users/Vaibhaw Chandel/Documents/opencv_contrib-3.2.0/modules”



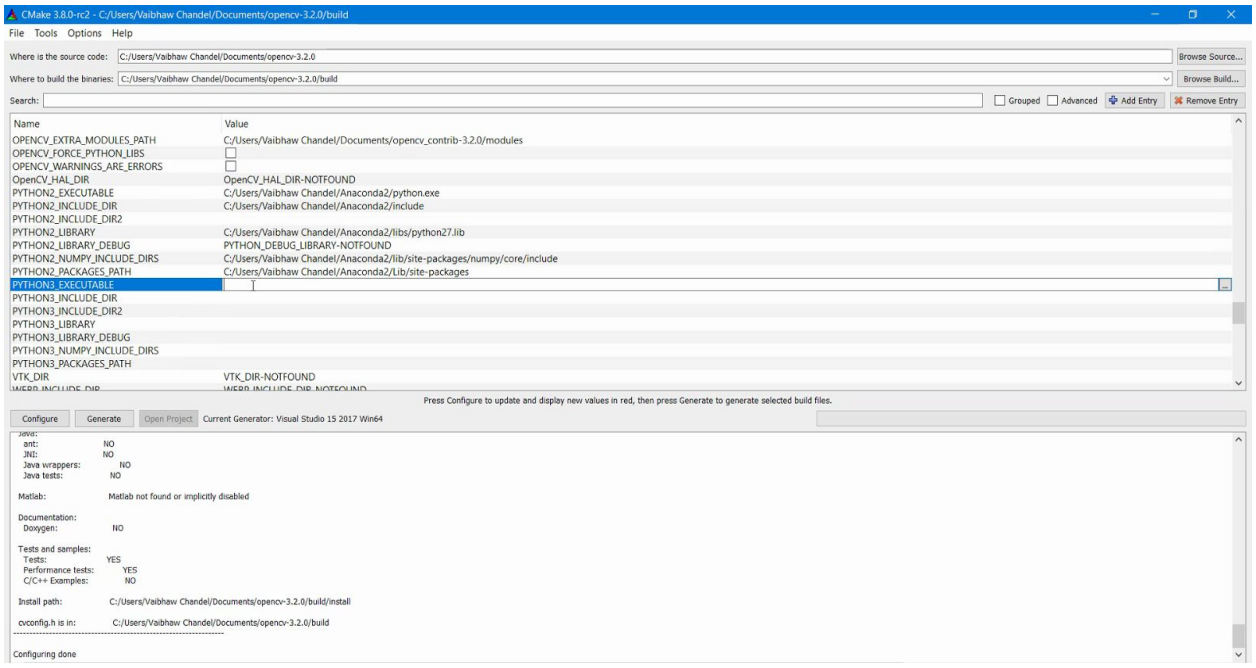


Now click on configure again.

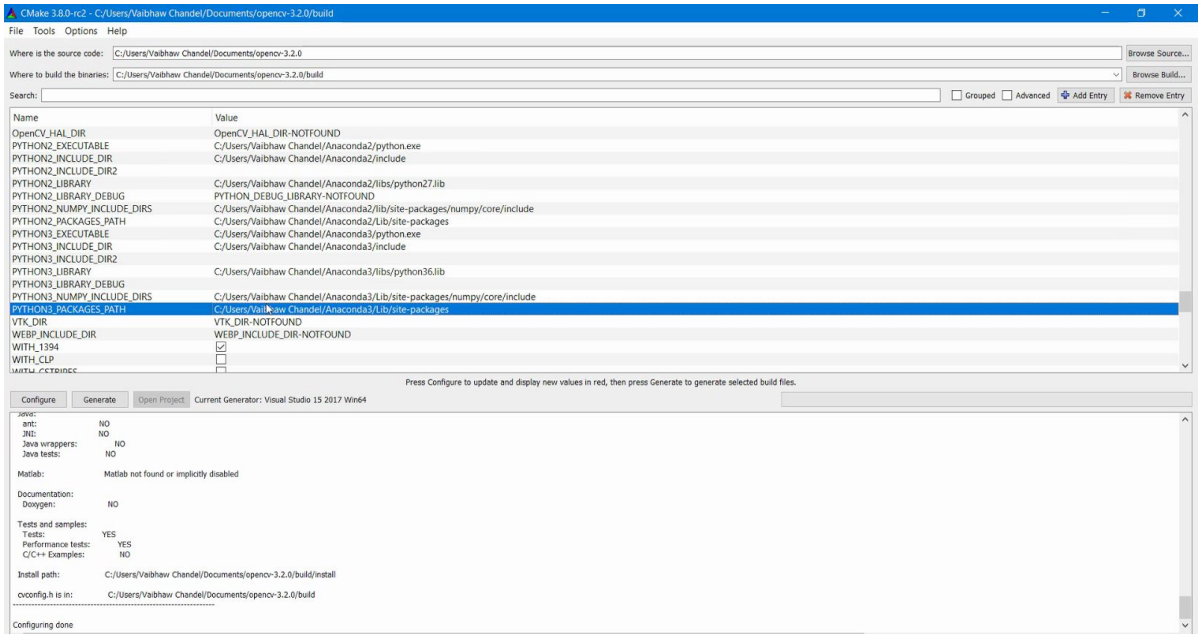
Step 5.2 : Add Python paths for both Python2 and Python3 (optional)

This section is only for people who want to **generate OpenCV's Python bindings for both Python2 and Python 3**. If you are going to use just one Python either 2 or 3, you should skip this section.

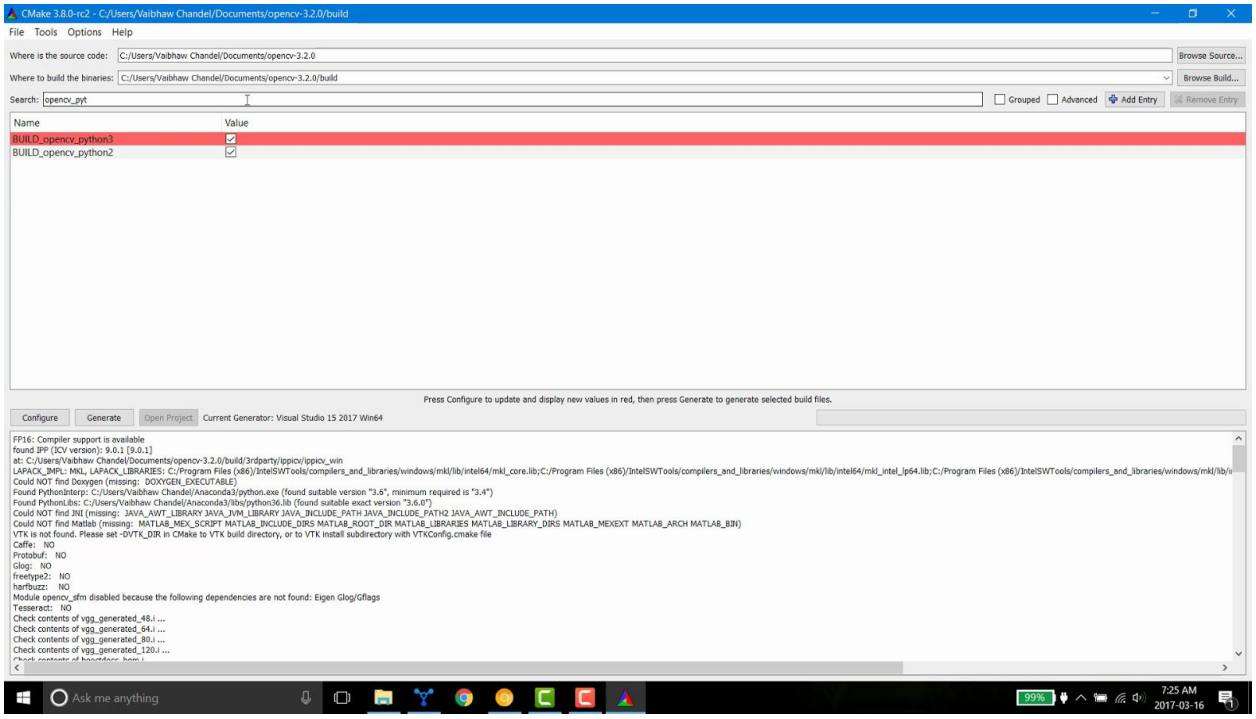
CMake was unable to find paths for my Python3 files.



So I manually added paths for Python3.



Now click configure again. After configuring is done, search **opencv_python** in search bar, both **BUILD_opencv_python2** and **BUILD_opencv_python3** will be automatically checked. Now we are sure that OpenCV binaries for both Python2 and Python 3 will be generated after compilation.



Step 5.3 : Generate the files

If CMake is able to configure without any errors it should say “Configuring done”.

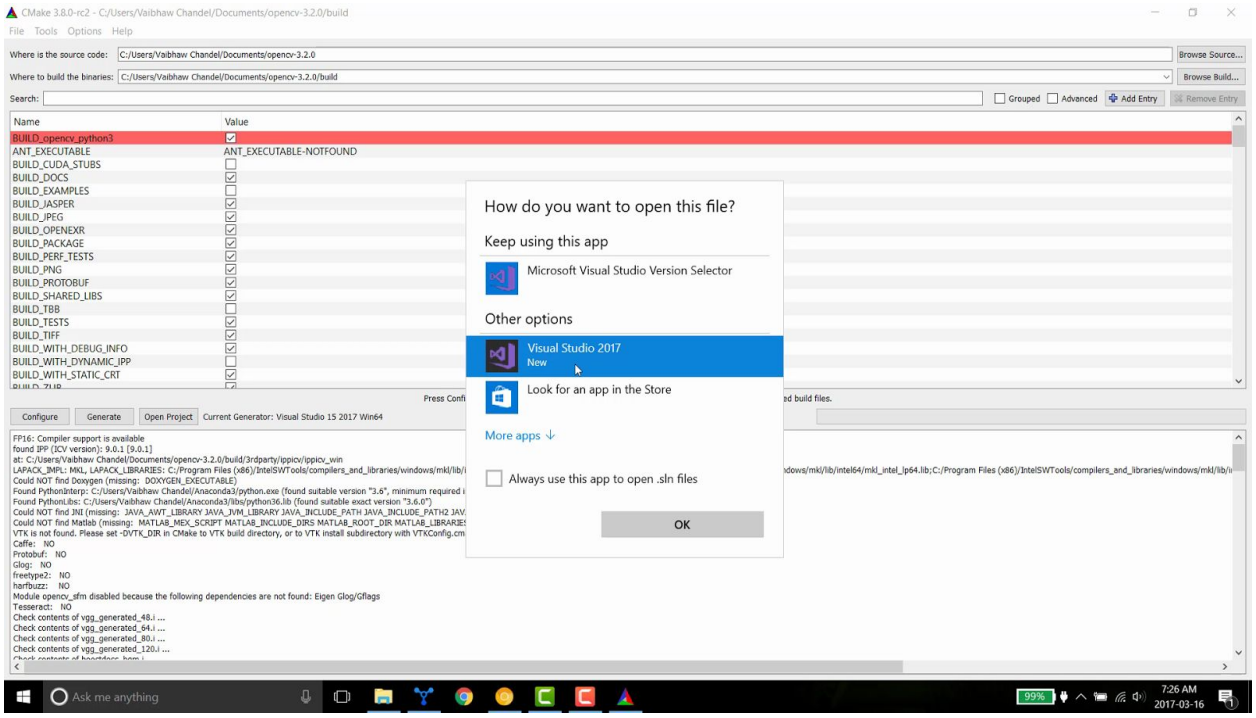
Click generate.

Note: Whenever you make any changes(check/uncheck boxes or change path) to configuration generated by CMake, always click configure and generate.

Step 6: Compile OpenCV

Step 6.1 : Open project in Visual Studio

Once CMake has generated the necessary files, click on Open Project. Choose Visual Studio 2015 if it asks for which app to use to open project.

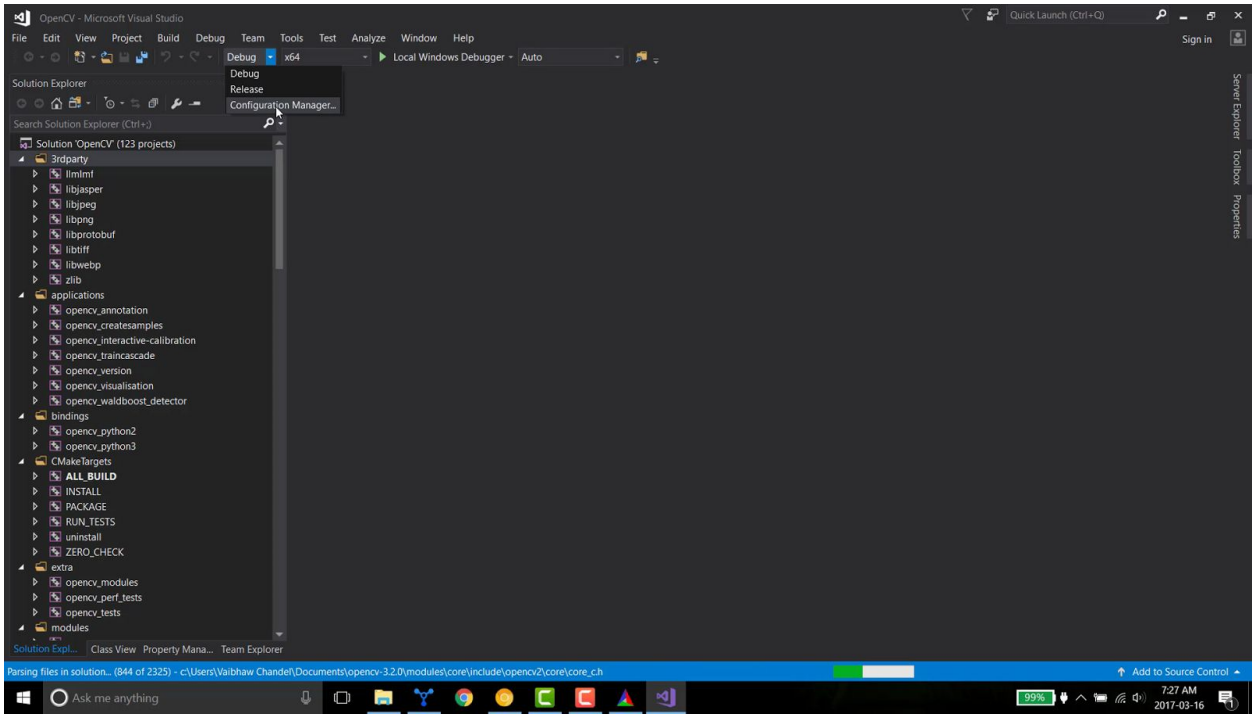


Or instead of clicking “Open Project” you can go to your build folder i.e. **OPENCV_PATH**/build and double click on .sln file. This will open Visual Studio.

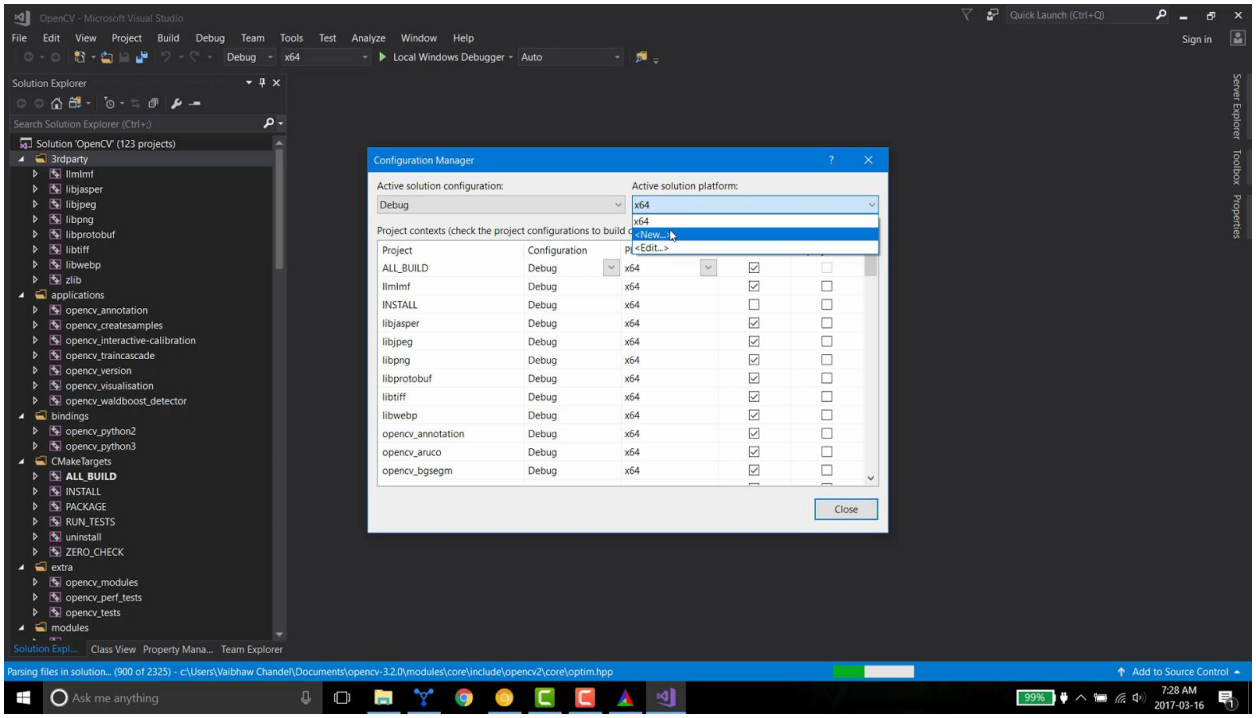
In Visual Studio, you can build a project in different configuration settings. We will build OpenCV with two configurations: **Debug mode - x64 platform** and **Release mode - x64 platform**.

Step 6.2 : if x64 or Release mode not present in drop-down menu

If x64 is not present click on Debug and select Configuration Manager in dropdown menu.



Now click x86 and select New from dropdown menu



The new window that appears, select x64 as “new platform” and in “copy settings from” keep x86. Click Ok.

Similarly if the Release mode is not present, click Configuration Manager -> Debug -> New

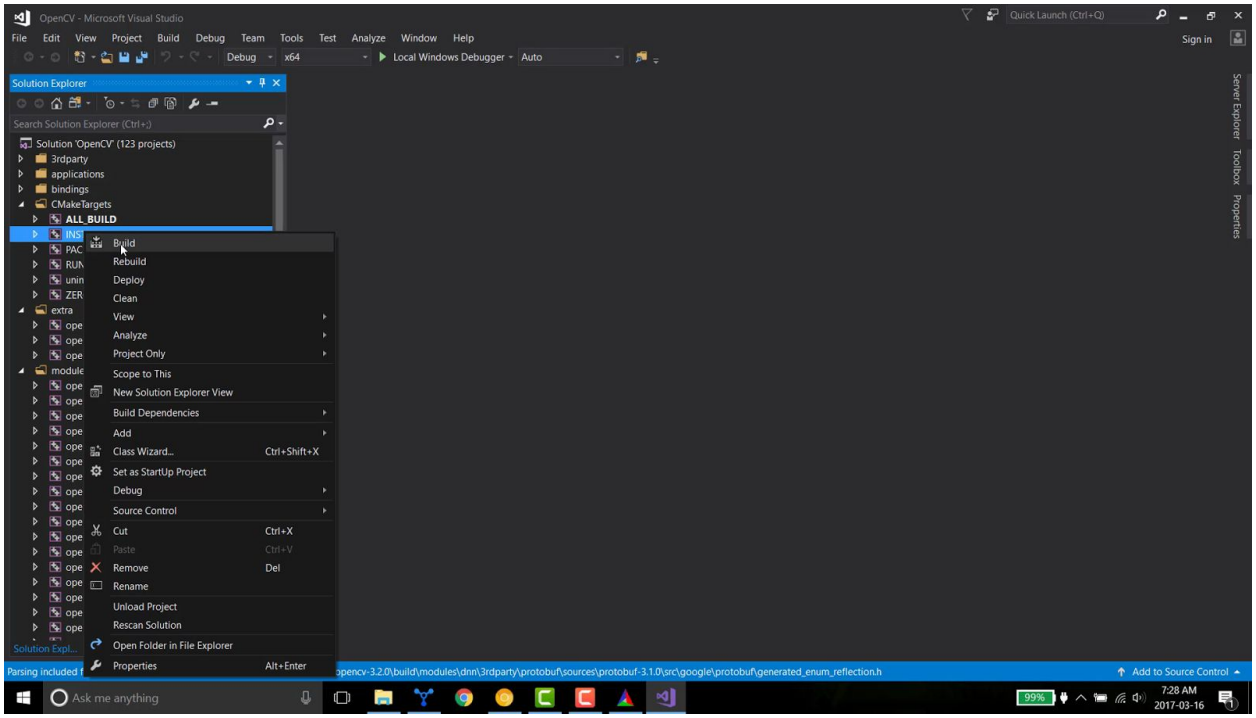
Type Release in “Name” and Debug in “copy settings from”.

Now we are done with creating Build Configurations in Visual Studio.

Step 6.3 : Install opencv in debug mode

Select mode as Debug and target as x64 from the drop down menu.

In Solution Explorer (a panel situated in left) under CMakeTargets find INSTALL, right-click select Build.

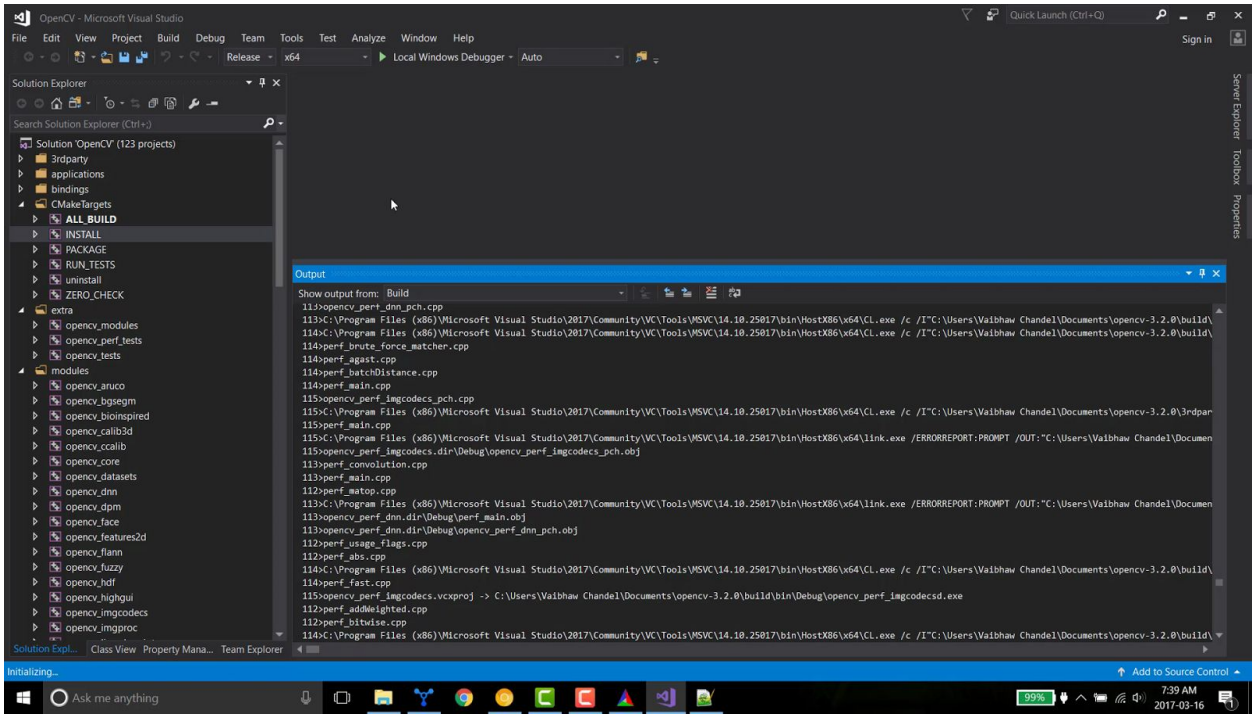


It may give an error for generating Python binary. You can ignore that because we will use Python binaries generated from Release mode.

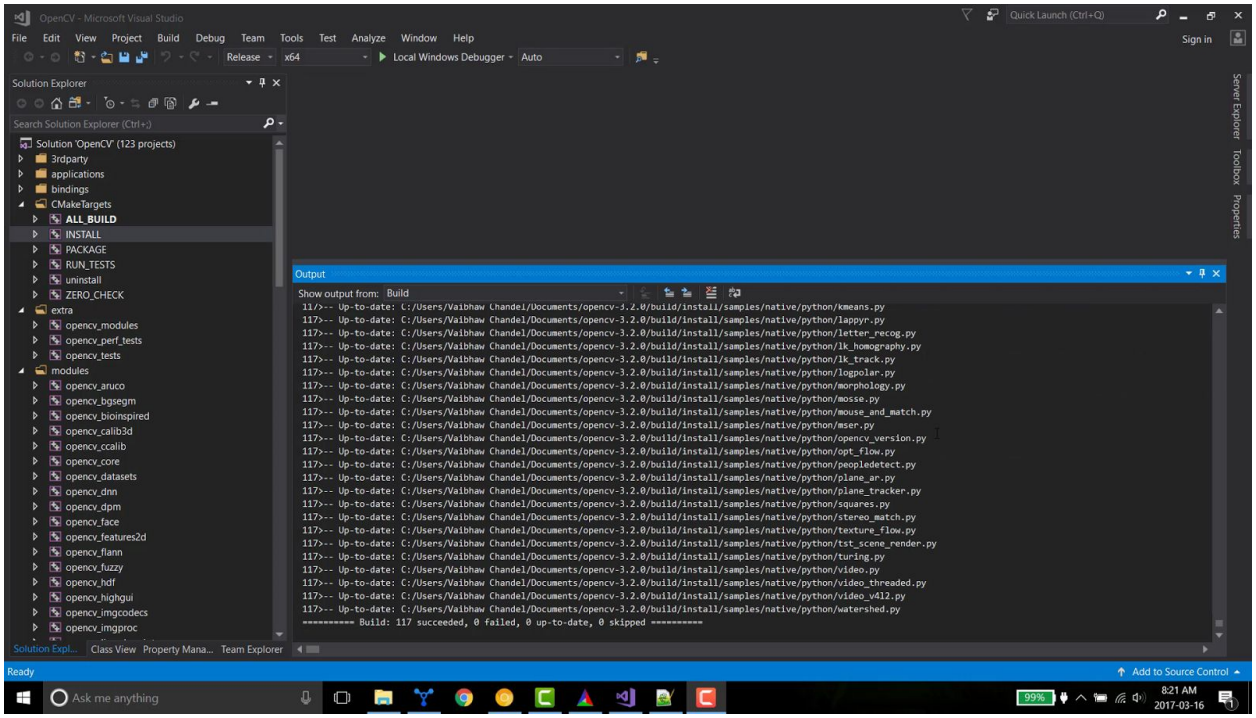
Step 6.4 : Install opencv in Release mode

Select Mode as Release and target as x64.

Again right click on “INSTALL” under CMakeTargets and select Build.



Once the build is completed check the log. It should look something like this:



Check - Build: 117 succeeded, 0 failed, 0 skipped

Now that we have compiled OpenCV we will find out how to configure a Visual Studio project to use OpenCV library.

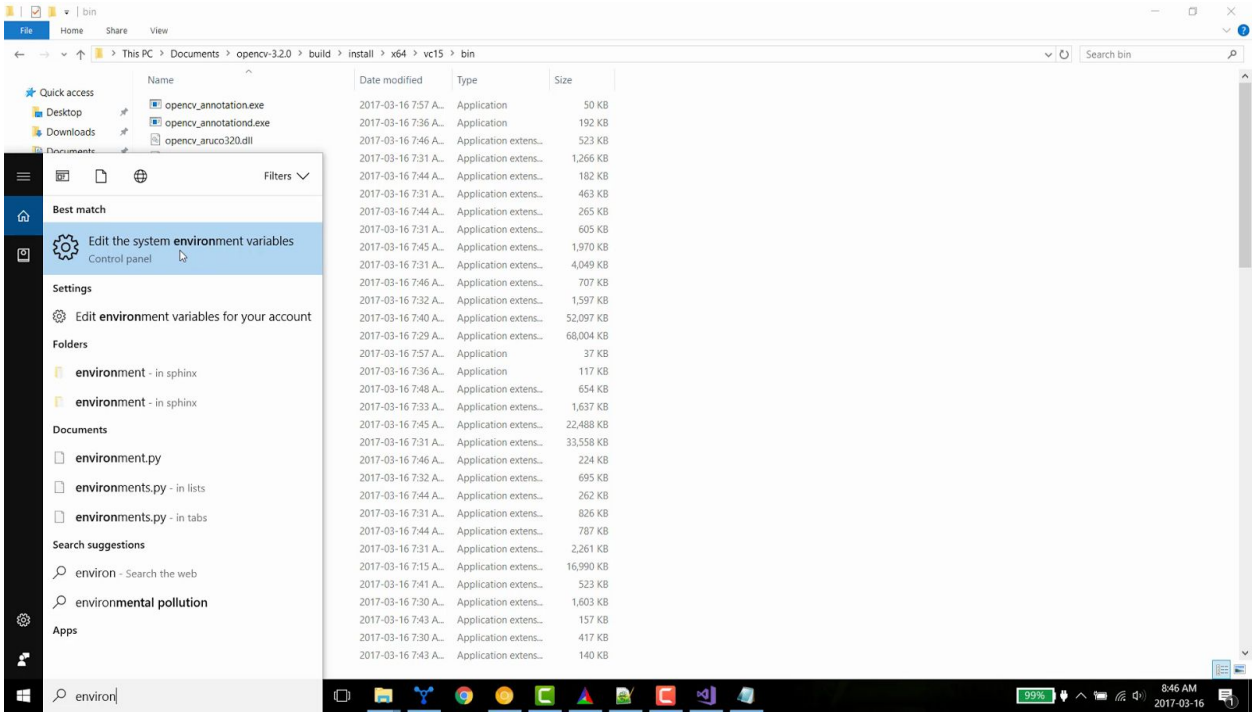
We will use an example from Learnopencv's github repository.
https://github.com/spmallick/learnopencv/tree/master/RedEyeRemover

You can also download the zip file redEyeRemover and extract it into a folder.

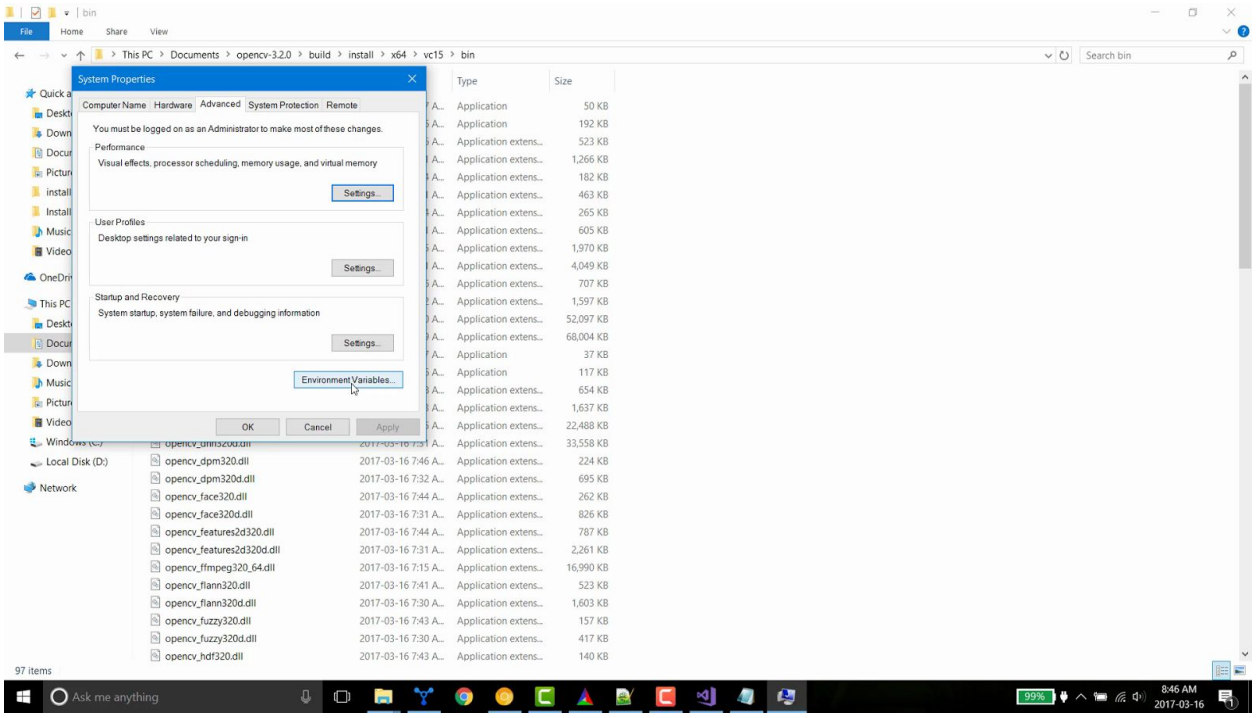
Step 6.5 : Update environment variables

First of all we will add OpenCV dll files' path to our system PATH.

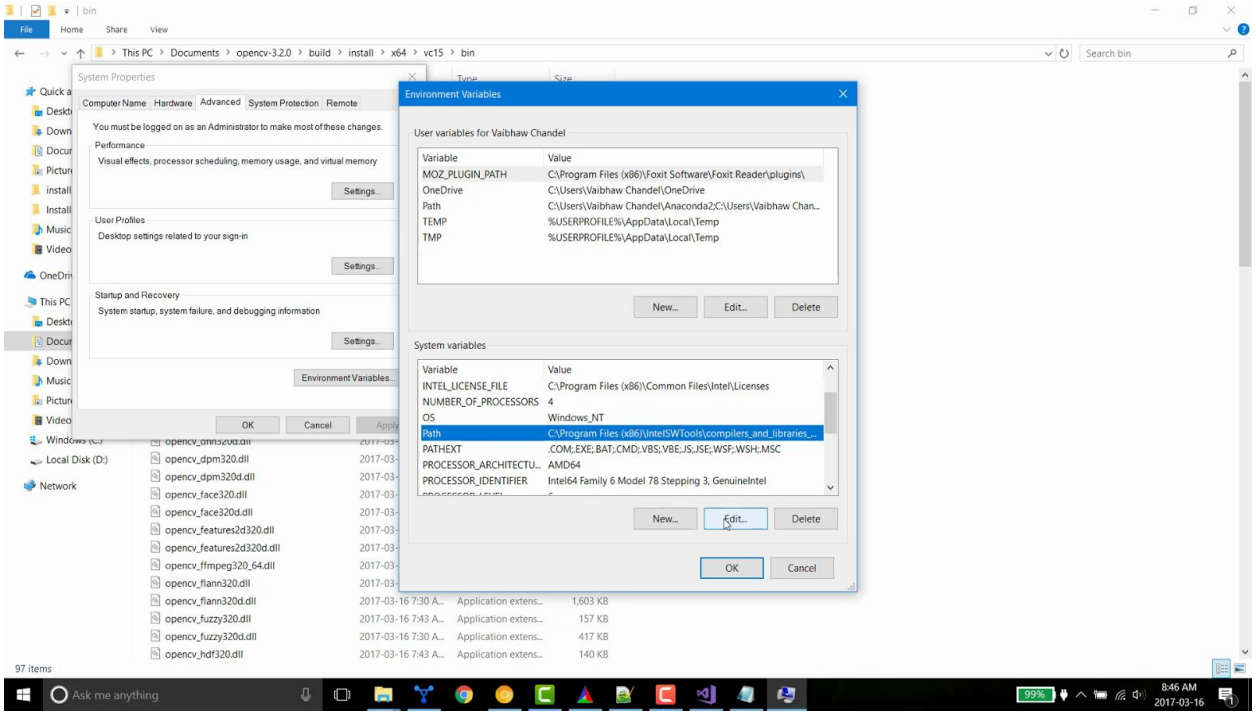
Press Windows Super key, search for "environment variables"



Click Environment Variables in System Properties window

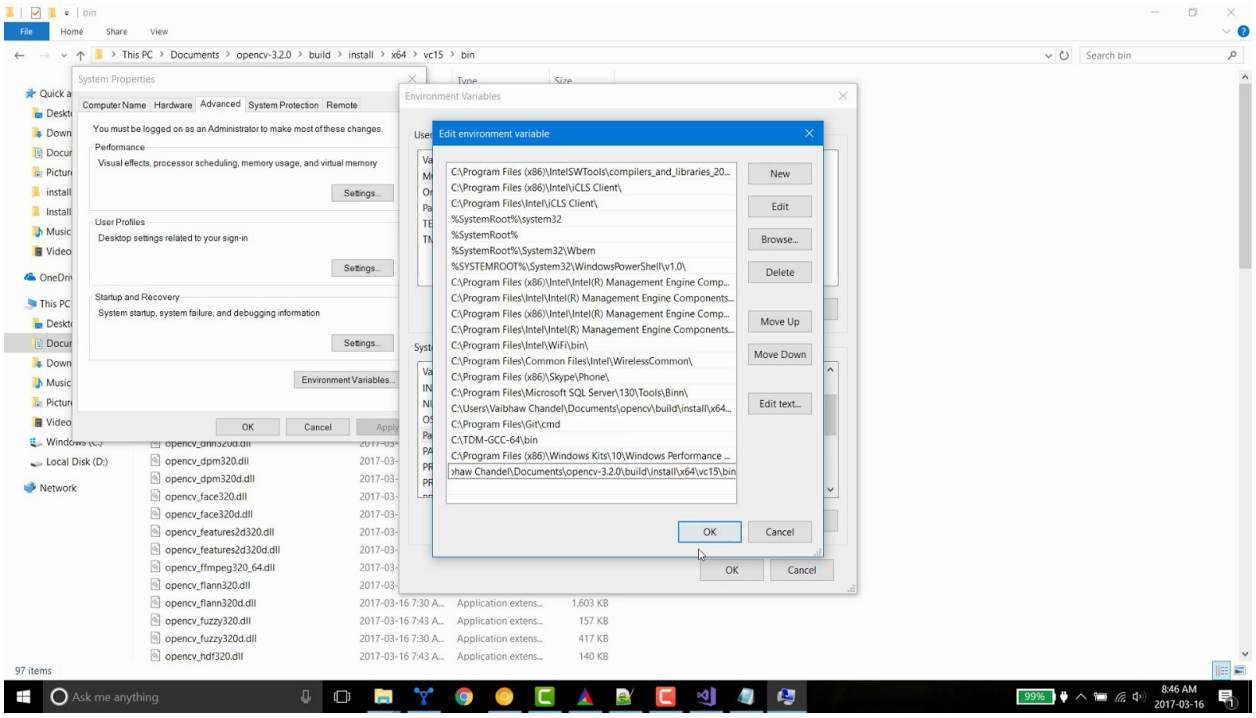


Under System Variables, Select Path and click edit



Click New, and give path to **OPENCV_PATH**\build\install\x64\vc15\bin and click Ok. Depending upon where you have kept opencv-3.2.0 folder and what version of Visual Studio you used to compile OpenCV, this path would be different. In my case full path is:

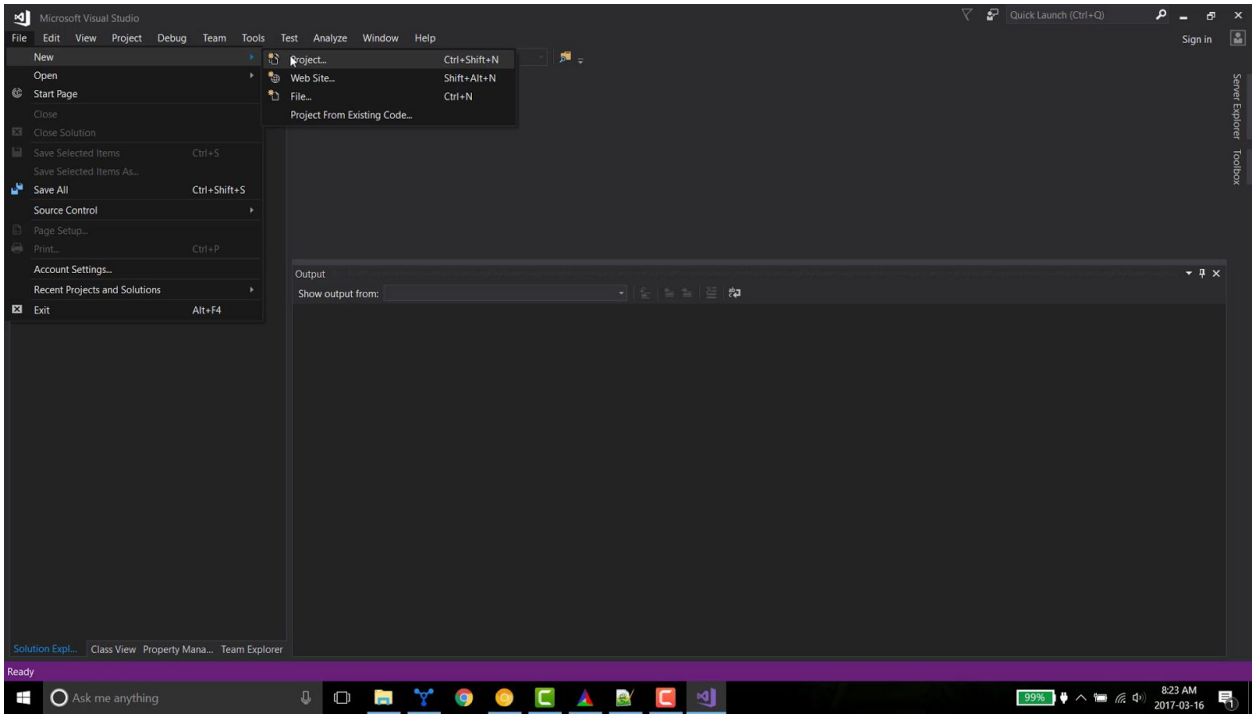
C:\Users\Vaibhaw Chandel\Documents\opencv-3.2.0\build\install\x64\vc15\bin



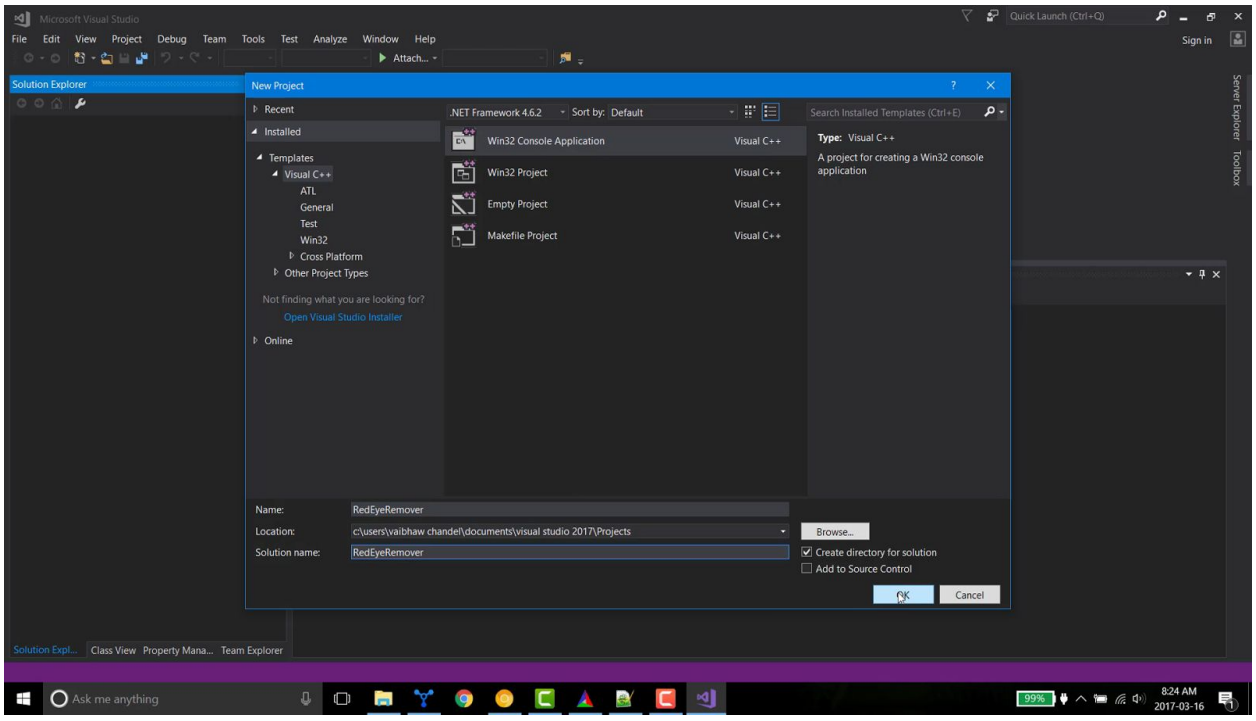
Step 7: Testing C++ code

Step 7.1 : Create New Project

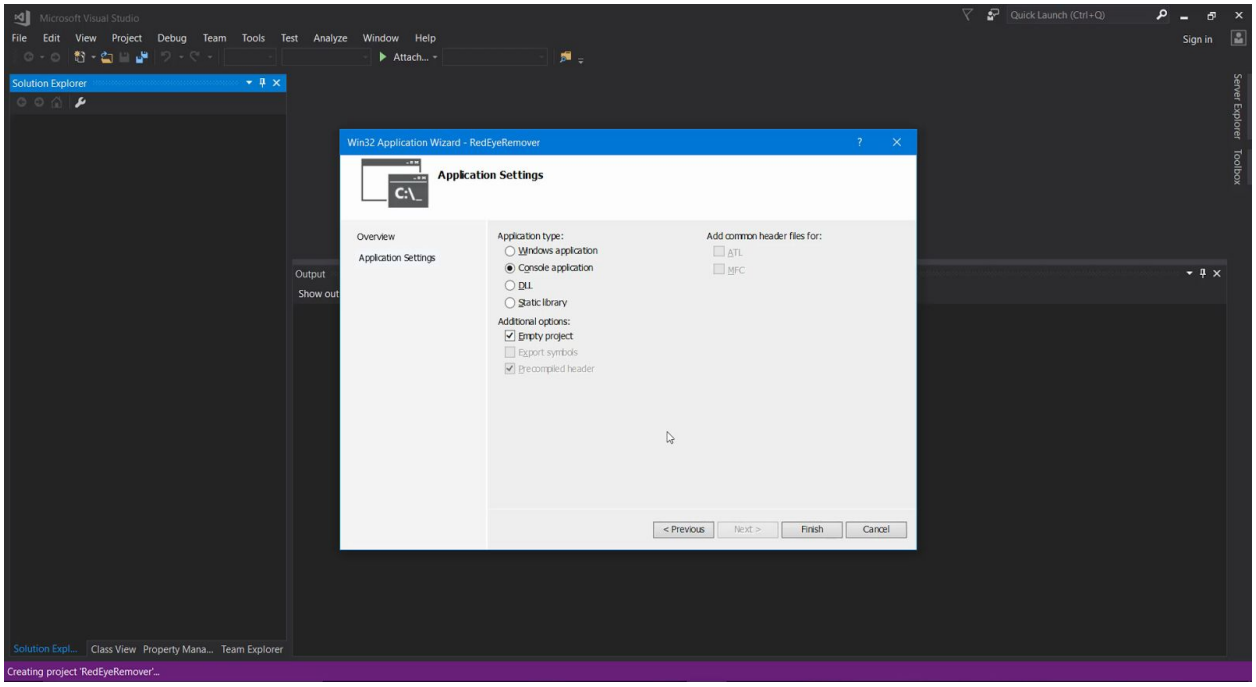
Now we will create a new project in Visual Studio



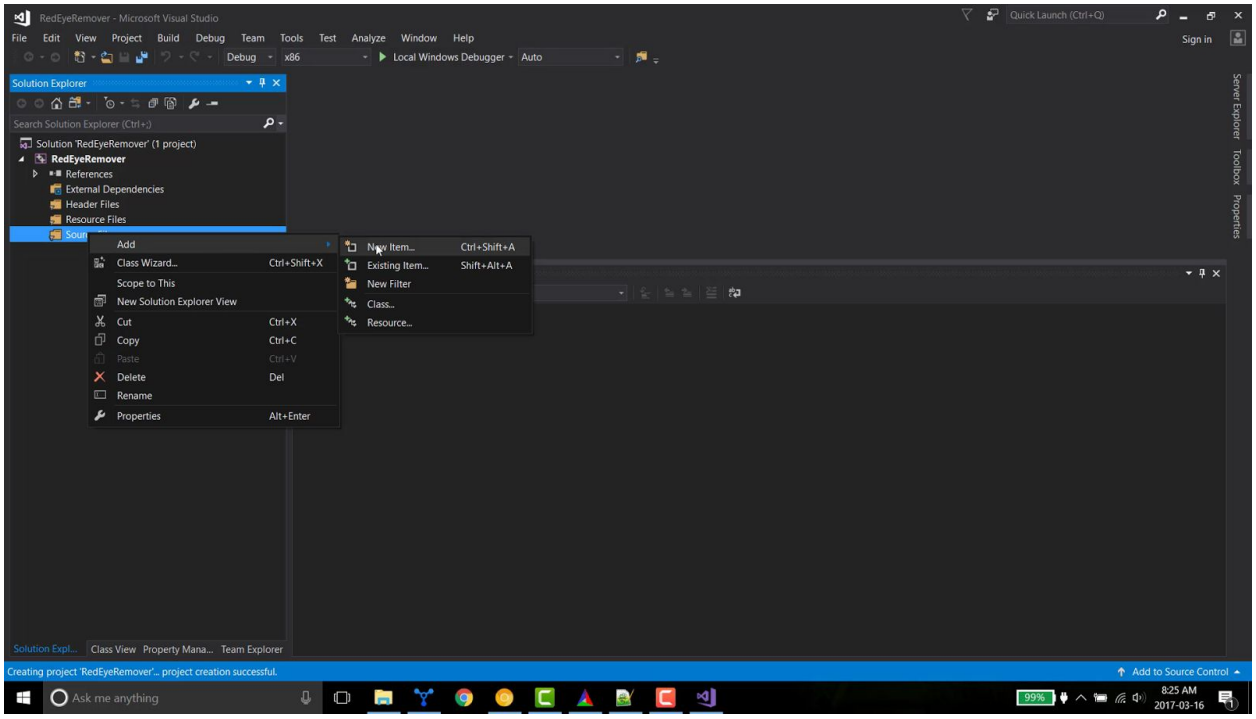
In next window, within Templates select Visual C++ then “Win32 Console Application”, give Project Name and Solution Name, then click OK. I gave “RedEyeRemover” in both Project Name and Solution Name.



In next window, select Console Application and check “Empty Project”, click Finish.



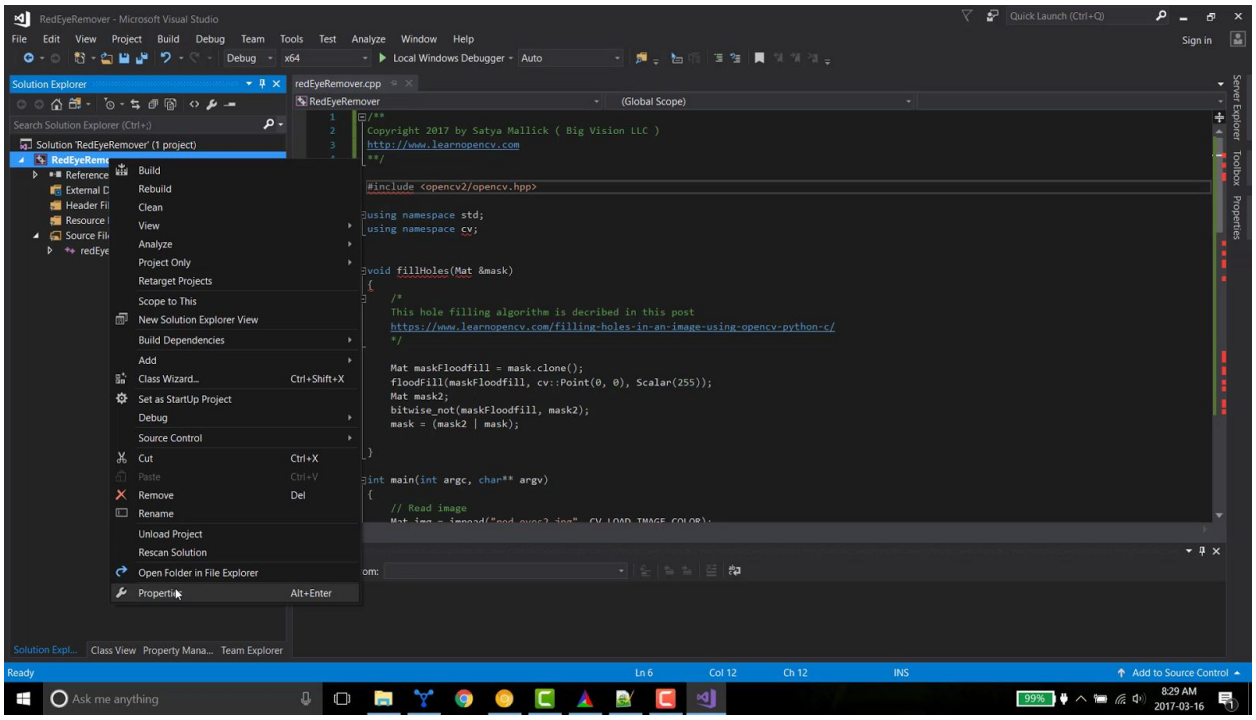
Now we will add a new File with Source Files. Right-click source files, click New Item.



In next window, select C++ file and give name to this file. I have given name redEyeRemover.cpp

This will create an empty C++ file. Copy code from **removeRedEyes.cpp** to this file.

NOTE : Before proceeding, make sure that the configurations in the drop down menu are Debug and x64, because when we create a new project, it might open in Debug and x86 by default.



As you can see Visual Studio is showing a lot of errors (definition not found errors). This is because Visual Studio doesn't know where OpenCV's header files are kept.

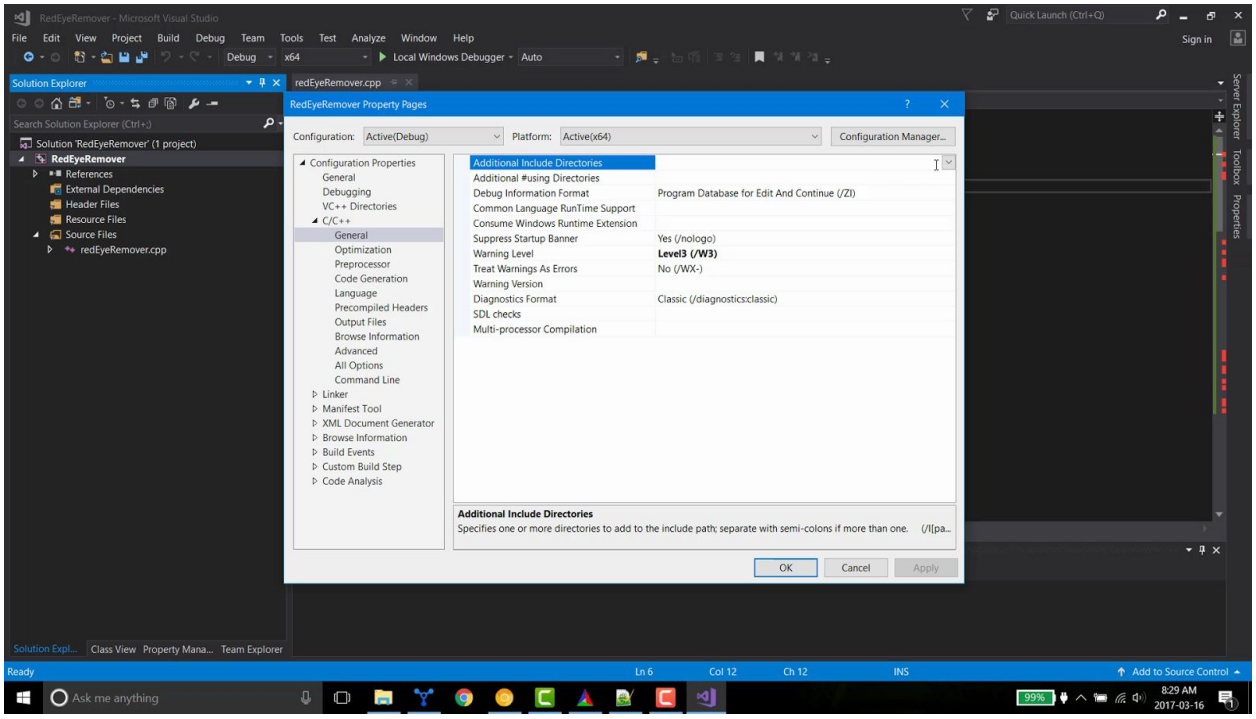
Step 7.2 : Specify OpenCV's include directories

We need to add paths of OpenCV's header files so that we can compile our project and path of OpenCV's library files so that linker can access library files.

Right click on your project and select Properties.

NOTE : Again, before proceeding, make sure that the configurations in the drop down menu of the “Configuration properties Window” are Debug and x64.

Under “Configuration Properties” -> C/C++ -> General, click on “Additional Include Directories”.



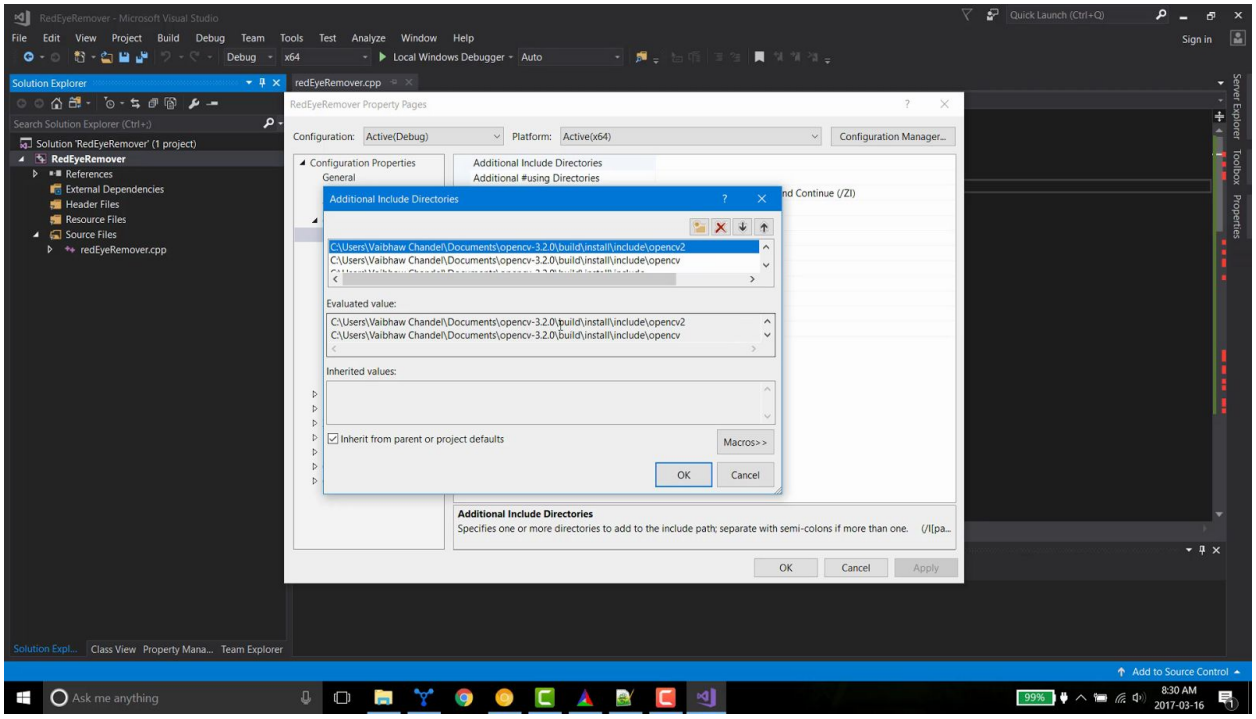
You have to add 3 paths here.

OPENCV_PATH\build\install\include
OPENCV_PATH\build\install\include\opencv
OPENCV_PATH\build\install\include\opencv2

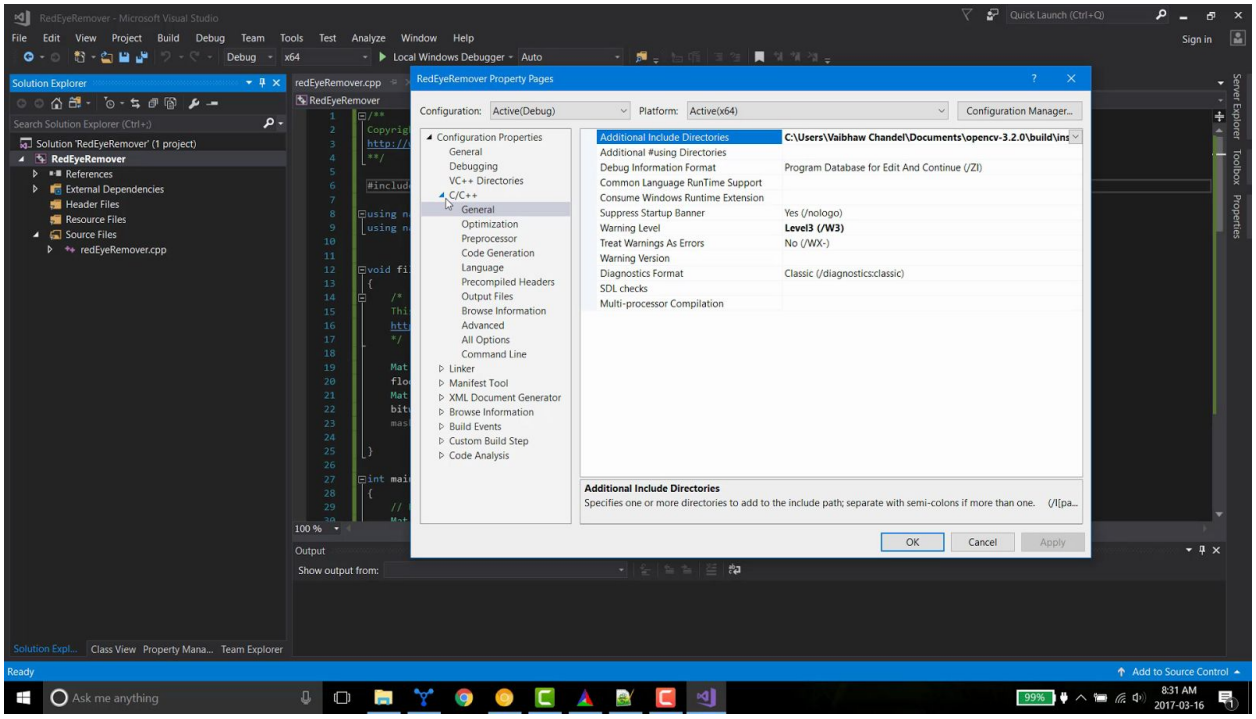
In my case paths are:

C:\Users\Vaibhaw Chandel\Documents\opencv-3.2.0\build\install\include
C:\Users\Vaibhaw Chandel\Documents\opencv-3.2.0\build\install\include\opencv
C:\Users\Vaibhaw Chandel\Documents\opencv-3.2.0\build\install\include\opencv2

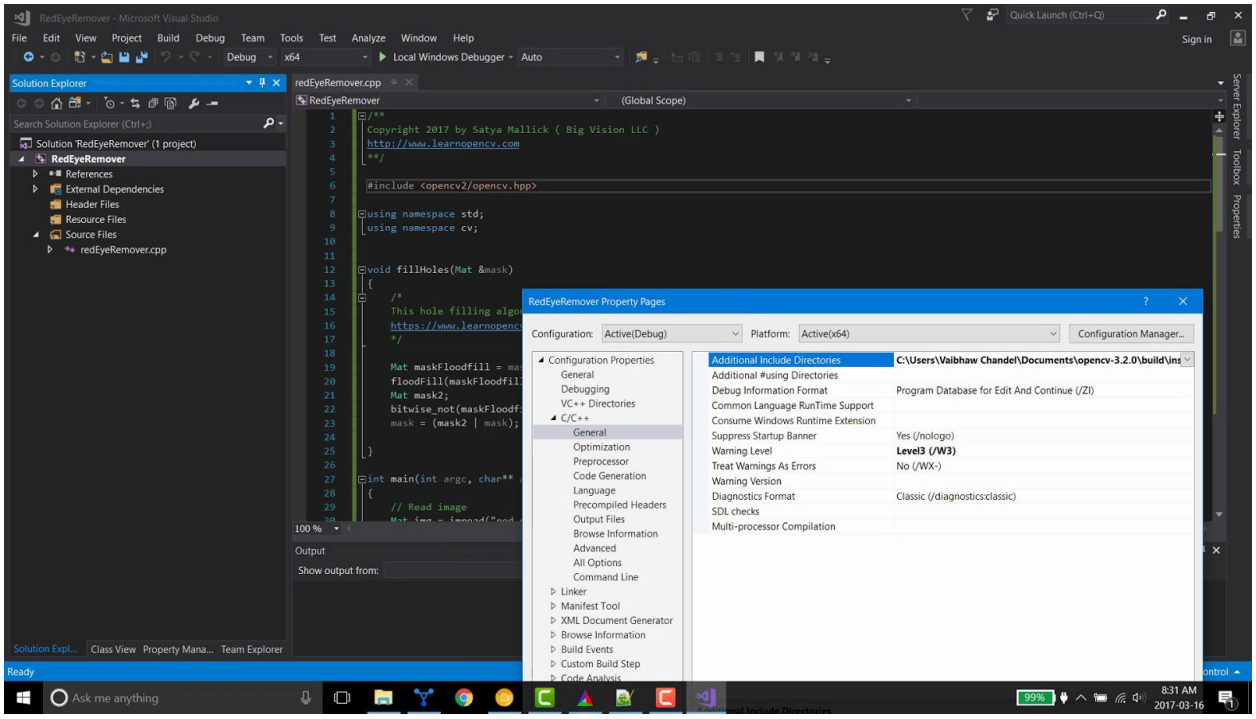
You can add a path by clicking on Folder icon left to Red Cross Icon.



Click OK.

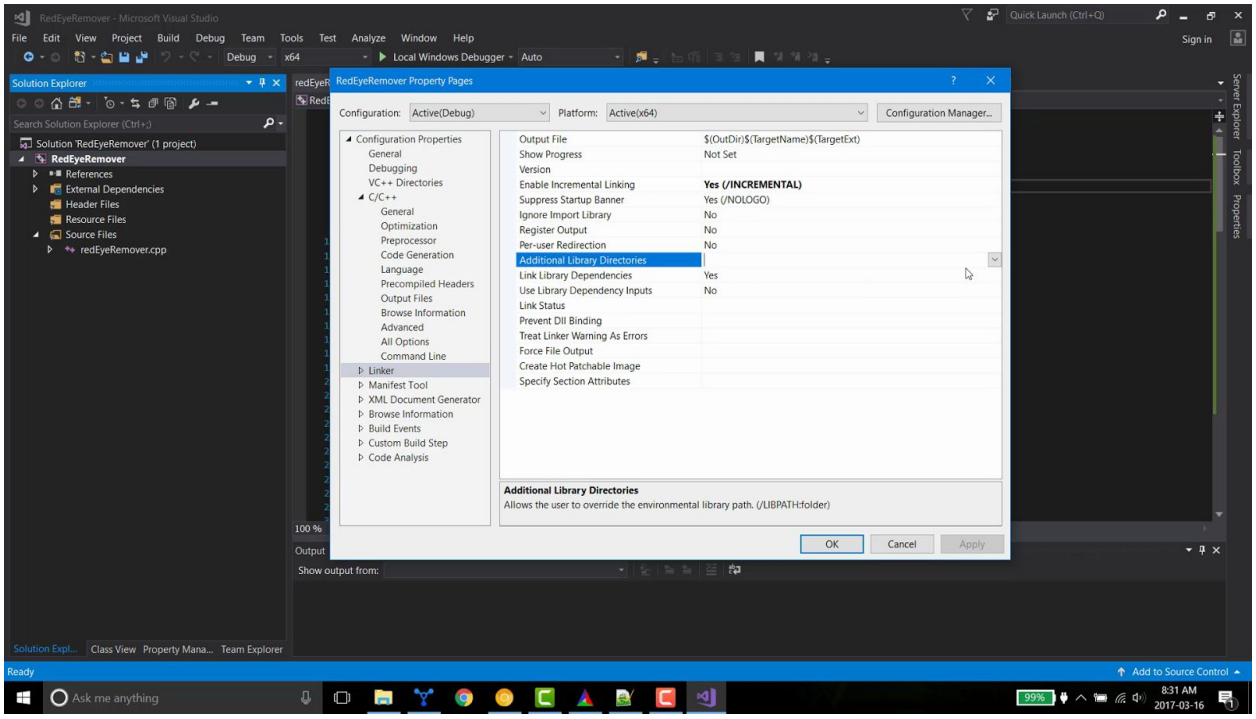


Now you will see, definition not found errors are gone in Visual Studio.

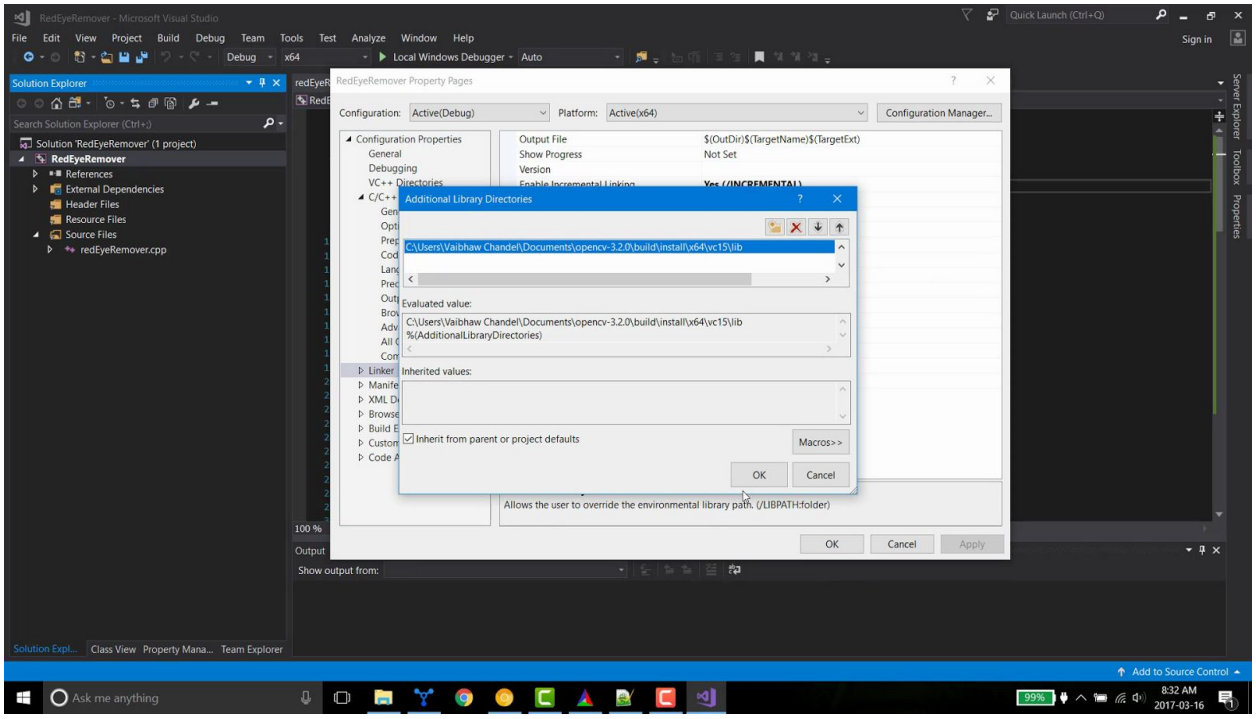


Step 7.3 : Specify OpenCV's library directory

Now we will add path to library directories. Click on Linker in left panel, then **Additional Library Directories**.

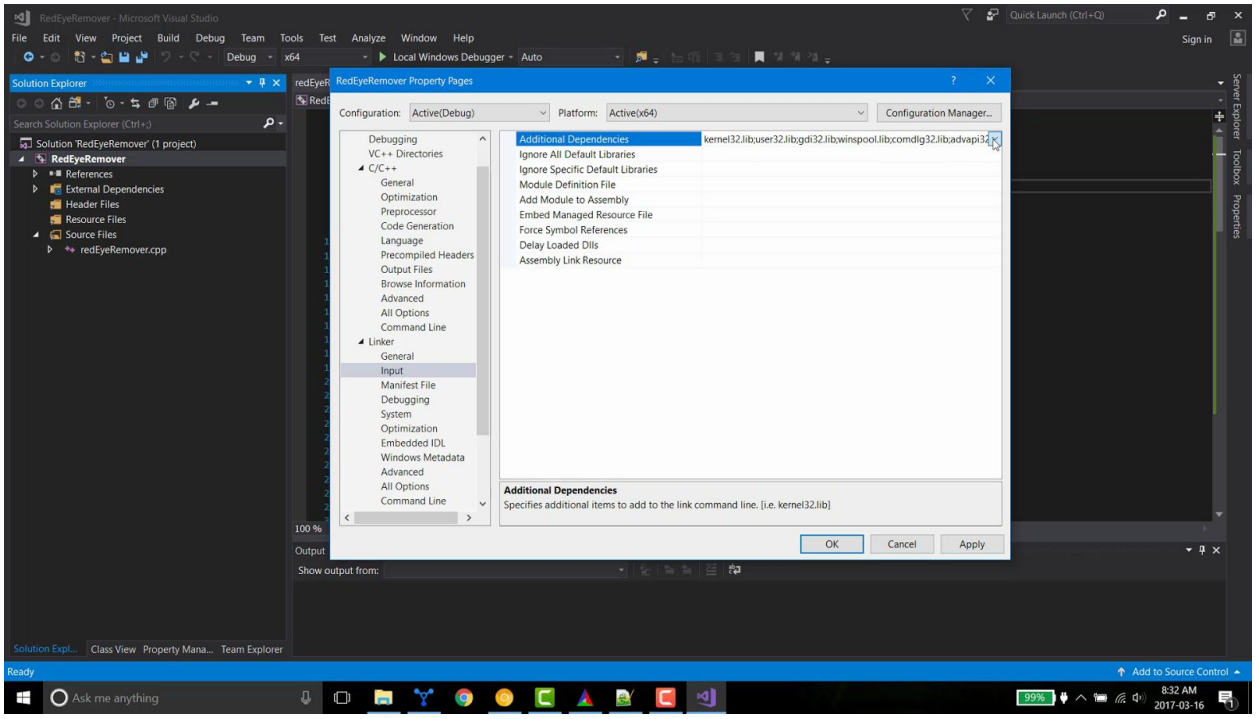


You have to add path **OPENCV_PATH**\build\install\x64\vc15\lib here.

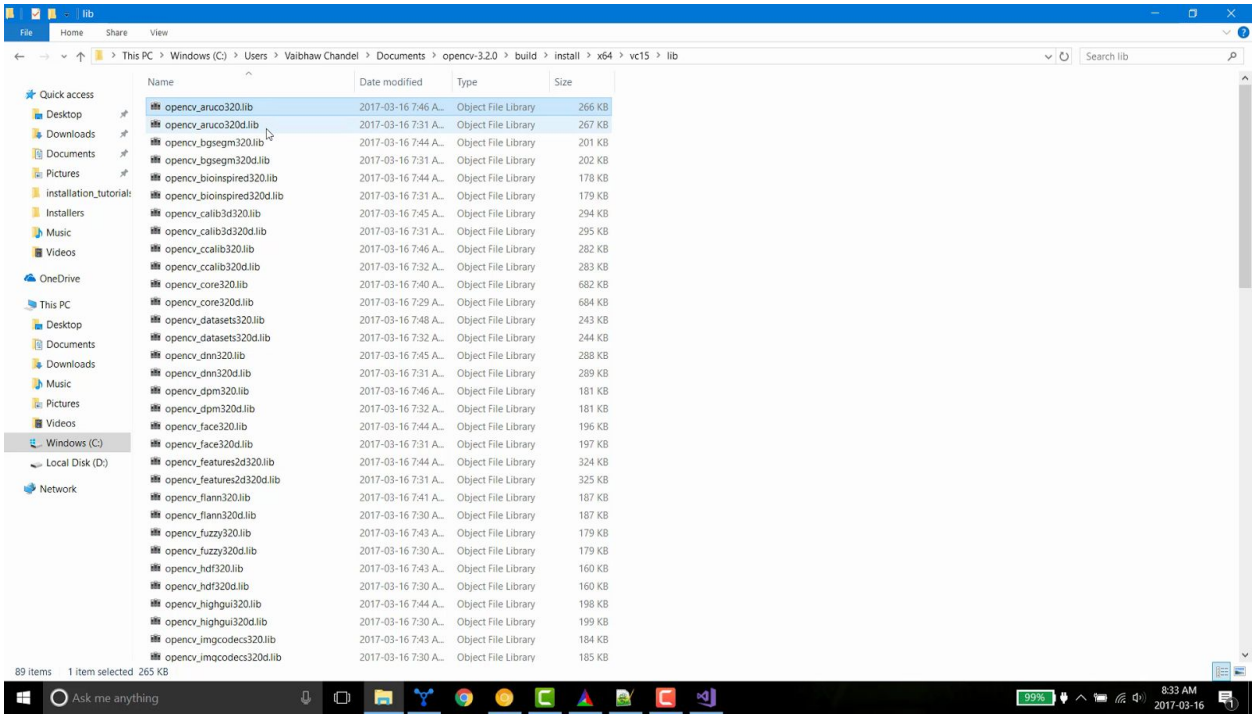


Step 7.4 : Specify OpenCV's libraries

Now within Linker click on Input. Here we will mention which library files we are going to use. Now in field "Additional Dependencies" we will add the names of all OpenCV library files.



We compiled OpenCV in both Debug mode and Release mode. So there are 2 versions of each library file.



In the screenshot here, you can see the first two files are named as

opencv_aruco320.lib and opencv_aruco320d.lib. First file was created in Release mode and second in Debug mode. d in second file denotes debug.

The important point to note here is that if you selected Configuration as “Debug” you should add debug library files in “Additional Dependencies” and if you selected Configuration as “Release” you should add release library files here. 320 stands for version 3.2.0.

I have created a list of all OpenCV library files.

Additional Dependencies: DEBUG mode

```
opencv_aruco320d.lib
opencv_bgsegm320d.lib
opencv_bioinspired320d.lib
opencv_calib3d320d.lib
opencv_ccalib320d.lib
opencv_core320d.lib
opencv_datasets320d.lib
opencv_dnn320d.lib
opencv_dpm320d.lib
opencv_face320d.lib
opencv_features2d320d.lib
```

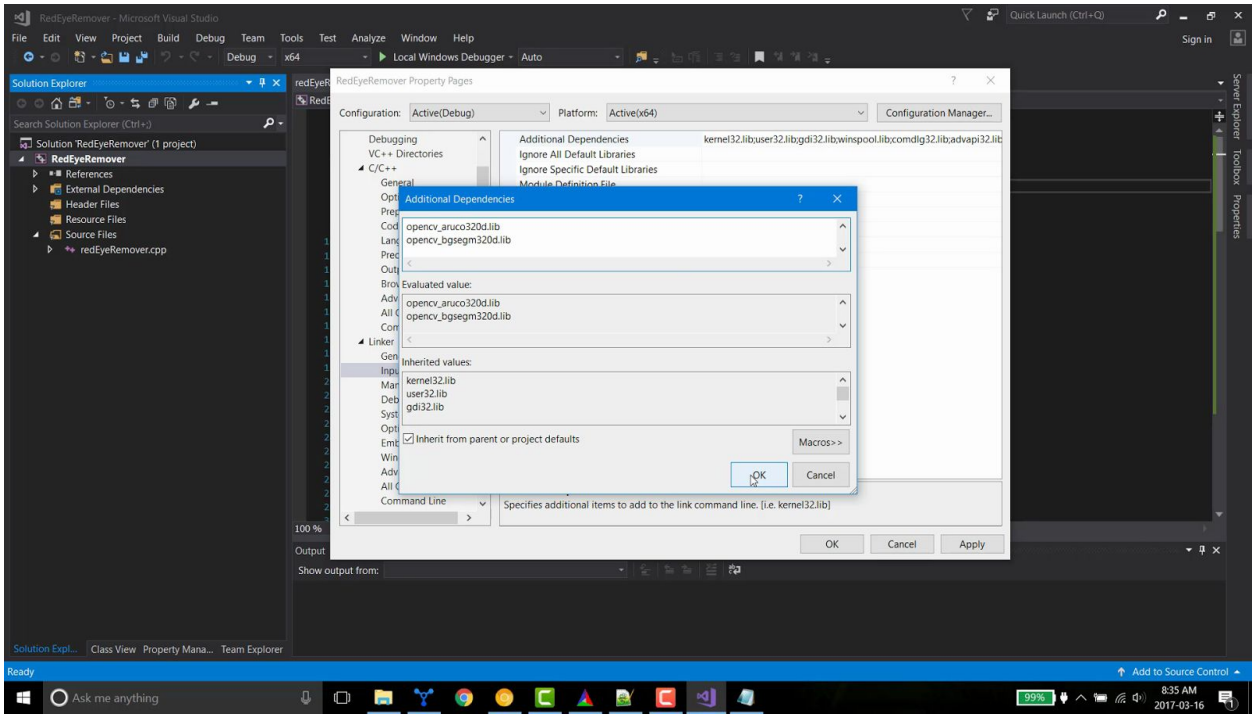
```
opencv_flann320d.lib
opencv_fuzzy320d.lib
opencv_hdf320d.lib
opencv_highgui320d.lib
opencv_imgcodecs320d.lib
opencv_imgproc320d.lib
opencv_line_descriptor320d.lib
opencv_ml320d.lib
opencv_objdetect320d.lib
opencv_optflow320d.lib
opencv_phase_unwrapping320d.lib
opencv_photo320d.lib
opencv_plot320d.lib
opencv_reg320d.lib
opencv_rgbd320d.lib
opencv_saliency320d.lib
opencv_shape320d.lib
opencv_stereo320d.lib
opencv_stitching320d.lib
opencv_structured_light320d.lib
opencv_superres320d.lib
opencv_surface_matching320d.lib
opencv_text320d.lib
opencv_tracking320d.lib
opencv_video320d.lib
opencv_videoio320d.lib
opencv_videostab320d.lib
opencv_xfeatures2d320d.lib
opencv_ximgproc320d.lib
opencv_xobjdetect320d.lib
opencv_xphoto320d.lib
```

Copy these Debug mode library names and add in “Additional Dependencies” box.

Later when you want to build in Release mode specify filenames given in following table.

Additional Dependencies: RELEASE mode

opencv_aruco320.lib
opencv_bgsegm320.lib
opencv_bioinspired320.lib
opencv_calib3d320.lib
opencv_ccalib320.lib
opencv_core320.lib
opencv_datasets320.lib
opencv_dnn320.lib
opencv_dpm320.lib
opencv_face320.lib
opencv_features2d320.lib
opencv_flann320.lib
opencv_fuzzy320.lib
opencv_hdf320.lib
opencv_highgui320.lib
opencv_imgcodecs320.lib
opencv_imgproc320.lib
opencv_line_descriptor320.lib
opencv_ml320.lib
opencv_objdetect320.lib
opencv_optflow320.lib
opencv_phase_unwrapping320.lib
opencv_photo320.lib
opencv_plot320.lib
opencv_reg320.lib
opencv_rgbd320.lib
opencv_saliency320.lib
opencv_shape320.lib
opencv_stereo320.lib
opencv_stitching320.lib
opencv_structured_light320.lib
opencv_superres320.lib
opencv_surface_matching320.lib
opencv_text320.lib
opencv_tracking320.lib
opencv_video320.lib
opencv_videoio320.lib
opencv_videostab320.lib
opencv_xfeatures2d320.lib
opencv_ximgproc320.lib
opencv_xobjdetect320.lib
opencv_xphoto320.lib



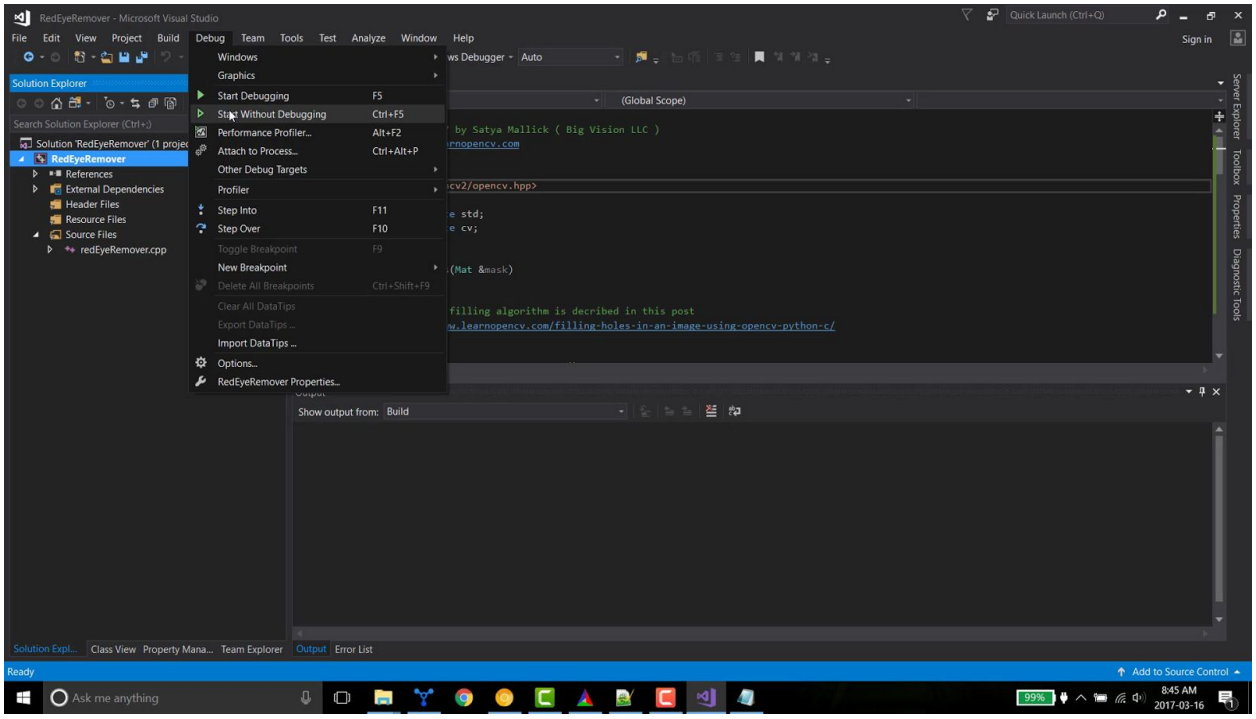
Another Important Point: Since we have built OpenCV for x64 platform we will always use platform “x64” in our projects.

Step 7.5 : Copy project files

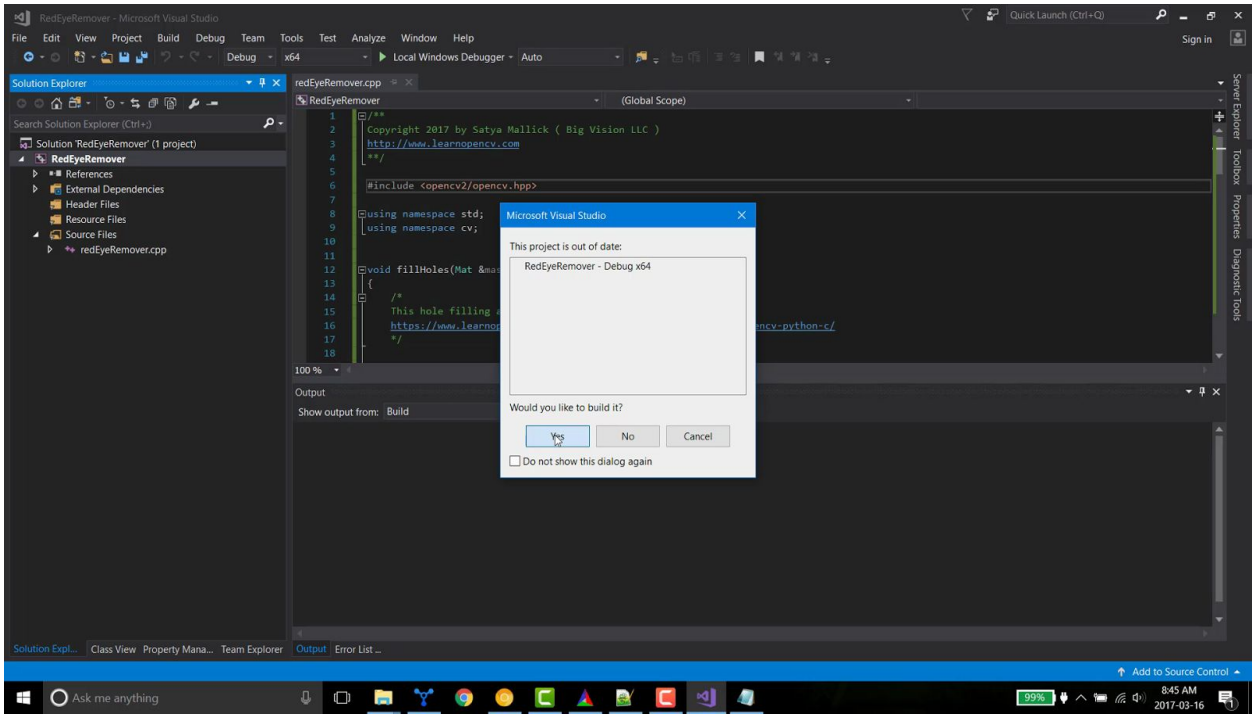
Now right-click project (RedEyeRemover), click “Open Folder In File Explorer”. Copy files red_eyes.jpg, red_eyes2.jpg and haarcascade_eye.xml from extracted folder(RedEyeRemover.zip) to this folder (Visual Studio project). All these files should be next to redEyeRemover.cpp file that we created earlier.

Step 7.6 : Build project

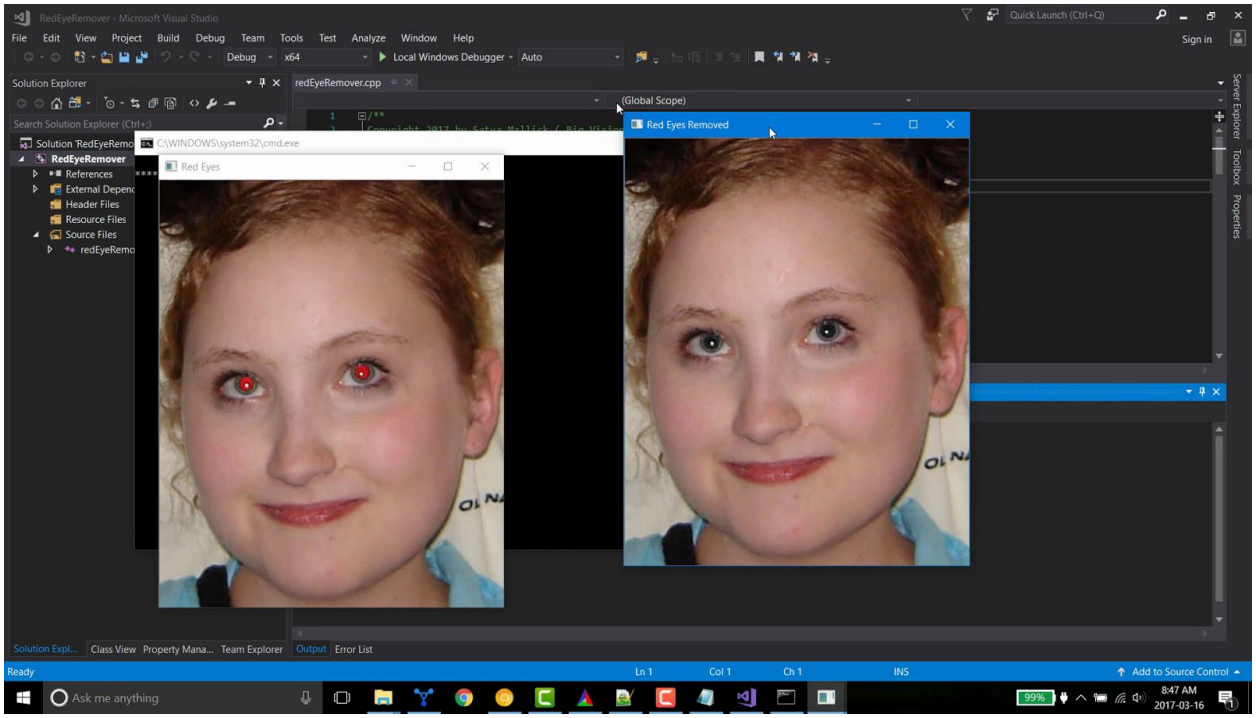
We are now all set to build our application. Click debug, select “Start Without Debugging”.



When it asks to build the project select “Yes”.



If the application is built correctly, you will see two image display windows. One with red-eyes and another without red-eyes.



Step 8: Testing Python code

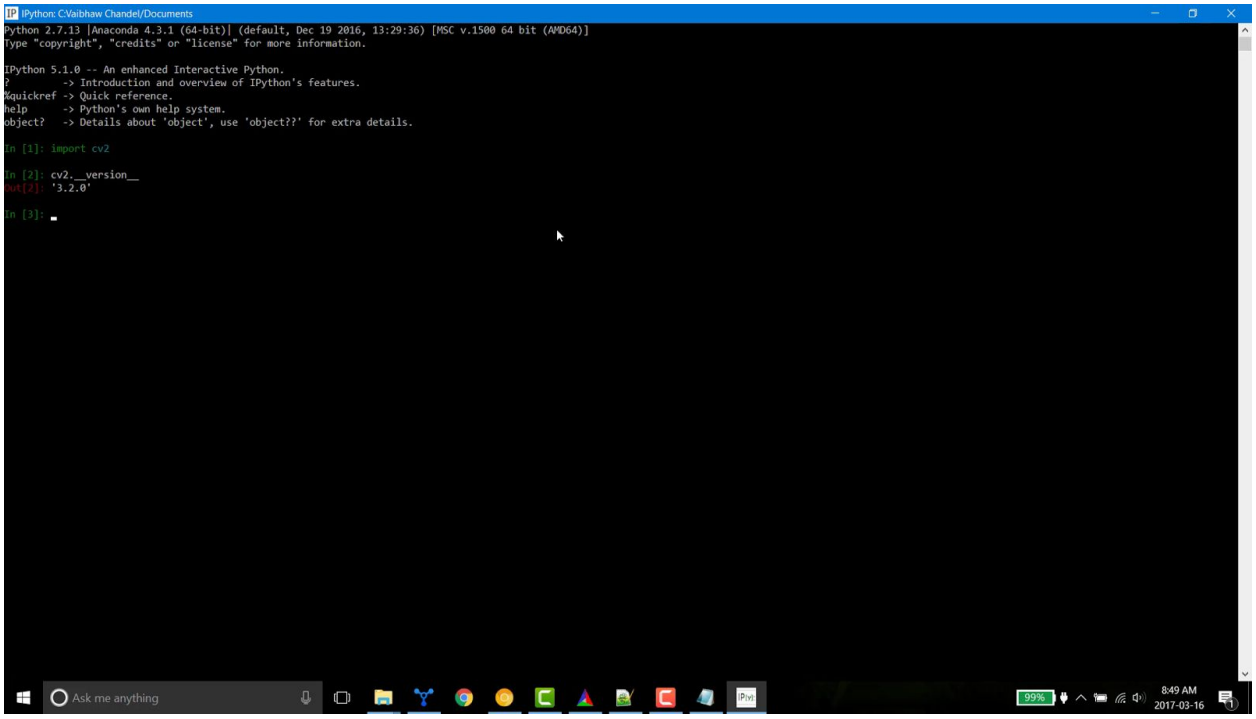
Step 8.1 : Quick check

Quick way to check whether OpenCV for Python is installed correctly or not is to import cv2 in python interpreter.

Open command prompt in Windows, run python command. This will open Python interpreter. Run these two commands

```
import cv2
print cv2.__version__
```

Anaconda comes with a feature-rich Python interpreter called IPython. I tested these commands in IPython.



If OpenCV for Python is installed correctly, running command “import cv2” will give no errors. If any error comes up it means installation failed.

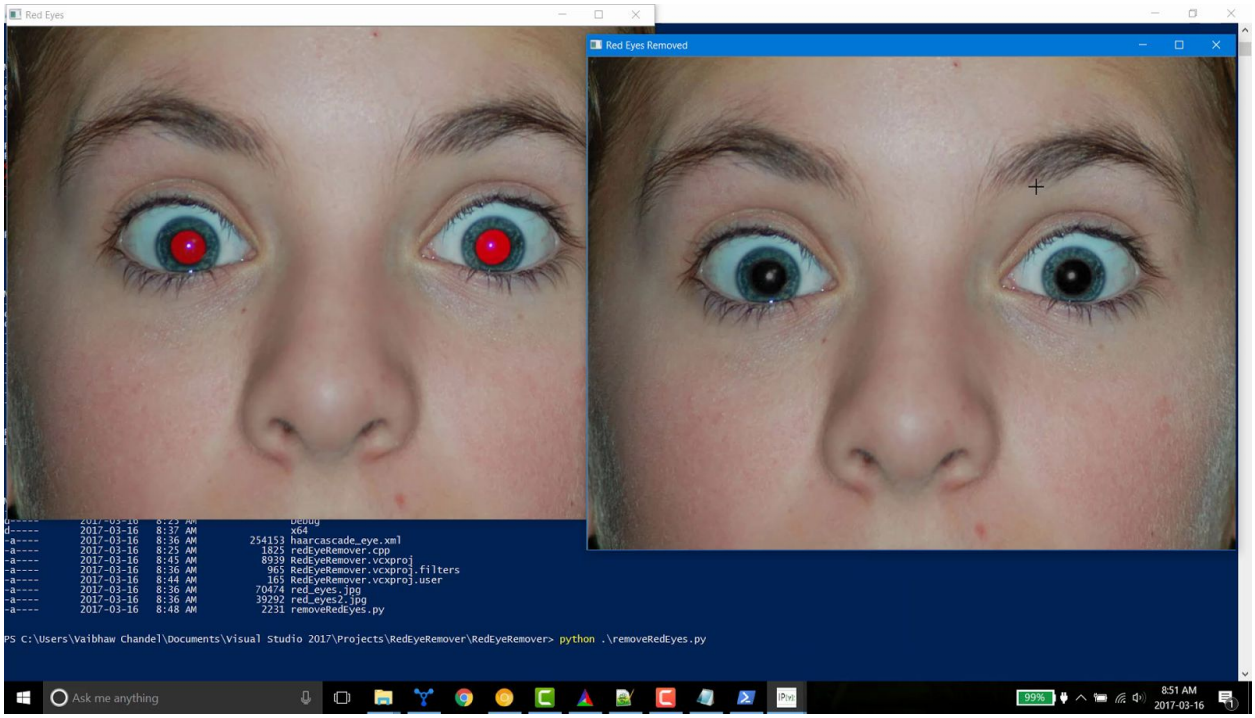
Step 8.2 : Test redEyeRemover application

Open Windows Power Shell and navigate to directory where you have extracted RedEyeRemover.zip. Now run Python code like this:

```
python removeRedEyes.py
```

```
PS C:\Users\Vaibhaw Chandel\Documents\Visual Studio 2017\Projects\RedEyeRemover\RedEyeRemover> ls


    Directory: C:\Users\Vaibhaw Chandel\Documents\Visual Studio 2017\Projects\RedEyeRemover\RedEyeRemover


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2017-03-16   8:25 AM                Debug
d-----        2017-03-16   8:37 AM                x64
-a----        2017-03-16   8:36 AM         254153 haarcascade_eye.xml
-a----        2017-03-16   8:25 AM           1825 redEyeRemover.cpp
-a----        2017-03-16   8:45 AM           8939 RedEyeRemover.vcxproj
-a----        2017-03-16   8:36 AM            965 RedEyeRemover.vcxproj.filters
-a----        2017-03-16   8:44 AM            165 RedEyeRemover.vcxproj.user
-a----        2017-03-16   8:36 AM          70474 red_eyes.jpg
-a----        2017-03-16   8:36 AM          39292 red_eyes2.jpg
-a----        2017-03-16   8:48 AM           2231 removeRedEyes.py.txt

PS C:\Users\Vaibhaw Chandel\Documents\Visual Studio 2017\Projects\RedEyeRemover\RedEyeRemover> mv .\removeRedEyes.py.txt .\removeRedEyes.py
PS C:\Users\Vaibhaw Chandel\Documents\Visual Studio 2017\Projects\RedEyeRemover\RedEyeRemover> ls


    Directory: C:\Users\Vaibhaw Chandel\Documents\Visual Studio 2017\Projects\RedEyeRemover\RedEyeRemover


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2017-03-16   8:25 AM                Debug
d-----        2017-03-16   8:37 AM                x64
-a----        2017-03-16   8:36 AM         254153 haarcascade_eye.xml
-a----        2017-03-16   8:25 AM           1825 redEyeRemover.cpp
-a----        2017-03-16   8:45 AM           8939 RedEyeRemover.vcxproj
-a----        2017-03-16   8:36 AM            965 RedEyeRemover.vcxproj.filters
-a----        2017-03-16   8:44 AM            165 RedEyeRemover.vcxproj.user
-a----        2017-03-16   8:36 AM          70474 red_eyes.jpg
-a----        2017-03-16   8:36 AM          39292 red_eyes2.jpg
-a----        2017-03-16   8:48 AM           2231 removeRedEyes.py

PS C:\Users\Vaibhaw Chandel\Documents\Visual Studio 2017\Projects\RedEyeRemover\RedEyeRemover> python .\removeRedEyes.py
```

If the program runs successfully, you will see two image windows one with red-eyes other with black eyes.



Installing Dlib on Windows

If you have Anaconda installed on your machine, Dlib examples will fail to compile. The reason is that Dlib uses libjpeg and libpng libraries to read/write JPEG and PNG images. Anaconda distribution comes with its own version of libjpeg and libpng library files which are incompatible with Dlib.

Dlib is shipped with libjpeg and libpng source files in “external” folder within dlib. So we will build Dlib core library and Dlib samples using libjpeg and libpng files provided with Dlib. Dlib library and examples get compiled using this technique but at runtime it fails.

Dlib supports OpenCV’s Mat object. So in our projects we will use OpenCV for image read/write operations and pass this image object to Dlib for Dlib specific operations.

Step 1: Install CMake

We have already installed CMake while installing OpenCV.

Step 2: Download Dlib

Download Dlib v19.4 from http://dlib.net/files/dlib-19.4.tar.bz2

Step 3: Build Dlib library

Extract this compressed file. Open Windows PowerShell. Move to directory where you have extracted this file.

If you are running these commands on Command Prompt replace ` (backtick) with ^ (caret).

```
cd dlib-19.4/
mkdir build
cd build
```

```
# If you don’t have Anaconda
cmake -G "Visual Studio 14 2015 Win64" ..

# If you have Anaconda installed on your machine
cmake -G "Visual Studio 14 2015 Win64" `
-DJPEG_INCLUDE_DIR=..\dlib\external\libjpeg `
-DJPEG_LIBRARY=..\dlib\external\libjpeg `
-DPNG_PNG_INCLUDE_DIR=..\dlib\external\libpng `
-DPNG_LIBRARY_RELEASE=..\dlib\external\libpng `
-DZLIB_INCLUDE_DIR=..\dlib\external\zlib `
-DZLIB_LIBRARY_RELEASE=..\dlib\external\zlib `
```

```
..
```

```
cmake --build . --config Release
cd ..
```

Step 4: Build Dlib examples

If you are running these commands on Command Prompt replace ` (backtick) with ^ (caret).

```
cd dlib-19.4/examples
mkdir build
cd build
```

```
# If you don't have Anaconda
cmake -G "Visual Studio 14 2015 Win64" ..

# If you have Anaconda installed on your machine
cmake -G "Visual Studio 14 2015 Win64" `
-DJPEG_INCLUDE_DIR=..\..\dlib\external\libjpeg `
-DJPEG_LIBRARY=..\..\dlib\external\libjpeg `
-DPNG_PNG_INCLUDE_DIR=..\..\dlib\external\libpng `
-DPNG_LIBRARY_RELEASE=..\..\dlib\external\libpng `
-DZLIB_INCLUDE_DIR=..\..\dlib\external\zlib `
-DZLIB_LIBRARY_RELEASE=..\..\dlib\external\zlib `
..
```

```
cmake --build . --config Release
cd ../..
```

Step 5: Install Dlib's Python module (only for Anaconda 3)

Dlib v19.4 can only be installed for Anaconda 3. Anaconda 2 provides an older version of Dlib which is not useful for our case as we will use some features which were introduced in v19.4

```
conda install -c conda-forge dlib=19.4
```

Step 6: Test Dlib's C++ example

We will test Face Landmark Detection demo to check whether we have installed Dlib correctly.

Download trained model of facial landmarks from Dlib's website. Extract this file (shape_predictor_68_face_landmarks.dat.bz2) to Dlib's root directory (dlib-19.4).

Now depending upon whether you had Anaconda installed on your machine or not, we will follow separate methods to test Facial Landmark Detection example.

If you don't have Anaconda installed on your machine

In this case all the example files were built properly. So we can directly run them without making any changes.

```
cd examples\build
```

```
.\Release\face_landmark_detection_ex.exe
..\..\shape_predictor_68_face_landmarks.dat ..\faces\2008_001009.jpg
```

If you have Anaconda installed on your machine

As we planned, we will use OpenCV to read image and convert OpenCV's Mat object to Dlib's object. I will show one such example. We will change face_landmark_detection_ex.cpp and write a CMakeLists.txt using which we will compile this C++ example.

First let's go to dlib/examples directory and make a backup of file CMakeFiles.txt and face_landmark_detection_ex.cpp

```
cd examples
mv CMakeLists.txt CMakeLists.txt.bak
mv face_landmark_detection_ex.cpp face_landmark_detection_ex.cpp.bak
```

Now create a new file named face_landmark_detection_ex.cpp and put following code in it.

```
#include <dlib/opencv.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <dlib/image_processing/frontal_face_detector.h>
#include <dlib/image_processing/render_face_detections.h>
#include <dlib/image_processing.h>
#include <dlib/gui_widgets.h>
#include <dlib/image_io.h>
#include <iostream>
#include <typeinfo>

using namespace dlib;
using namespace std;
```

```
int main(int argc, char** argv)
{
    try
    {
        if (argc == 1)
        {
            cout << "Call this program like this:" << endl;
            cout << "./face_landmark_detection_ex
shape_predictor_68_face_landmarks.dat faces/*.jpg" << endl;
            cout << "\nYou can get the shape_predictor_68_face_landmarks.dat file
from:\n";
            cout <<
"http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2" << endl;
            return 0;
        }

        frontal_face_detector detector = get_frontal_face_detector();
        shape_predictor sp;
        deserialize(argv[1]) >> sp;

        image_window win, win_faces;
        for (int i = 2; i < argc; ++i)
        {
            cout << "processing image " << argv[i] << endl;
                    cv::Mat im_bgr = cv::imread(argv[i], cv::IMREAD_COLOR);
            // Make the image larger so we can detect small faces.
                    cv::resize(im_bgr, im_bgr, cv::Size(), 2.0, 2.0);
                    // Convert OpenCV's Mat to Dlib's cv_image
                    cv_image<bgr_pixel> img(im_bgr);

            std::vector<dlib::rectangle> dets = detector(img);
            cout << "Number of faces detected: " << dets.size() << endl;

            std::vector<full_object_detection> shapes;
            for (unsigned long j = 0; j < dets.size(); ++j)
            {
                full_object_detection shape = sp(img, dets[j]);
                cout << "number of parts: "<< shape.num_parts() << endl;
                cout << "pixel position of first part:  " << shape.part(0) << endl;
                cout << "pixel position of second part: " << shape.part(1) << endl;
                shapes.push_back(shape);
            }

            win.clear_overlay();
            win.set_image(img);
            win.add_overlay(render_face_detections(shapes));

            dlib::array<array2d<rgb_pixel> > face_chips;
            extract_image_chips(img, get_face_chip_details(shapes), face_chips);
            win_faces.set_image(tile_images(face_chips));

            cout << "Hit enter to process the next image..." << endl;
```

```
            cin.get();
        }
    }
    catch (exception& e)
    {
        cout << "\nexception thrown!" << endl;
        cout << e.what() << endl;
    }
}
```

Next we will create CMakeLists.txt file and put following code in it.

```
cmake_minimum_required(VERSION 2.8)
project( redEyeRemover )
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( removeRedEyes removeRedEyes.cpp )
target_link_libraries( removeRedEyes ${OpenCV_LIBS} )
```

Save file and go to Power Shell window and run following commands:

```
mkdir build
cd build
cmake -G "Visual Studio 14 2015 Win64" ..
cmake --build . --config Release
```

Once build is complete, it will generate a file in build\Release folder. Run the executable by running:

```
.\Release\face_landmark_detection_ex.exe
..\..\shape_predictor_68_face_landmarks.dat ..\faces\2008_001009.jpg
```

Step 7: Test Dlib's Python example

Open Windows PowerShell and move to dlib-19.4 directory.

```
cd dlib-19.4/python_examples
```

```
python face_landmark_detection.py ..\shape_predictor_68_face_landmarks.dat
..\examples\faces\2008_001009.jpg
```