ORB SLAM 2 + OpenCV + OpenNI Comprehensive Installation Guide for Windows 10

Table of Contents

Abstract

This guide demonstrates how to compile and install ORB SLAM 2 with OpenCV and OpenNI2 support for the Windows operating system. The shipped samples in the ORB SLAM 2 bundle can then easily be used/rewritten to use an RGB-D camera.

Prerequisites

1. Install Visual Studio 2015 vc14 x64

    - Install all available C/C++ modules/compiler options

2. Install latest CMake for Windows (release >= 3.10)

3a. Install AstraSDK + Orbbec Astra Drivers + OpenNI package (release == 2.3) from:

    - https://orbbec3d.com/develop/

3b. Alternatively install OpenNI Package from:

    - https://structure.io/openni (release == 2.2)

4. Install TortoiseGit from:

- https://tortoisegit.org/

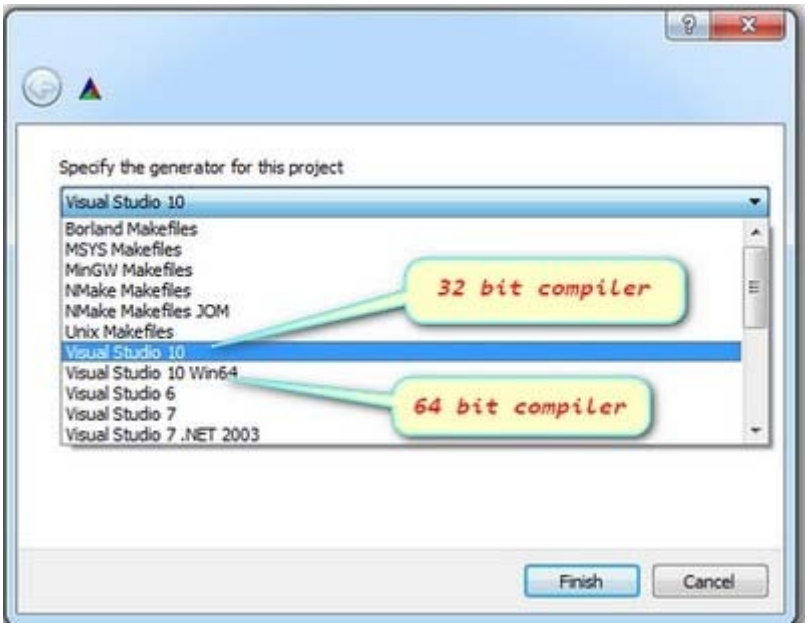
Compile OpenCV with OpenNI Support

A. Compile OpenCV (release >= 3.4.0) with OpenNI support (use C:\opencv\ as folder):

- https://docs.opencv.org/3.4.0/d3/d52/tutorial_windows_install.html

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. 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. 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. Choose the 32 or 64 bit version according to the type of OS you work in. While installing, locate your msysgit (if it does not 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 characters in it, for example D:/OpenCV. For this tutorial, I will suggest you do so. If you use your own path and know, what you are doing – it is OK.
    a. 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/opencv/opencv.git) and to what directory (D:/OpenCV).
    b. 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. 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).

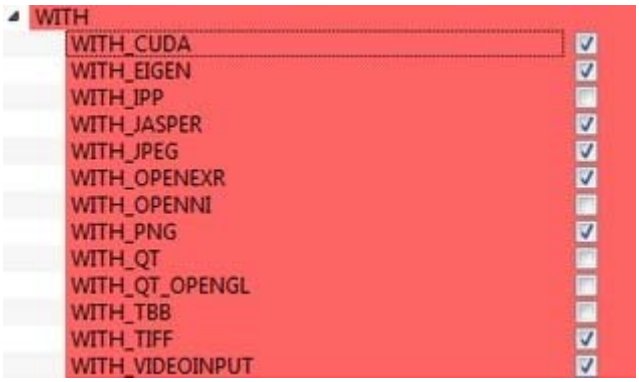
7. Press the Configure button to specify the compiler (and *IDE*) you want to use. Note that in case you can choose between different compilers for making either 64 bit or 32 bit libraries. Select the one you use in your application development.



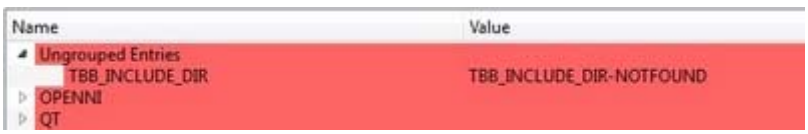
8. CMake will start out and based on your system variables will try to automatically locate as many packages as possible. You can modify the packages to use for the build in the WITH –> WITH_X menu points (where *X* is the package abbreviation).

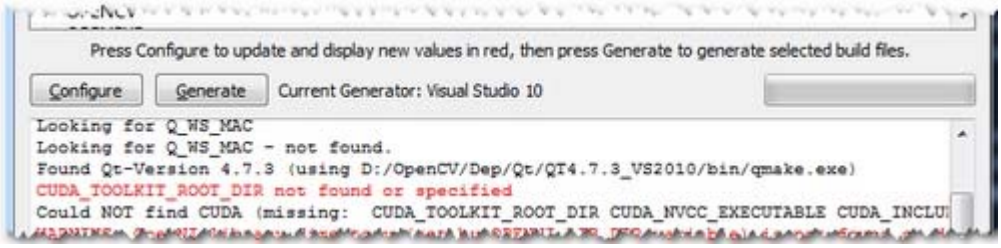
9. **Turn on "WITH_OPENNI2" and search for "opencv world" in the search bar and check this too if not yet!**

10. Here are a list of current packages you can turn on or off:

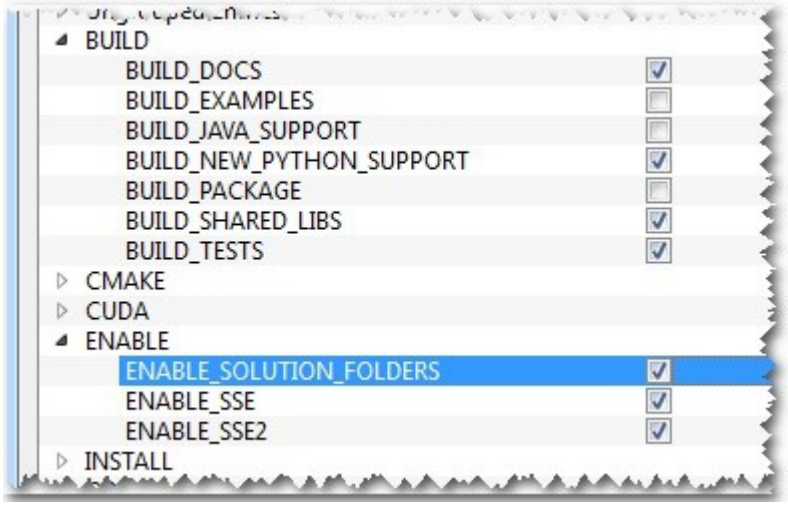


11. Select all the packages you want to use and press again the *Configure* button. For an easier overview of the build options make sure the *Grouped*option under the binary directory selection is turned on. For some of the packages CMake may not find all of the required files or directories. In case 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:

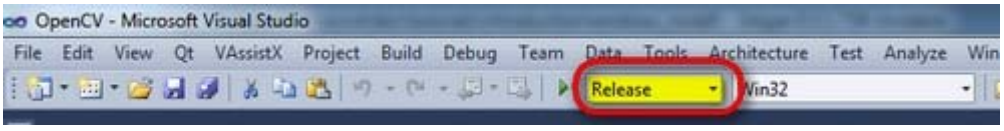




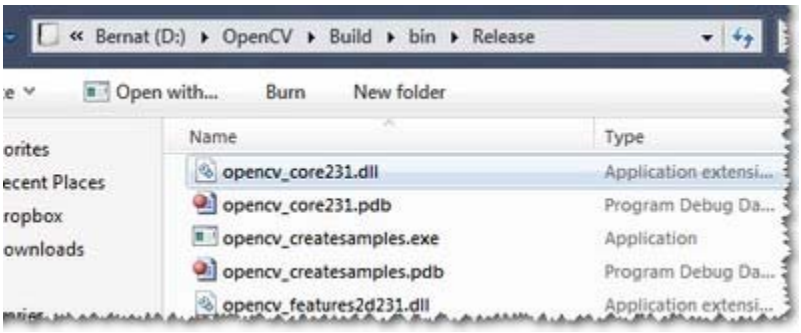
12. For these you need to manually set the queried directories or files path. After this press again the *Configure* button to see if the value entered by you was accepted or not. Do this until all entries are good and you cannot see errors in the field/value or the output part of the GUI. Now I want 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.



13. Furthermore, you need to select what part of OpenCV you want to build.
14. 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.



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



14. Set OpenCV Environment Variables

- Open Windows Enviroment Variables and add the following:
  - Append to the end of "Path" variable: " C:\opencv\build\bin\Release"
  - Add a new System Variable (folder where "OpenCVConfig.cmake" is):
    - OPENCV_DIR :: C:\opencv\build

Compile ORB SLAM 2

Follow instructions at: https://github.com/Phylliida/orbslam-windows

1. Make a directory called build in orbslam-windows/Thirdparty/DBoW2

- Run CMake GUI and set source code to orbslam-windows/Thirdparty/DBoW2 and where to build the binaries to orbslam-windows/Thirdparty/DBoW2/build
- Press Configure and choose Visual Studio 14 2015 Win64 or Visual Studio 12 2013 Win64
- Press Generate
- Open the resulting project in the build directory in Visual Studio
- Change build type to Release (in white box up top, should initially say Debug)
- Right click on DBoW2 project -> Properties -> General: change Target Extension to .lib and Configuration Type to Static Library (.lib)
- Go to C/C++ Tab -> Code Generation and change Runtime Library to Multi-threaded (/MT)
- Build ALL_BUILD. You should get lots of warnings but no errors

2. Make a directory called build in orbslam-windows/Thirdparty/g2o

- Run CMake GUI and set source code to orbslam-windows/Thirdparty/g2o and where to build the binaries to orbslam-windows/Thirdparty/g2o/build
- Press Configure and choose Visual Studio 14 2015 Win64 or Visual Studio 12 2013 Win64
- Press Generate
- Open the resulting project in the build directory in Visual Studio
- Change build type to Release (in white box up top, should initially say Debug)
- Right click on g2o project -> Properties -> General: change Target Extension to .lib and Configuration Type to Static Library (.lib)
- Go to C/C++ Tab -> Code Generation and change Runtime Library to Multi-threaded (/MT)
- Go to C/C++ -> Preprocessor and press the dropdown arrow in the Preprocessor Definitions, then add a new line with WINDOWS on it (no underscore), then press OK, then Apply
- Build ALL_BUILD.

3. Make a directory called build in orbslam-windows/Thirdparty/Pangolin

- Run CMake GUI and set source code to orbslam-windows/Thirdparty/Pangolin and where to build the binaries to orbslam-windows/Thirdparty/Pangolin/build
- Press Configure and choose Visual Studio 14 2015 Win64 or Visual Studio 12 2013 Win64. You'll have a lot of RED and a lot of things that say DIR-NOTFOUND but as long as the window at the bottom says Configuring Done you're fine
- Press Generate
- Open the resulting project in the build directory in Visual Studio
- Change build type to Release (in white box up top, should initially say Debug)
- Build ALL_BUILD. You'll have an error by project testlog that says "cannot open input file 'pthread.lib'" but that doesn't matter cause we don't use testlog. Everything else should build fine, i.e., you should have ========== Build: 18 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========

3. Make a directory called build in orbslam-windows

- Run CMake GUI and set source code to orbslam-windows and where to build the binaries to orbslam-windows/build
- Press Configure and choose Visual Studio 14 2015 Win64 or Visual Studio 12 2013 Win64
- Press Generate
- Open the resulting project in the build directory in Visual Studio
- Change build type to Release (in white box up top, should initially say Debug)
- Right click on ORB_SLAM2 project -> Properties -> General: change Target Extension to .lib and Configuration Type to Static Library (.lib)
- Go to C/C++ Tab -> Code Generation and change Runtime Library to Multi-threaded (/MT)

I had to disable warnings in Orb Slam because otherwise there were so many they crashed visual studio. You will still see a few but not very many

- Right click on the ORB_SLAM2 project (NOT ALL_BUILD) and click Build
- If you're lucky, that will take few minutes then successfully build!

If you want to build any of the examples (such as mono_euroc), do the following:

- Right click on that project and go to Properties -> C/C++ -> Code Generation, and change Runtime Library to Multi-threaded (/MT). Then press apply
- Right click on it and press build

Then you will find them, say if you do mono_, in (orbslam-windows\Examples\Monocular\Release)

B. After instructions followed above and Examples are compiled, copy the content of the following folder from the previously downloaded Orbbec Astra SDK package:

"… ORB SLAM 2\02_Drivers\01_Astra_Drivers__OpenNI_Driver_Package_EXE"

... directly to the folder where the example.exe files are (this folder includes the OpenNI.dll and Astra Driver *.dll)

Post Modifications

If your compiled examples do not run, you might want to copy the following files to the OpenNI2/Drivers/ folder within your examples folder:

1. OniFile.dll + OniFile.pdb
2. Orbbec.dll + orbbec.ini

Moreover there should be OpenNI.ini and OpenNI2.dll be present in your executable folder. The *.ini file points to the driver folder at OpenNI2/Drivers/.

If your compiled samples complain about missing OpenCV libraries, copy the C:/opencv/bin content to your executables path too.

Example Source Code for RGB-D

I just altered the code of mono_euro.cc to avoid setting up a new project within the ORB_SLAM2 solution.

```
#include <iostream>
#include <algorithm>
#include <fstream>
#include <chrono>

#include <opencv2/core/core.hpp>

#include <System.h>

#include <time.h>

using namespace std;

// From http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToQuaternion/

int main(int argc, char **argv)
{
		string vocabPath = "../ORBvoc.txt";
		string settingsPath = "../webcam.yaml";
		if (argc == 1) {

		} else if (argc == 2) {
				vocabPath = argv[1];
		} else if (argc == 3) {
				vocabPath = argv[1];
				settingsPath = argv[2];
		} else {
				cerr << endl << "Usage: mono_euroc.exe path_to_vocabulary path_to_settings" << endl;
				return 1;
		}

		// Create SLAM system. It initializes all system threads and gets ready to process frames.
		ORB_SLAM2::System SLAM(vocabPath, settingsPath, ORB_SLAM2::System::RGBD, true);

		cout << endl << "Start processing ..." << endl;

		cv::VideoCapture cap(CV_CAP_OPENNI2);
		cap.set(CV_CAP_OPENNI_IMAGE_GENERATOR_OUTPUT_MODE, CV_CAP_OPENNI_VGA_30HZ);
		cap.set(CV_CAP_OPENNI_DEPTH_GENERATOR_REGISTRATION, 1);

		// From http://stackoverflow.com/questions/19555121/how-to-get-current-timestamp-in-milliseconds-since-1970-just-the-way-
java-gets
		__int64 now =
std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().time_since_epoch()).count();

		bool init = false;
		cv::Mat image, depth, tcw;		// rgb image, depth map, pose output
		while (true) {

				if (cap.grab()) {

						// CV_CAP_OPENNI_BGR_IMAGE -> CV_8UC3
						// CV_CAP_OPENNI_GRAY_IMAGE -> CV_8UC1 [x]

						// CV_CAP_OPENNI_DEPTH_MAP - depth values in mm(CV_16UC1)
						// CV_CAP_OPENNI_POINT_CLOUD_MAP - XYZ in meters(CV_32FC3) [x]
						// CV_CAP_OPENNI_DISPARITY_MAP - disparity in pixels(CV_8UC1)
						// CV_CAP_OPENNI_DISPARITY_MAP_32F - disparity in pixels(CV_32FC1) [x]
						// CV_CAP_OPENNI_VALID_DEPTH_MASK - mask of valid pixels(not ocluded, not shaded etc.)
(CV_8UC1)

						cap.retrieve(image, CV_CAP_OPENNI_GRAY_IMAGE);
						cap.retrieve(depth, CV_CAP_OPENNI_POINT_CLOUD_MAP);

				} else {
						cout << "ERROR: Could not grab image data." << endl;
				}

				if (!image.data) {
						cout << "ERROR: RGB not retrieved." << endl;
				}

				if (!depth.data) {
						cout << "ERROR: Depth map not retrieved." << endl;
				}

				if (!image.data || !depth.data) {
						return -1;
				}

				__int64 curNow =
std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().time_since_epoch()).count();

				// Pass the image to the SLAM system and returns camera pose (empty if tracking fails)
				tcw = SLAM.TrackRGBD(image, depth, curNow/1000.0);

				// This can write each image with its position to a file if you want
				/*if (!Tcw.empty()) {
						cv::Mat Rwc = Tcw.rowRange(0, 3).colRange(0, 3).t();
						cv::Mat twc = -Rwc*Tcw.rowRange(0, 3).col(3);
						std::ostringstream stream;
```

```
                                        //stream << "imgs/" << Rwc.at<float>(0, 0) << " " << Rwc.at<float>(0, 1) << " " <<
Rwc.at<float>(0, 2) << " " << twc.at<float>(0) << " " <<
                                        //            Rwc.at<float>(1, 0) << " " << Rwc.at<float>(1, 1) << " " << Rwc.at<float>(1, 2)
<< " " << twc.at<float>(1) << " " <<
                                        //Rwc.at<float>(2, 0) << " " << Rwc.at<float>(2, 1) << " " << Rwc.at<float>(2, 2) << " " <<
twc.at<float>(2) << ".jpg";
                                        stream << "imgs/" << curNow << ".jpg";
                                        string fileName = stream.str();
                                        cv::imwrite(fileName, im);
                                } */

                                // This will make a third window with the color images, you need to click on this then press any key to quit
                                cv::imshow("Image", image);
                                cv::imshow("Depth", depth);

                                if (cv::waitKey(1) >= 0)
                                        break;
                }

                // Stop all threads
                SLAM.Shutdown();
                cap.release();

                // Save camera trajectory
                SLAM.SaveKeyFrameTrajectoryTUM("KeyFrameTrajectory.txt");

                return 0;
}
```