OpenCV installation overview

Next Tutorial: **OpenCV configuration options reference**

Table of Contents

There are two ways of installing OpenCV on your machine: download prebuilt version for your platform or compile from sources.

Prebuilt version

In many cases you can find prebuilt version of OpenCV that will meet your needs.

Packages by OpenCV core team

Packages for Android, iOS and Windows built with default parameters and recent compilers are published for each release, they do not contain *opencv_contrib* modules.

- GitHub releases: https://github.com/opencv/opencv/releases
- SourceForge.net: https://sourceforge.net/projects/opencvlibrary/files/

Third-party packages

Other organizations and people maintain their own binary distributions of OpenCV. For example:

- System packages in popular Linux distributions (https://pkgs.org/search/?q=opencv)
- PyPI (https://pypi.org/search/?q=opencv)
- Conda (https://anaconda.org/search?q=opencv)
- Conan (https://github.com/conan-community/conan-opencv)
- vcpkg (https://github.com/microsoft/vcpkg/tree/master/ports/opencv)
- NuGet (https://www.nuget.org/packages?q=opencv)
- Brew (https://formulae.brew.sh/formula/opencv)
- Maven (https://search.maven.org/search?q=opencv)

Build from sources

It can happen that existing binary packages are not applicable for your use case, then you'll have to build custom version of OpenCV by yourself. This section gives a high-level overview of the build process, check tutorial for specific platform for actual build instructions.

OpenCV uses CMake build management system for configuration and build, so this section mostly describes generalized process of building software with CMake.

Step 0: Prerequisites

Install C++ compiler and build tools. On *NIX platforms it is usually GCC/G++ or Clang compiler and Make or Ninja build tool. On Windows it can be Visual Studio IDE or MinGW-w64 compiler. Native toolchains for Android are provided in the Android NDK. XCode IDE is used to build software for OSX and iOS platforms.

Install CMake from the official site or some other source.

Get other third-party dependencies: libraries with extra functionality like decoding videos or showing GUI elements; libraries providing optimized implementations of selected algorithms; tools used for documentation generation and other extras. Check **OpenCV configuration options reference** for available options and corresponding dependencies.

Step 1: Get software sources

Typical software project consists of one or several code repositories. OpenCV have two repositories with code: *opencv* - main repository with stable and actively supported algorithms and *opencv_contrib* which contains experimental and non-free (patented) algorithms; and one repository with test data: *opencv_extra*.

You can download a snapshot of repository in form of an archive or clone repository with full history.

To download snapshot archives:

- Go to https://github.com/opencv/opencv/releases and download "Source code" archive from any release.
- (optionally) Go to https://github.com/opencv/opencv_contrib/releases and download "Source code" archive for the same release as *opencv*
- (optionally) Go to https://github.com/opencv/opencv_extra/releases and download "Source code" archive for the same release as *opencv*
- Unpack all archives to some location

To clone repositories run the following commands in console (*git* must be installed):

```
git clone https://github.com/opencv/opencv
git -C opencv checkout <some-tag>
```

```
# optionally
git clone https://github.com/opencv/opencv_contrib
git -C opencv_contrib checkout <same-tag-as-opencv>

# optionally
git clone https://github.com/opencv/opencv_extra
git -C opencv_extra checkout <same-tag-as-opencv>
```

Note

If you want to build software using more than one repository, make sure all components are compatible with each other. For OpenCV it means that *opencv* and *opencv_contrib* repositories must be checked out at the same tag or that all snapshot archives are downloaded from the same release.

When choosing which version to download take in account your target platform and development tools versions, latest versions of OpenCV can have build problems with very old compilers and vice versa. We recommend using latest release and fresh OS/compiler combination.

Step 2: Configure

At this step CMake will verify that all necessary tools and dependencies are available and compatible with the library and will generate intermediate files for the chosen build system. It could be Makefiles, IDE projects and solutions, etc. Usually this step is performed in newly created build directory:

```
cmake -G<generator> <configuration-options> <source-directory>
```

Note

`cmake-gui` application allows to see and modify available options using graphical user interface. See https://cmake.org/runningcmake/ for details.

Step 3: Build

During build process source files are compiled into object files which are linked together or otherwise combined into libraries and applications. This step can be run using universal command:

```
cmake --build <build-directory> <build-options>
```

... or underlying build system can be called directly:

```
make
```

(optional) Step 3: Install

During installation procedure build results and other files from build directory will be copied to the install location. Default installation location is `/usr/local` on UNIX and `C:/Program Files` on Windows. This location can be changed at the configuration step by setting `CMAKE_INSTALL_PREFIX` option. To perform installation run the following command:

```
cmake --build <build-directory> --target install <other-options>
```

Note

This step is optional, OpenCV can be used directly from the build directory.

If the installation root location is a protected system directory, so the installation process must be run with superuser or administrator privileges (e.g. `sudo cmake ...`).

(optional) Step 4: Build plugins

It is possible to decouple some of OpenCV dependencies and make them optional by extracting parts of the code into dynamically-loaded plugins. It helps to produce adaptive binary distributions which can work on systems with less dependencies and extend functionality just by installing missing libraries. For now modules *core*, *videoio* and *highgui* support this mechanism for some of their dependencies. In some cases it is possible to build plugins together with OpenCV by setting options like `VIDEOIO_PLUGIN_LIST` or `HIGHGUI_PLUGIN_LIST` , more options related to this scenario can be found in the **OpenCV configuration options reference**. In other cases plugins should be built separately in their own build procedure and this section describes such standalone build process.

Note

It is recommended to use compiler, configuration and build options which are compatible to the one used for OpenCV build, otherwise resulting library can refuse to load or cause other runtime problems. Note that some functionality can be limited or work slower when backends are loaded dynamically due to extra barrier between OpenCV and corresponding third-party library.

Build procedure is similar to the main OpenCV build, but you have to use special CMake projects located in corresponding subdirectories, these folders can also contain reference scripts and Docker images. It is important to use `opencv_<module>_<backend>` name prefix for plugins so that loader is able to find them. Each supported prefix can be used to load only one library, however multiple candidates can be probed for a single prefix. For example, you can have *libopencv_videoio_ffmpeg_3.so* and *libopencv_videoio_ffmpeg_4.so* plugins and the first one which can be loaded successfully will occupy internal slot and stop probing process. Possible prefixes and project locations are presented in the table below:

module	backends	location
core	parallel_tbb, parallel_onetbb, parallel_openmp	*opencv/modules/core/misc/plugins*
highgui	gtk, gtk2, gtk3	*opencv/modules/highgui/misc/plugins*
videoio	ffmpeg, gstreamer, intel_mfx, msmf	*opencv/modules/videoio/misc*

Example:

```
# set-up environment for TBB detection, for example:
#   export TBB_DIR=<dir-with-tbb-cmake-config>
cmake -G<generator> \
    -DOPENCV_PLUGIN_NAME=opencv_core_tbb_<suffix> \
    -DOPENCV_PLUGIN_DESTINATION=<dest-folder> \
    -DCMAKE_BUILD_TYPE=<config> \
    <opencv>/modules/core/misc/plugins/parallel_tbb
cmake --build . --config <config>
```

Note

On Windows plugins must be linked with existing OpenCV build. Set `OpenCV_DIR` environment or CMake variable to the directory with *OpenCVConfig.cmake* file, it can be OpenCV build directory or some path in the location where you performed installation.