Jetson/Installing OpenCV

< [Jetson](https://elinux.org/Jetson)

**Contents**

 [hide]

* [1Installing OpenCV (including the GPU module) on Jetson TK1](https://elinux.org/Jetson/Installing_OpenCV#Installing_OpenCV_.28including_the_GPU_module.29_on_Jetson_TK1)
* [2Prebuilt OpenCV library versus Building the OpenCV library from source](https://elinux.org/Jetson/Installing_OpenCV#Prebuilt_OpenCV_library_versus_Building_the_OpenCV_library_from_source)
* [3Native vs Cross-development](https://elinux.org/Jetson/Installing_OpenCV#Native_vs_Cross-development)
* [4Natively compiling the OpenCV library from source onboard the device](https://elinux.org/Jetson/Installing_OpenCV#Natively_compiling_the_OpenCV_library_from_source_onboard_the_device)
* [5Testing OpenCV](https://elinux.org/Jetson/Installing_OpenCV#Testing_OpenCV)
* [6Getting to know OpenCV](https://elinux.org/Jetson/Installing_OpenCV#Getting_to_know_OpenCV)

Installing OpenCV (including the GPU module) on Jetson TK1

First you should download & install the CUDA Toolkit by following the [Installing CUDA](https://elinux.org/Tegra/Installing_CUDA) instructions, since it is needed by OpenCV.

And if you haven't already added the "universe" repository to your system, do it now since you will need it for some OpenCV dependencies:

sudo apt-add-repository universe

sudo apt-get update

Once you have installed the [CUDA Toolkit](https://elinux.org/Tegra/Installing_CUDA), you can begin to install OpenCV. You have several options (deciding between prebuilt library vs building the library from source, and then deciding whether to compile your code onboard the Jetson TK1 or cross-compile it from your desktop).

Prebuilt OpenCV library versus Building the OpenCV library from source

There are two compatible implementations of OpenCV available for [Jetson TK1](https://elinux.org/Jetson_TK1), but only the first option gives you the CPU optimized library, so you are highly recommended to use this first option:

**Option 1) Prebuilt OpenCV4Tegra library for L4T**:

OpenCV4Tegra is a CPU & GPU optimized version of OpenCV, available as a prebuilt library in [JetPack](https://developer.nvidia.com/jetson-tk1-development-pack) or as separate .deb packages (available for [L4T rel-19](https://developer.nvidia.com/linux-tegra-rel-19) and [L4T rel-21](https://developer.nvidia.com/linux-tegra-rel-21)).

eg: For OpenCV v2.4.10.1, run the following commands to manually install the prebuilt packages:

Under L4T 19.x:

sudo dpkg -i libopencv4tegra\_2.4.10.1\_armhf.deb

sudo dpkg -i libopencv4tegra-dev\_2.4.10.1\_armhf.deb

Under L4T 21.x:

sudo dpkg -i libopencv4tegra-repo\_l4t-r21\_2.4.10.1\_armhf.deb (does not work, please fix!)

sudo apt-get update

sudo apt-get install libopencv4tegra libopencv4tegra-dev

Continue to the "Testing OpenCV" section lower in this page to ensure OpenCV4Tegra works on your device.

**Option 2) Building the public OpenCV library from source**:

If you want the latest OpenCV code, or you want to customize the OpenCV library, and don't want NVIDIA's CPU & multi-core optimizations of OpenCV4Tegra, then follow the instructions below to compile the OpenCV library from source code. Note that you won't obtain the full performance of OpenCV4Tegra.

**Note about SIFT/SURF in the nonfree module**: OpenCV4Tegra doesn't include the opencv\_nonfree package (containing SIFT & SURF feature detectors) since those algorithms are patented by other companies and therefore anyone using opencv\_nonfree is at risk of liability.

If you need something from the nonfree module, you have 2 options:

1. Analyze the public OpenCV source code then copy/paste the parts of the nonfree module that you want (eg: SURF feature detector) from OpenCV into your own project. You will have the CPU optimizations of OpenCV4Tegra for most of your code and will have the GPU module and will have the non-optimized patented code that you need from the nonfree package such as SURF. So this option gives full performance (for everything except the nonfree code) but is tedious.
2. Ignore OpenCV4Tegra, and instead, download & build public OpenCV (by following the instructions below for natively compiling the OpenCV library from source). You will still have the GPU module but not any CPU optimizations, but you won't need to spend time ripping out parts of the OpenCV non-free module code. So this option is easiest but produces slower code if you are running most of your code on CPU.

Native vs Cross-development

Just like with the [CUDA development guide](https://elinux.org/Jetson/Installing_CUDA), you have two options for developing OpenCV applications for [Jetson TK1](https://elinux.org/Jetson_TK1):

* **native compilation** (compiling code onboard the [Jetson TK1](https://elinux.org/Jetson_TK1))
* **cross-compilation** (compiling code on an x86 desktop in a special way so it can execute on the [Jetson TK1](https://elinux.org/Jetson_TK1) target device).

Native compilation is generally the easiest option, but takes longer to compile, whereas cross-compilation is typically more complex to configure and debug, but for large projects it will be noticeably faster at compiling.

Natively compiling the OpenCV library from source onboard the device

Note: Compiling OpenCV from source will give you NVIDIA's CUDA GPU optimizations but not NVIDIA's ARM NEON CPU optimizations that are only available in the closed-source prebuilt OpenCV4Tegra packages.

If you haven't added the "universal" repository to Ubuntu, then do it now:

sudo add-apt-repository universe

sudo apt-get update

Now you need to install many libraries:

# Some general development libraries

sudo apt-get -y install build-essential make cmake cmake-curses-gui g++

# libav video input/output development libraries

sudo apt-get -y install libavformat-dev libavutil-dev libswscale-dev

# Video4Linux camera development libraries

sudo apt-get -y install libv4l-dev

# Eigen3 math development libraries

sudo apt-get -y install libeigen3-dev

# OpenGL development libraries (to allow creating graphical windows)

sudo apt-get -y install libglew1.6-dev

# GTK development libraries (to allow creating graphical windows)

sudo apt-get -y install libgtk2.0-dev

Also, if you think you will use Gstreamer (skip if you don't know what it is), install those dependencies:

# (Optional) Gstreamer development libraries (for Network IP camera input)

sudo apt-get install libgstreamer1.0 libgstreamer1.0-dev libgstreamer-plugins-bad1.0-0 libgstreamer-plugins-base1.0-0 libgstreamer-plugins-base1.0-dev

Download the source code of OpenCV (for Linux) onto your device. eg: Open the webpage of the [latest release of OpenCV](https://github.com/opencv/opencv/releases/latest) and click on the zip file, or from the command-line you can run this command on the device (but perhaps change 'VERSION' to a newer version of OpenCV):

VERSION=3.4.4

cd ~/Downloads

rm ${VERSION}.zip

wget [https://github.com/opencv/opencv/archive/${VERSION}.zip](https://github.com/opencv/opencv/archive/$%7BVERSION%7D.zip)

Unzip the OpenCV source code:

unzip ${VERSION}.zip

mv opencv-${VERSION} ~/

Configure OpenCV using CMake. (For compatibility with Tegra K1, use CUDA\_ARCH\_BIN="3.2", but you can increase this if you only care about newer Tegra's, such as CUDA\_ARCH\_BIN="7.2" if you only care about Tegra Xavier):

cd ~/opencv-${VERSION}

mkdir build

cd build

cmake -DWITH\_CUDA=ON -DCUDA\_ARCH\_BIN="3.2" -DCUDA\_ARCH\_PTX="" -DBUILD\_TESTS=OFF -DBUILD\_PERF\_TESTS=OFF ..

You can run cmake multiple times to add more settings, such as adding the contrib modules or using TBB and fast math for better performance at potential loss of some accuracy:

cmake -DCMAKE\_BUILD\_TYPE=RELEASE -DWITH\_CUBLAS=ON -DENABLE\_FAST\_MATH=ON -DCUDA\_FAST\_MATH=ON -DWITH\_TBB=ON ..

And if you want to also use the OpenCV "contrib" modules:

cd ~/Downloads

rm ${VERSION}.zip

wget [https://github.com/opencv/opencv\_contrib/archive/${VERSION}.zip](https://github.com/opencv/opencv_contrib/archive/$%7BVERSION%7D.zip)

unzip ${VERSION}.zip

mv opencv\_contrib-${VERSION} ~/

cd ~/opencv-${VERSION}/build

cmake -DOPENCV\_EXTRA\_MODULES\_PATH=~/opencv\_contrib-${VERSION}/modules ..

If you want to customize any more of the build settings such as whether to support Firewire cameras or Qt GUI, it is easiest to use the curses interactive version of CMake from here on:

ccmake ..

(Change any settings you want, then click Configure and Generate).

Now you should be ready to build OpenCV and then install it. Unfortunately, OpenCV is currently experiencing a problem with CMake where installing the built libraries (that normally takes a few seconds) re-compiles the whole OpenCV (that normally takes close to an hour). So to save time, instead of running "make -j4 ; make install", we will build & install OpenCV using a single command.

To build the OpenCV library using 4 Tegra CPU cores (takes around 40 minutes on Tegra K1):

make -j4

(Or on Tegra Xavier, run "make -j8" that takes around 25 minutes on Tegra Xavier)

To install the built OpenCV library by copying the OpenCV library to "/usr/local/include" and "/usr/local/lib":

sudo make install

Finally, make sure your system searches the "/usr/local/lib" folder for libraries, and update the library cache:

echo "# Use OpenCV and other custom-built libraries." >> ~/.bashrc

echo "export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/local/lib/" >> ~/.bashrc

source ~/.bashrc

sudo ldconfig

Testing OpenCV

Compile & run a few of the OpenCV sample programs to make sure OpenCV is working. To get the OpenCV sample codes, you can download the [full OpenCV source code](http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.9/), or just download each sample individually from the [CPP folder](https://github.com/Itseez/opencv/tree/2.4.9.1/samples/cpp) and the [GPU folder](https://github.com/Itseez/opencv/tree/2.4.9.1/samples/gpu). For simple programs you can just link to a few OpenCV libs, but for other programs you might as well link to all the OpenCV libs:

# Make sure we have installed a C++ compiler.

sudo apt-get install build-essential g++

# Test a simple OpenCV program. Creates a graphical window, hence you should plug a HDMI monitor in or use a remote viewer such as X Tunneling or VNC or TeamViewer on your desktop.

cd ~/opencv-2.4.9/samples/cpp

g++ edge.cpp -lopencv\_core -lopencv\_imgproc -lopencv\_highgui -o edge

(*Or for OpenCV 3.0: g++ edge.cpp -lopencv\_core -lopencv\_imgproc -lopencv\_highgui -lopencv\_imgcodecs -o edge*)

./edge

# If you have a USB webcam plugged in to your board, then test one of the live camera programs and linking to every OpenCV module.

g++ laplace.cpp -lopencv\_core -lopencv\_imgproc -lopencv\_highgui -lopencv\_calib3d -lopencv\_contrib -lopencv\_features2d -lopencv\_flann -lopencv\_gpu -lopencv\_legacy -lopencv\_ml -lopencv\_objdetect -lopencv\_photo -lopencv\_stitching -lopencv\_superres -lopencv\_video -lopencv\_videostab -o laplace

(*Or for OpenCV 3.0 with full GPU libs: g++ laplace.cpp -lopencv\_adas -lopencv\_bgsegm -lopencv\_bioinspired -lopencv\_calib3d -lopencv\_ccalib -lopencv\_core -lopencv\_datasets -lopencv\_face -lopencv\_features2d -lopencv\_flann -lopencv\_highgui -lopencv\_imgcodecs -lopencv\_imgproc -lopencv\_latentsvm -lopencv\_line\_descriptor -lopencv\_ml -lopencv\_objdetect -lopencv\_optflow -lopencv\_photo -lopencv\_reg -lopencv\_rgbd -lopencv\_saliency -lopencv\_shape -lopencv\_stitching -lopencv\_superres -lopencv\_surface\_matching -lopencv\_text -lopencv\_tracking -lopencv\_videoio -lopencv\_video -lopencv\_videostab -lopencv\_xfeatures2d -lopencv\_ximgproc -lopencv\_xobjdetect -lopencv\_xphoto -o laplace*)

(*Or for regular OpenCV 3.0: g++ laplace.cpp -lopencv\_calib3d -lopencv\_core -lopencv\_features2d -lopencv\_flann -lopencv\_highgui -lopencv\_imgcodecs -lopencv\_imgproc -lopencv\_ml -lopencv\_objdetect -lopencv\_photo -lopencv\_shape -lopencv\_stitching -lopencv\_superres -lopencv\_videoio -lopencv\_video -lopencv\_videostab -o laplace*)

./laplace

# Test a GPU accelerated OpenCV sample.

cd ../gpu

g++ houghlines.cpp -lopencv\_core -lopencv\_imgproc -lopencv\_highgui -lopencv\_calib3d -lopencv\_contrib -lopencv\_features2d -lopencv\_flann -lopencv\_gpu -lopencv\_legacy -lopencv\_ml -lopencv\_objdetect -lopencv\_photo -lopencv\_stitching -lopencv\_superres -lopencv\_video -lopencv\_videostab -o houghlines

./houghlines ../cpp/logo\_in\_clutter.png

If the houghlines GPU sample program worked then you have successfully installed OpenCV and its GPU module!

Getting to know OpenCV

Now you can start building your own projects using OpenCV on CPU and GPU, such as by following the [Using OpenCV with gcc and CMake](http://docs.opencv.org/doc/tutorials/introduction/linux_gcc_cmake/linux_gcc_cmake.html) introduction tutorial then following many of the official [OpenCV Tutorials](http://docs.opencv.org/doc/tutorials/tutorials.html), playing with the sample GPU programs in the **samples/gpu** folder of OpenCV and the many sample CPU programs in the **samples/cpp** folder. To get offline documentation and tutorials of OpenCV, the easiest way is to download the [OpenCV Reference Manual](http://docs.opencv.org/opencv2refman.pdf) PDF file, or if you want offline HTML docs then try building it:

sudo apt-get install python sphinx-common python-sphinx texlive-binaries

make html\_docs

Then open the **"doc/\_html/index.html"** page.

You can learn more about NVIDIA's Tegra hardware acceleration of OpenCV on the [OpenCV Performance](https://elinux.org/Tegra/OpenCV_Performance" \o "Tegra/OpenCV Performance) page, including a long list showing how much electrical power is used by Jetson TK1 for various OpenCV & computer vision sample programs.

Navigation menu

* [Log in](https://elinux.org/index.php?title=Special:UserLogin&returnto=Jetson%2FInstalling+OpenCV)
* [Request account](https://elinux.org/Special:RequestAccount)
* [Page](https://elinux.org/Jetson/Installing_OpenCV)
* [Discussion](https://elinux.org/Talk:Jetson/Installing_OpenCV)
* [Read](https://elinux.org/Jetson/Installing_OpenCV)
* [View source](https://elinux.org/index.php?title=Jetson/Installing_OpenCV&action=edit)
* [View history](https://elinux.org/index.php?title=Jetson/Installing_OpenCV&action=history)

**Search**

Top of Form



Bottom of Form

* [Main Page](https://elinux.org/Main_Page)
* [Community portal](https://elinux.org/eLinux.org:Community_portal)
* [Current events](https://elinux.org/eLinux.org:Current_events)
* [Recent changes](https://elinux.org/Special:RecentChanges)
* [Help](https://www.mediawiki.org/wiki/Special:MyLanguage/Help:Contents)
* [Volunteering](https://elinux.org/Sitesupport-url)
* [Bug Tracker](http://bugs.elinux.org/)

Where else to find us

* [Google+ Community](https://plus.google.com/u/0/communities/113852495250648539361)
* [Twitter (@elinux)](https://twitter.com/elinux)
* [#elinux on Freenode](NULL#elinux@irc.freenode.net)
* [Facebook (@elinux.org)](https://www.facebook.com/elinuxorg-750739005020534/)
* [Mailing Lists](http://www.elinux.org/ELinuxWiki:Mailing_List)

Tools

* [What links here](https://elinux.org/Special:WhatLinksHere/Jetson/Installing_OpenCV)
* [Related changes](https://elinux.org/Special:RecentChangesLinked/Jetson/Installing_OpenCV)
* [Special pages](https://elinux.org/Special:SpecialPages)
* [Printable version](https://elinux.org/index.php?title=Jetson/Installing_OpenCV&printable=yes)
* [Permanent link](https://elinux.org/index.php?title=Jetson/Installing_OpenCV&oldid=484376)
* [Page information](https://elinux.org/index.php?title=Jetson/Installing_OpenCV&action=info)
* This page was last edited on 8 January 2019, at 19:26.
* Content is available under [a Creative Commons Attribution-ShareAlike 3.0 Unported License](https://creativecommons.org/licenses/by-sa/3.0/) unless otherwise noted.