

OpenCV 4 on Raspbian Buster

The following tutorial has been adapted from Dr. Adrian Rosebrock with PyimageSearch.com, and from Q-engineering.eu

Once you have the hardware ready, you'll need to **flash a fresh copy of the Raspbian Buster operating system** to the microSD card.

1. Head on over to the [official BusterOS download page \(Figure 2\)](#), and start your download. I recommend the *"Raspbian Buster with Desktop and recommended software"*.
2. Download [Balena Etcher](#) — software for flashing memory cards. It works on every major OS.

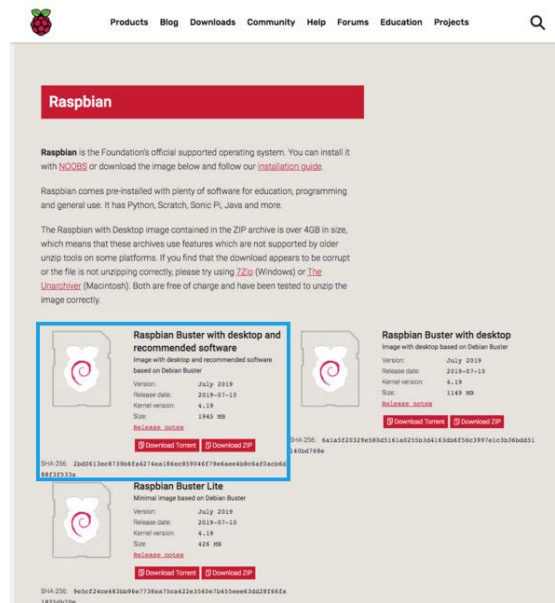


Figure 1: Download Raspbian Buster for your Raspberry Pi and OpenCV 4.

3. Use Etcher to flash BusterOS to your memory card (**Figure 3**).

After downloading the Raspbian Buster .img file you can flash it to your micro-SD card using Etcher:



Figure 2: Flash Raspbian Buster with Etcher. We will use BusterOS to install OpenCV 4 on our Raspberry Pi 4.

After a few minutes the flashing process should be complete — slot the micro-SD card into your Raspberry Pi 4 and then boot.

From there you can move on to the rest of the OpenCV install steps in this guide.

Step #1: Expand filesystem and reclaim space

For the remainder of this tutorial I'll be making the following assumptions:

1. You are working with a brand new, fresh install of **Raspbian Buster** (see the previous section to learn how to flash Buster to your microSD).
2. You are comfortable with the command line and Unix environments.
3. You have an **SSH or VNC connection established with your Pi**. Alternatively, you could use a keyboard + mouse + screen.

Go ahead and insert your microSD into your Raspberry Pi and boot it up with a screen attached.

Once booted, configure your WiFi/ethernet settings to connect to the internet (you'll need an internet connection to download and install required packages for OpenCV).

From there you can **use SSH** as I have done, or go ahead and open a terminal.

The first step is to run `raspi-config` and expand your filesystem:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo raspi-config
```

And then select the ***“7 Advanced Options”*** menu item:

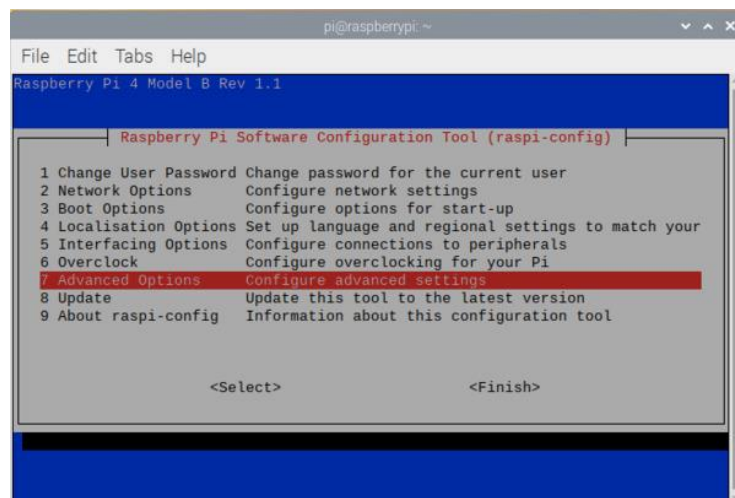


Figure 3: The ``raspi-config`` configuration screen for Raspbian Buster. Select ``7 Advanced Options`` so that we can expand our filesystem.

Followed by selecting ***“A1 Expand filesystem”***:

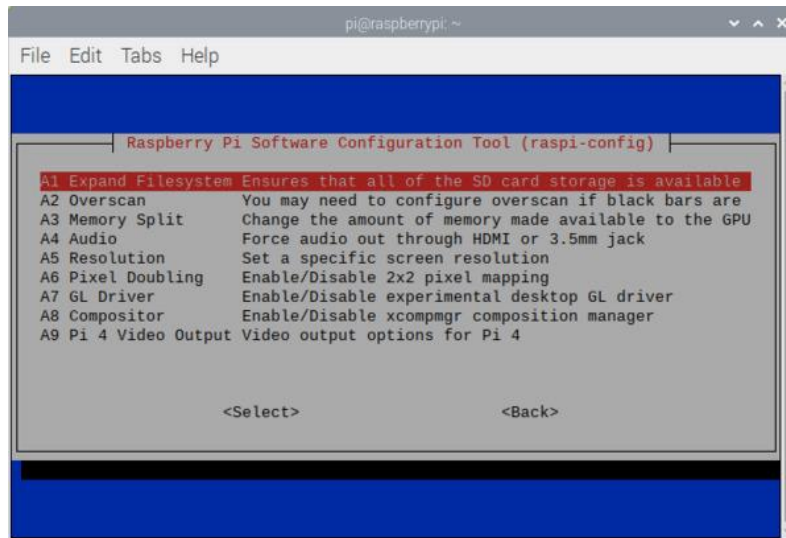


Figure 4: The ‘A1 Expand Filesystem’ menu item allows you to expand the filesystem on your microSD card containing the Raspberry Pi Buster operating system. Then we can proceed to install OpenCV 4.

Once prompted, you should select the first option, **“A1 Expand File System”**, hit enter on your keyboard, arrow down to the **“<Finish>”** button, and then reboot your Pi — you may be prompted to reboot, but if you aren’t you can execute:

Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster

```
$ sudo reboot
```

After rebooting, your file system should have been expanded to include all available space on your micro-SD card. You can verify that the disk has been expanded by executing `df -h` and examining the output:

Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster

```
$ df -h
```

```
Filesystem Size Used Avail Use% Mounted on
```

```
/dev/root 29G 5.3G 23G 20% /
```

```
devtmpfs 1.8G 0 1.8G 0% /dev
```

```
tmpfs 1.9G 0 1.9G 0% /dev/shm
```

```
tmpfs 1.9G 8.6M 1.9G 1% /run
```

```
tmpfs 5.0M 4.0K 5.0M 1% /run/lock
```

```
tmpfs 1.9G 0 1.9G 0% /sys/fs/cgroup
```

```
/dev/mmcblk0p1 253M 40M 213M 16% /boot
```

```
tmpfs 386M 0 386M 0% /run/user/1000
```

As you can see, my Raspbian filesystem has been expanded to include all 32GB of the micro-SD card.

Step #2: Install dependencies

The following commands will update and upgrade any existing packages, followed by installing dependencies, I/O libraries, and optimization packages for OpenCV.

The first step is to update and upgrade any existing packages:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo apt-get update && sudo apt-get upgrade
```

We then need to install some developer tools, including **CMake**, which helps us configure the OpenCV build process:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo apt-get install -y build-essential cmake pkg-config
```

Next, we need to install some image I/O packages that allow us to load various image file formats from disk. Examples of such file formats include JPEG, PNG, TIFF, etc.:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo apt-get install -y libjpeg-dev libtiff5-dev libjasper-dev libpng-dev
```

Just as we need image I/O packages, we also need video I/O packages. These libraries allow us to read various video file formats from disk as well as work directly with video streams:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo apt-get install -y libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
```

```
$ sudo apt-get install -y libxvidcore-dev libx264-dev
```

The OpenCV library comes with a sub-module named `highgui` which is used to display images to our screen and build basic GUIs. In order to compile the `highgui` module, we need to install the GTK development library and prerequisites:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo apt-get install -y libfontconfig1-dev libcairo2-dev
```

```
$ sudo apt-get install -y libgdk-pixbuf2.0-dev libpango1.0-dev
```

```
$ sudo apt-get install -y libgtk2.0-dev libgtk-3-dev
```

Many operations inside of OpenCV (namely matrix operations) can be optimized further by installing a few extra dependencies:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo apt-get install -y libatlas-base-dev gfortran
```

These optimization libraries are *especially important* for resource-constrained devices such as the Raspberry Pi.

The following are for HDF5 datasets and Qt GUIs:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo apt-get install -y libhdf5-dev libhdf5-serial-dev libhdf5-103
```

```
$ sudo apt-get install -y libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5
```

Additionally, there are a few libraries that will be helpful for Basic Linear Algebra Subprograms (BLAS) and multicore processing (`libtbb`)

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo apt-get install -y gcc-arm* protobuf-compiler
```

```
$ sudo apt-get install -y libopenblas-dev libblas-dev
```

```
$ sudo apt-get install -y libtbb2 libtbb-dev libdc1394-22-dev
```

Lastly, let's install Python 3 header files so we can compile OpenCV with Python bindings:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo apt-get install python3-dev
```

If you're working with a fresh install of the OS, it is possible that these versions of Python are already at the newest version (you'll see a terminal message stating this).

Step #3: Create your Python virtual environment and install NumPy

We'll be using Python **virtual environments**, a best practice when working with Python.

A Python virtual environment is an *isolated* development/testing/production environment on your system — it is fully sequestered from other environments. Best of all, you can manage the Python packages inside your virtual environment inside with pip (Python's package manager).

Of course, there are alternatives for managing virtual environments and packages (namely Anaconda/conda). I've used/tried them all, but have settled on pip, **virtualenv**, and **virtualenvwrapper** as the preferred tools that I install on all of my systems. If you use the same tools as me, you'll receive the best support from me.

You can install pip using the following commands:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ wget https://bootstrap.pypa.io/get-pip.py
```

```
$ sudo python get-pip.py
```

```
$ sudo python3 get-pip.py
```

```
$ sudo rm -rf ~/.cache/pip
```

Let's install virtualenv and virtualenvwrapper now

Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster

```
$ sudo pip install virtualenv virtualenvwrapper
```

Once the installation is complete, open up your `~/.bashrc` file using the following command:

Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster

```
$ nano ~/.bashrc
```

...and append the following lines to the *bottom* of the file:

Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster

```
# virtualenv and virtualenvwrapper
```

```
export WORKON_HOME=$HOME/.virtualenvs
```

```
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
```

```
source /usr/local/bin/virtualenvwrapper.sh
```

```
export VIRTUALENVWRAPPER_ENV_BIN_DIR=bin
```

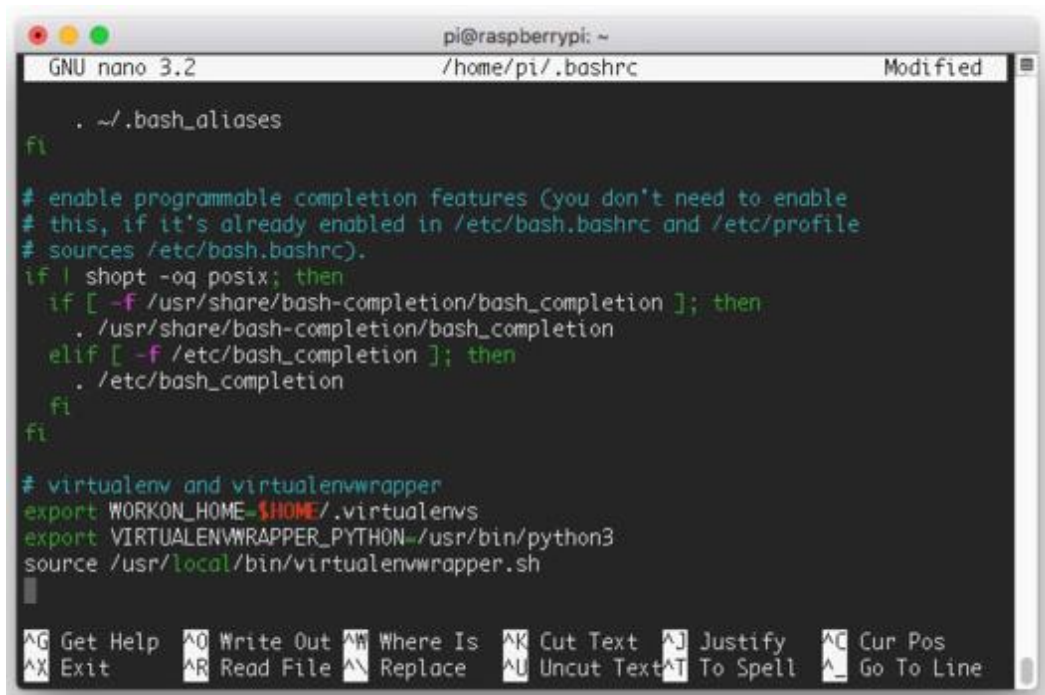


Figure 5: Using the `nano` editor to update `~/.bashrc` with `virtualenvwrapper` settings.

Save and exit via `ctrl + x`, `y`, and `enter`.

From there, reload your `~/.bashrc` file to apply changes to your current bash session

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ source ~/.bashrc
```

Next, create your **Python 3 virtual environment**:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ mkvirtualenv cv -p python3
```

Here we are creating a Python virtual environment named `cv` using Python 3. Going forward, I recommend Python 3 with OpenCV 4+.

Note: *Python 2.7 will reach end of its life on January 1st, 2020 so I **do not** recommend using Python 2.7.*

You can name the virtual environment whatever you want, but I use `cv` as the standard naming convention here on PyImageSearch.

If you have a Raspberry Pi Camera Module attached to your RPi, you should install the **PiCamera API** now as well:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ pip install "picamera[array]"
```

Check for correct paths for the picamera module

After this installation there should be a module called “picamera” stored within your `/home/pi/.virtualenvs/cv/lib/python3.7/site-packages` directory. Run the following commands to check.

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ cd /home/pi/.virtualenvs/cv/lib/python3.7/site-packages
```

```
$ ls
```

If the list DOES NOT contain blue text saying “picamera”, any attempts to use a picamera with your hardware within the virtual environment will fail. Copy the module into this directory and verify using the following commands.

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo cp -avr /usr/lib/python3/dist-packages/picamera /home/pi/.virtualenvs/cv/lib/python3.7/site-packages/
```

```
$ ls
```

Step #4: Time for the 2-hour complete install

This method gives you the **full install** of OpenCV 4. It will take 2-4 hours depending on the processor in your Raspberry Pi.

This option also includes patented (“Non-free”) algorithms.

Let’s go ahead and download the OpenCV source code for both the **opencv** and **opencv_contrib** repositories, followed by unarchiving them:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ cd ~
```

```
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/4.1.1.zip
```

```
$ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.1.1.zip
```

```
$ unzip opencv.zip
```

```
$ unzip opencv_contrib.zip
```

```
$ mv opencv-4.1.1 opencv
```

```
$ mv opencv_contrib-4.1.1 opencv_contrib
```

For this blog post, we’ll be using **OpenCV 4.1.1**; however, as newer versions of OpenCV are released you can update the corresponding version numbers.

Increasing your SWAP space

Before you start the compile you must **increase your SWAP space**. Increasing the SWAP will enable you to compile OpenCV with **all four cores** of the Raspberry Pi (*and* without the compile hanging due to memory exhausting).

Go ahead and open up your `/etc/dphys-swapfile` file:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo nano /etc/dphys-swapfile
```

...and then edit the `CONF_SWAPSIZE` variable:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
# set size to absolute value, leaving empty (default) then uses computed value
```

```
# you most likely don't want this, unless you have an special disk situation
```

```
# CONF_SWAPSIZE=100
```

```
CONF_SWAPSIZE=2048
```

Notice that I'm increasing the swap from 100MB to **2048MB**. This is critical to compiling OpenCV with multiple cores on Raspbian Buster.

2019-11-21 Update: Our testing has shown that a 2048MB swap is most effective to prevent lock-ups while OpenCV compiles.

Save and exit via `ctrl + x` , `y` , and enter

If you do not increase SWAP it's very likely that your Pi will hang during the compile.

From there, restart the swap service:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ sudo /etc/init.d/dphys-swapfile stop
```

```
$ sudo /etc/init.d/dphys-swapfile start
```

Note: Increasing swap size is a great way to burn out your Raspberry Pi microSD card. Flash-based storage has a limited number of writes you can perform until the card is essentially unable to hold the 1's and 0's anymore. We'll only be enabling large swap for a short period of time, so it's not a big deal. Regardless, be sure to backup your `.img` file after installing OpenCV + Python just in case your card dies unexpectedly early. You can read more about large swap sizes corrupting memory cards on [**this page**](#).

Compile and install OpenCV 4 on Raspbian Buster

We're now ready to compile and install the full, optimized OpenCV library on the Raspberry Pi 4.

Ensure you are in the `cv` virtual environment using the `workon` command:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ workon cv
```

Then, go ahead and install NumPy, imutils, and SciPy (OpenCV dependencies) into the Python virtual environment (these commands will take several minutes):

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ pip install numpy
```

```
$ pip install imutils
```

```
$ pip install scipy
```

And from there configure your build: here you tell CMake what, where, and how to make OpenCV on your Raspberry. There are many flags involved; let's save space by excluding any (Python and C) examples or tests. There are only bare spaces before the `-D` flags, not tabs. Also, keep a single space between the `-D` and the argument. Otherwise, CMake will misinterpret the command. The two last dots are no typo. It tells CMake where it can find its `CMakeLists.txt` (recipe); one directory up.

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ cd ~/opencv
```

```
$ mkdir build
```

```
$ cd build
```

```
$ $ cmake -D CMAKE_BUILD_TYPE=RELEASE \  
-D CMAKE_INSTALL_PREFIX=/usr/local \  
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \  
-D ENABLE_NEON=ON \  
-D ENABLE_VFPV3=ON \
```

```
-D WITH_OPENMP=ON \
-D BUILD_TIFF=ON \
-D WITH_FFMPEG=ON \
-D WITH_GSTREAMER=ON \
-D WITH_TBB=ON \
-D BUILD_TBB=ON \
-D BUILD_TESTS=OFF \
-D WITH_EIGEN=OFF \
-D WITH_V4L=ON \
-D WITH_LIBV4L=ON \
-D WITH_VTK=OFF \
-D OPENCV_EXTRA_EXE_LINKER_FLAGS=-latomic \
-D CMAKE_SHARED_LINKER_FLAGS=-latomic \
-D OPENCV_ENABLE_NONFREE=ON \
-D INSTALL_C_EXAMPLES=OFF \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D BUILD_NEW_PYTHON_SUPPORT=ON \
-D BUILD_opencv_python3=TRUE \
-D OPENCV_GENERATE_PKGCONFIG=ON \
-D BUILD_EXAMPLES=OFF ..
```

There are four CMake flags I'd like to bring to your attention:

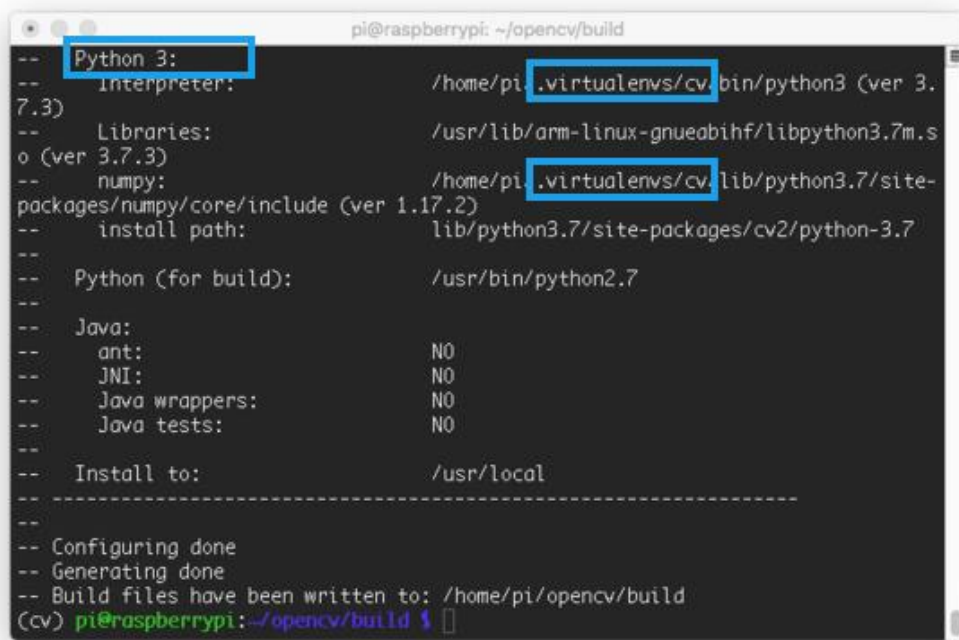
- **(1) NEON and (2) VFPv3** optimization flags have been enabled. These lines ensure that you compile the fastest and most optimized Deep Learning OpenCV for the ARM processor on the Raspberry Pi (**Lines 7 and 8**).
 - **Note:** The *Raspberry Pi Zero W* hardware is not compatible with NEON or VFPv3. Be sure to remove **Lines 7 and 8** if you are compiling for a Raspberry Pi Zero W.
- **(3)** Patented “NonFree” algorithms give you the full install of OpenCV (**Line 11**).
- And by drilling into OpenCV's source, it was determined that we need the **(4)** `-latomic` shared linker flag (**Line 12**).

I'd like to take a second now to bring awareness to a **common pitfall** for beginners:

- In the terminal block above, you must change directories into `~/opencv/`.
- You then **MUST** create a `build/` directory therein and change directories into it.
- **If you try to execute CMake without being in the `~/opencv/build` directory, CMake will fail.**
- As a sanity check, you can try running `pwd` to see which working directory you are in *before* running `cmake`.

The `cmake` command will take about 3-5 minutes to run as it prepares and configures OpenCV for the compile.

When CMake finishes, be sure to inspect the output of CMake under the Python 3 section:

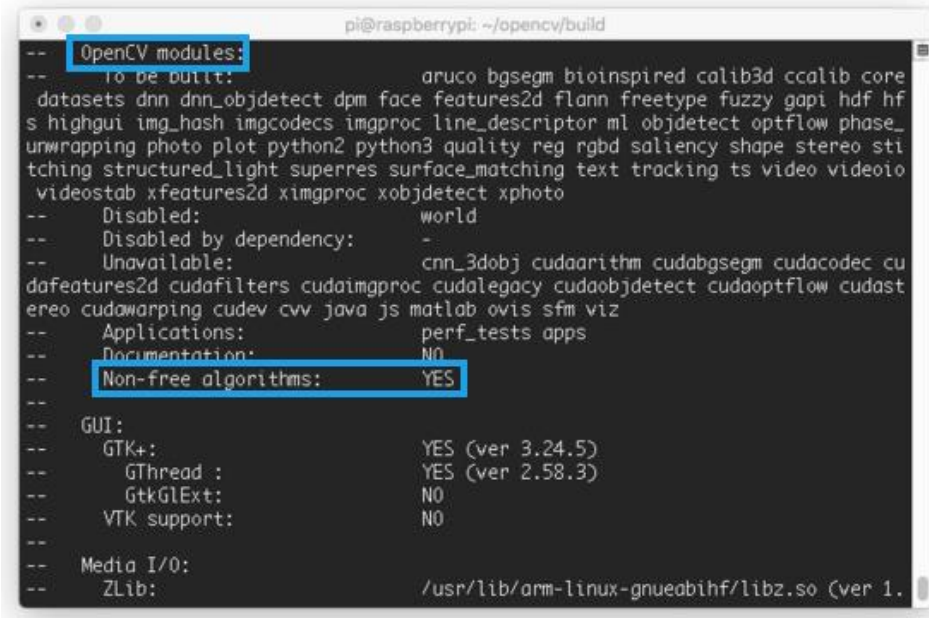
A terminal window on a Raspberry Pi 4 running Buster, showing the output of the CMake configuration process. The window title is 'pi@raspberrypi: ~/opencv/build'. The output is as follows:

```
-- Python 3:
-- Interpreter:      /home/pi/.virtualenvs/cv/bin/python3 (ver 3.
7.3)
-- Libraries:       /usr/lib/arm-linux-gnueabi/libpython3.7m.s
o (ver 3.7.3)
-- numpy:           /home/pi/.virtualenvs/cv/lib/python3.7/site-
packages/numpy/core/include (ver 1.17.2)
-- install path:    lib/python3.7/site-packages/cv2/python-3.7
--
-- Python (for build): /usr/bin/python2.7
--
-- Java:
--   ant:            NO
--   JNI:            NO
--   Java wrappers:  NO
--   Java tests:     NO
--
-- Install to:      /usr/local
-----
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/opencv/build
(cv) pi@raspberrypi:~/opencv/build $
```

Figure 6: CMake configures your OpenCV 4 compilation from source on your Raspberry Pi 4 running Buster.

Notice how the `Interpreter`, `Libraries`, `numpy`, and `packages` path variables have been properly set. Each of these refers to our `cv` virtual environment.

Now go ahead and **scroll up** to ensure that the “**Non-Free algorithms**” are set to be installed:



```
pi@raspberrypi: ~/opencv/build
-- OpenCV modules:
--   To be built: aruco bgsegm bioinspired calib3d ccalib core
datasets dnn dnn_objdetect dpm face features2d flann freetype fuzzy gapi hdf hf
s highgui img_hash imgcodecs imgproc line_descriptor ml_objdetect optflow phase_
unwrapping photo plot python2 python3 quality reg rgbd saliency shape stereo sti
tching structured_light superres surface_matching text tracking ts video videoio
videostab xfeatures2d ximgproc xobjdetect xphoto
--   Disabled: world
--   Disabled by dependency: -
--   Unavailable: cnn_3dobj cudaarithm cudabgsegm cudacodec cu
dafeatures2d cudafilters cudaimgproc cudalegacy cudaobjdetect cudaoptflow cudast
ereo cudawarping cudev cvv java js matlab ovis sfm viz
--   Applications: perf_tests apps
--   Documentation: NO
-- Non-free algorithms: YES
--
GUI:
--   GTK+: YES (ver 3.24.5)
--   GThread : YES (ver 2.58.3)
--   GtkGLExt: NO
--   VTK support: NO
--
Media I/O:
--   ZLib: /usr/lib/arm-linux-gnueabi/libz.so (ver 1.
```

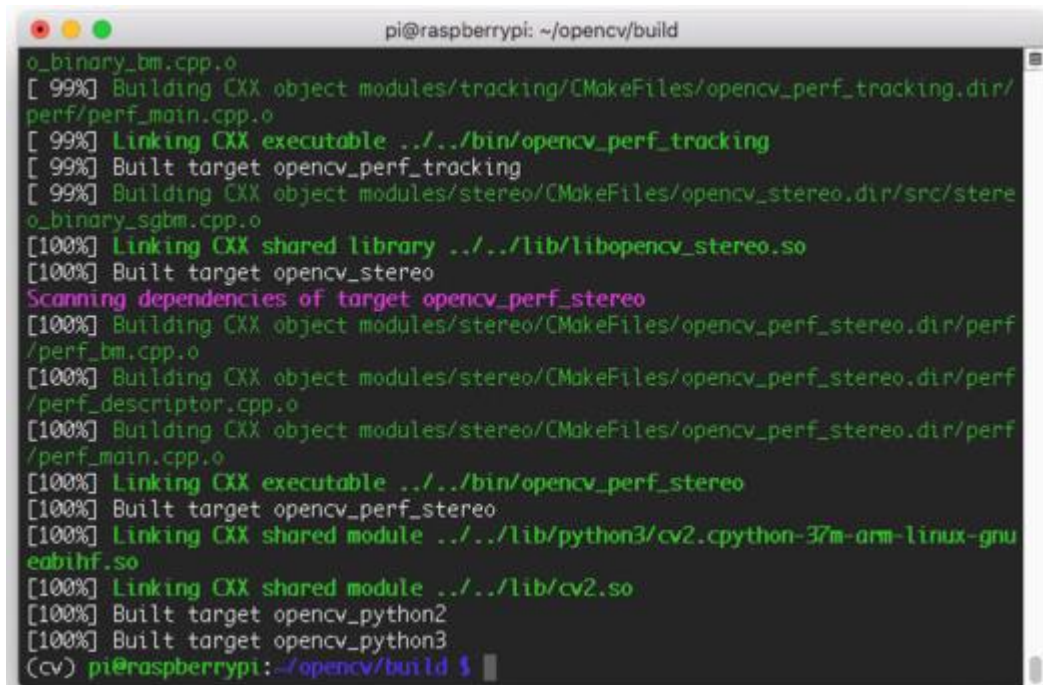
Figure 7: Installing OpenCV 4 with “Non-free algorithms” on Raspbian Buster.

As you can see, “Non-free algorithms” for OpenCV 4 will be compiled + installed.

Now that we’ve prepared for our OpenCV 4 compilation, it is time to launch the compile process using all four cores:

Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster

```
$ make -j4
```


A terminal window titled 'pi@raspberrypi: ~/opencv/build' showing the progress of compiling OpenCV 4. The output includes: 'o_binary_bm.cpp.o', '[99%] Building CXX object modules/tracking/CMakeFiles/opencv_perf_tracking.dir/perf/perf_main.cpp.o', '[99%] Linking CXX executable ../../bin/opencv_perf_tracking', '[99%] Built target opencv_perf_tracking', '[99%] Building CXX object modules/stereo/CMakeFiles/opencv_stereo.dir/src/stereo_binary_sgbm.cpp.o', '[100%] Linking CXX shared library ../../lib/libopencv_stereo.so', '[100%] Built target opencv_stereo', 'Scanning dependencies of target opencv_perf_stereo', '[100%] Building CXX object modules/stereo/CMakeFiles/opencv_perf_stereo.dir/perf/perf_bm.cpp.o', '[100%] Building CXX object modules/stereo/CMakeFiles/opencv_perf_stereo.dir/perf/perf_descriptor.cpp.o', '[100%] Building CXX object modules/stereo/CMakeFiles/opencv_perf_stereo.dir/perf/perf_main.cpp.o', '[100%] Linking CXX executable ../../bin/opencv_perf_stereo', '[100%] Built target opencv_perf_stereo', '[100%] Linking CXX shared module ../../lib/python3/cv2.cpython-37m-arm-linux-gnueabi.so', '[100%] Linking CXX shared module ../../lib/cv2.so', '[100%] Built target opencv_python2', '[100%] Built target opencv_python3', and '(cv) pi@raspberrypi:~/opencv/build \$'.

```
pi@raspberrypi: ~/opencv/build
o_binary_bm.cpp.o
[ 99%] Building CXX object modules/tracking/CMakeFiles/opencv_perf_tracking.dir/perf/perf_main.cpp.o
[ 99%] Linking CXX executable ../../bin/opencv_perf_tracking
[ 99%] Built target opencv_perf_tracking
[ 99%] Building CXX object modules/stereo/CMakeFiles/opencv_stereo.dir/src/stereo_binary_sgbm.cpp.o
[100%] Linking CXX shared library ../../lib/libopencv_stereo.so
[100%] Built target opencv_stereo
Scanning dependencies of target opencv_perf_stereo
[100%] Building CXX object modules/stereo/CMakeFiles/opencv_perf_stereo.dir/perf/perf_bm.cpp.o
[100%] Building CXX object modules/stereo/CMakeFiles/opencv_perf_stereo.dir/perf/perf_descriptor.cpp.o
[100%] Building CXX object modules/stereo/CMakeFiles/opencv_perf_stereo.dir/perf/perf_main.cpp.o
[100%] Linking CXX executable ../../bin/opencv_perf_stereo
[100%] Built target opencv_perf_stereo
[100%] Linking CXX shared module ../../lib/python3/cv2.cpython-37m-arm-linux-gnueabi.so
[100%] Linking CXX shared module ../../lib/cv2.so
[100%] Built target opencv_python2
[100%] Built target opencv_python3
(cv) pi@raspberrypi:~/opencv/build $
```

Figure 8: We used Make to compile OpenCV 4 on a Raspberry Pi 4 running Raspbian Buster.

Running `make` could take anywhere from 1-4 hours depending on your Raspberry Pi hardware (this tutorial is compatible with the Raspberry Pi 3B, 3B+, and 4). The Raspberry Pi 4 is the fastest at the time of this writing.

Assuming OpenCV compiled without error (as in my screenshot above), you can install your optimized version of OpenCV on your Raspberry Pi:

Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster

```
$ sudo make install
```

```
$ sudo ldconfig
```

Reset your SWAP

Re-open your `/etc/dphys-swapfile` file and:

1. Reset `CONF_SWAPSIZE` to 100MB.

2. Restart the swap service.

Sym-link your OpenCV 4 on the Raspberry Pi

Symbolic links are a way of pointing from one directory to a file or folder elsewhere on your system. For this sub-step, we will sym-link the `cv2.so` bindings into your `cv` virtual environment.

Let's proceed to create our sym-link. Be sure to **use “tab-completion”** for all paths below (rather than copying these commands blindly):

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ cd /usr/local/lib/python3.7/site-packages/cv2/python-3.7
```

```
$ sudo mv cv2.cpython-37m-arm-linux-gnueabi.so cv2.so
```

```
$ cd ~/.virtualenvs/cv/lib/python3.7/site-packages/
```

```
$ ln -s /usr/local/lib/python3.7/site-packages/cv2/python-3.7/cv2.so cv2.so
```

Keep in mind that the exact paths may change and you should **use “tab-completion”**.

Step 5: Testing your OpenCV 4 Raspberry Pi BusterOS install

As a quick sanity check, access the `cv` virtual environment, fire up a Python shell, and try to import the OpenCV library:

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ cd ~
```

```
$ workon cv
```

```
$ python
```

```
>>> import cv2
```

```
>>> cv2.__version__
```

```
'4.1.1'
```

```
>>>quit()
```

If all went well, the following commands should also work outside your virtual environment.

Congratulations! You've just installed an OpenCV 4 on your Raspberry Pi.

For developing Python code on the Pi, I recommend using the default Thonny IDE. For C++ applications, a good option is “codeblocks”, which can be installed with the following terminal command:

```
$ sudo apt-get install -y codeblocks
```

Note the following command for compiling and linking opencv c++ files via terminal:

```
$ g++ $(pkg-config --cflags --libs opencv4) -std=c++11 <source> -o <target executable>
```

Be sure to insert the name of your c++ source file in the place of <source file>, and whatever you want your executable to be called in the place of <target executable>; omitting the “<>” symbols

An alternative is to create a CMakeLists.txt file in your project folder by navigating to the folder where you want your project and typing:

```
$ sudo nano CMakeLists.txt
```

And adding this to the empty file it generates:

```
cmake_minimum_required(VERSION 3.8)
```

```
project(opencvProj)
```

```
find_package(OpenCV REQUIRED)
```

```
add_executable(mainRun main.cpp)
```

```
target_link_libraries(mainRun ${OpenCV_LIBS})
```

Note for this CMakeLists.txt, main.cpp is the c++ file you'll create with the code you want to run (source), and mainRun is the name of the executable you want to generate.

Compiling and linking can be done within the folder containing your source (main.cpp) file via the following commands

```
Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster
```

```
$ cmake
```

```
$ make
```

References

- [1] A. Rosebrock, "Install OpenCV 4 on Rasbian Buster," Pyimagesearch, 16 September 2019. [Online]. Available: <https://www.pyimagesearch.com/2019/09/16/install-opencv-4-on-raspberry-pi-4-and-raspbian-buster/>. [Accessed 27 July 2020].
- [2] "Install OpenCV 4.3.0 on Raspberry Pi 4," Q-Engineering, [Online]. Available: <https://qengineering.eu/install-opencv-4.3-on-raspberry-pi-4.html>. [Accessed 27 July 2020].