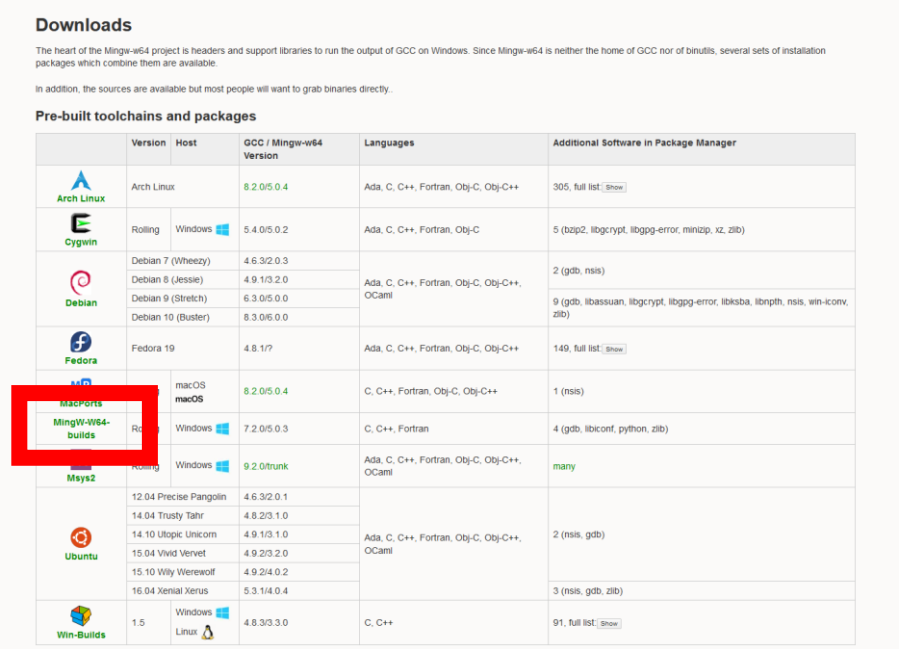
Préparation des outils

1 – Télécharger et installer **Microsoft Visual Studio**

2 – Installer **CMake**:

Télécharger et installer le depuis le site officiel (Pour les utilisateurs de Windows seulement)



Downloads

The heart of the Mingw-w64 project is headers and support libraries to run the output of GCC on Windows. Since Mingw-w64 is neither the home of GCC nor of binutils, several sets of installation packages which combine them are available.

In addition, the sources are available but most people will want to grab binaries directly.

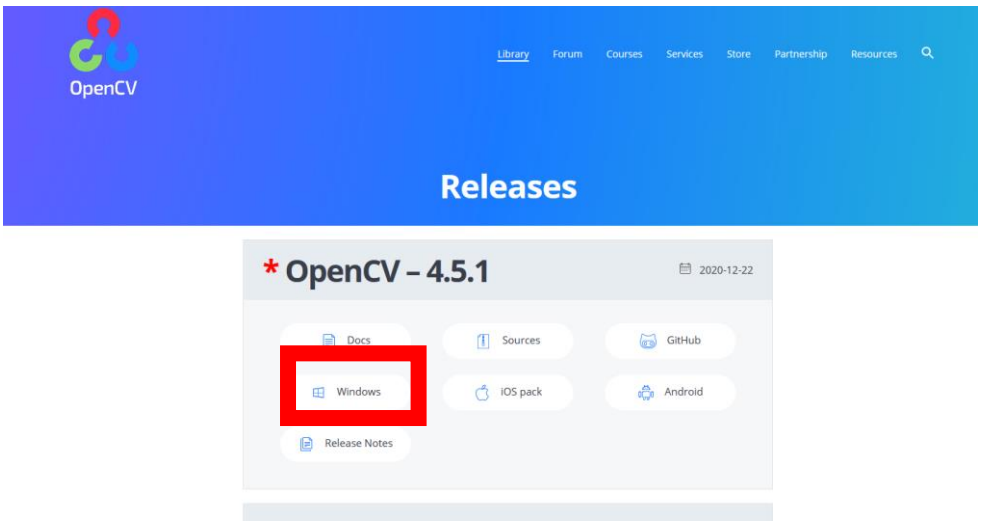
Pre-built toolchains and packages

	Version	Host	GCC / Mingw-w64 Version	Languages	Additional Software in Package Manager
Arch Linux	Arch Linux		8.2.0/5.0.4	Ada, C, C++, Fortran, Obj-C, Obj-C++	305, full list: Show
Cygwin	Rolling	Windows	5.4.0/5.0.2	Ada, C, C++, Fortran, Obj-C	5 (bzip2, libgcrypt, libgpg-error, minizip, xz, zlib)
Debian	Debian 7 (Wheezy)		4.6.3/2.0.3	Ada, C, C++, Fortran, Obj-C, Obj-C++, OCaml	2 (gdb, nsis)
	Debian 8 (Jessie)		4.9.1/3.2.0		
	Debian 9 (Stretch)		6.3.0/5.0.0		9 (gdb, libassuan, libgcrypt, libgpg-error, libksba, libnpth, nsis, win-iconv, zlib)
	Debian 10 (Buster)		8.3.0/6.0.0		
Fedora	Fedora 19		4.8.1/?	Ada, C, C++, Fortran, Obj-C, Obj-C++	149, full list: Show
MacPorts		macOS **macOS**	8.2.0/5.0.4	C, C++, Fortran, Obj-C, Obj-C++	1 (nsis)
MingW-W64-builds	Rolling	Windows	7.2.0/5.0.3	C, C++, Fortran	4 (gdb, libiconf, python, zlib)
Msys2	Rolling	Windows	9.2.0/trunk	Ada, C, C++, Fortran, Obj-C, Obj-C++, OCaml	many
Ubuntu	12.04 Precise Pangolin		4.6.3/2.0.1	Ada, C, C++, Fortran, Obj-C, Obj-C++, OCaml	2 (nsis, gdb)
	14.04 Trusty Tahr		4.8.2/3.1.0		
	14.10 Utopic Unicorn		4.9.1/3.1.0		
	15.04 Vivid Vervet		4.9.2/3.2.0		
	15.10 Wily Werewolf		4.9.2/4.0.2		
	16.04 Xenial Xerus		5.3.1/4.0.4		3 (nsis, gdb, zlib)
Win-Builds	1.5	Windows Linux	4.8.3/3.3.0	C, C++	91, full list: Show

Ajouter le chemin de **CMake** aux variables d'environnement

3 – Installer **OpenCV :**

[Windows] : vous n'avez qu'à télécharger le projet



[Linux et Mac] : vous devez télécharger le code source et le compiler sur vos machines

Installation des dépendances

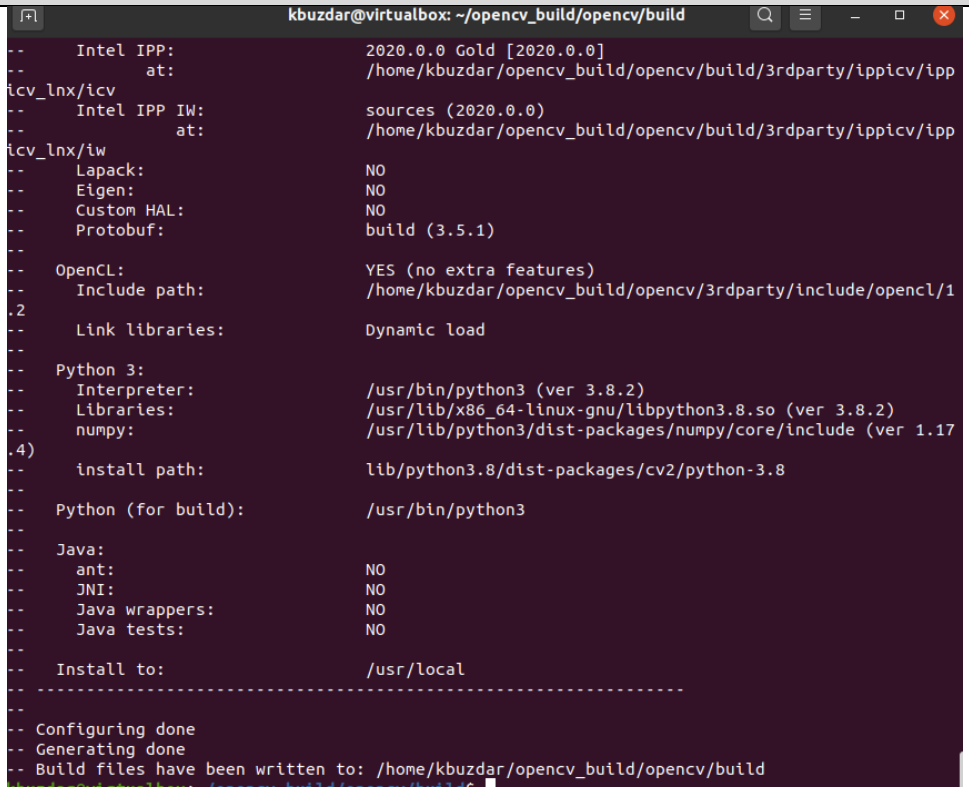
```
sudo apt install build-essential cmake git pkg-config libgtk-3-dev \
      libavcodec-dev libavformat-dev libswscale-dev libv4l-dev \
      libxvidcore-dev libx264-dev libjpeg-dev libpng-dev libtiff-dev \
      gfortran openexr libatlas-base-dev python3-dev python3-numpy \
      libtbb2 libtbb-dev libdc1394-22-dev libopenexr-dev \
      libgstreamer-plugins-base1.0-dev libgstreamer1.0-dev
```

On commence par télécharger OpenCV et ces examples

```
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
```

Maintenant, on compile le projet en fixant certaines configurations. Pour cela, on utilise les commandes suivantes. Mais il vous reste à fixer le chemin vers le dossier **opencv_contrib.**

```
cd opencv
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
      -D CMAKE_INSTALL_PREFIX=/usr/local \
      -D INSTALL_C_EXAMPLES=ON \
      -D INSTALL_PYTHON_EXAMPLES=ON \
      -D OPENCV_GENERATE_PKGCONFIG=ON \
      -D
      OPENCV_EXTRA_MODULES_PATH=/<Chemin_complet>/openc/vopencv_contrib/
      modules \
      -D BUILD_EXAMPLES=ON ..
```



Start a compilation

```
make -j8 <- le 8 signifi le nombre de core à utiliser
```

```
kbuzdar@virtualbox: ~/opencv_build/opencv/build
kbuzdar@virtualbox:~/opencv_build/opencv/build$ make -j8
Scanning dependencies of target gen-pkgconfig
Scanning dependencies of target quirc
Scanning dependencies of target ittnotify
Scanning dependencies of target ippiw
Scanning dependencies of target libjasper
Scanning dependencies of target ade
Scanning dependencies of target libwebp
Scanning dependencies of target libprotobuf
[  0%] Generate opencv4.pc
[  0%] Building C object 3rdparty/quirc/CMakeFiles/quirc.dir/src/decode.c.o
[  0%] Building C object 3rdparty/ittnotify/CMakeFiles/ittnotify.dir/src/ittnotify/ittnotify_st
atic.c.o
[  0%] Building CXX object modules/CMakeFiles/ade.dir/__/3rdparty/ade/ade-0.1.1f/sources/ade/so
urce/alloc.cpp.o
[  0%] Building C object 3rdparty/ippiw/CMakeFiles/ippiw.dir/src/iw_core.c.o
[  0%] Built target gen-pkgconfig
[  0%] Building C object 3rdparty/libjasper/CMakeFiles/libjasper.dir/jas_cm.c.o
Scanning dependencies of target opencv_videoio_plugins
[  0%] Built target opencv_videoio_plugins
Scanning dependencies of target gen_opencv_python_source
[  0%] Building C object 3rdparty/libwebp/CMakeFiles/libwebp.dir/src/dec/alpha_dec.c.o
[  0%] Generate files for Python bindings and documentation
[  0%] Building CXX object 3rdparty/protobuf/CMakeFiles/libprotobuf.dir/src/google/protobuf/are
na.cc.o
[  0%] Building C object 3rdparty/ippiw/CMakeFiles/ippiw.dir/src/iw_image.c.o
[  0%] Building C object 3rdparty/libwebp/CMakeFiles/libwebp.dir/src/dec/buffer_dec.c.o
[  0%] Building CXX object modules/CMakeFiles/ade.dir/__/3rdparty/ade/ade-0.1.1f/sources/ade/so
urce/assert.cpp.o
[  0%] Building CXX object modules/CMakeFiles/ade.dir/__/3rdparty/ade/ade-0.1.1f/sources/ade/so
urce/check_cycles.cpp.o
[  0%] Building C object 3rdparty/libwebp/CMakeFiles/libwebp.dir/src/dec/frame_dec.c.o
Note: Class Feature2D has more than 1 base class (not supported by Python C extensions)
      Bases:  cv::Algorithm, cv::class, cv::Feature2D, cv::Algorithm
      Only the first base class will be used
[  0%] Building C object 3rdparty/libjasper/CMakeFiles/libjasper.dir/jas_debug.c.o
[  0%] Building C object 3rdparty/ippiw/CMakeFiles/ippiw.dir/src/iw_image_color_convert_all.c.o
[  0%] Building C object 3rdparty/libwebp/CMakeFiles/libwebp.dir/src/dec/idec_dec.c.o
```

Installation d'OpenCV

sudo make install

```
kbuzdar@virtualbox:~/opencv_build/opencv/build$ sudo make install
[sudo] password for kbuzdar:
[  0%] Built target gen-pkgconfig
[  4%] Built target libwebp
[  5%] Built target libjasper
[  6%] Built target ippiw
[  9%] Built target libprotobuf
[ 10%] Built target quirc
[ 10%] Built target ittnotify
[ 10%] Built target ade
[ 10%] Built target opencv_videoio_plugins
[ 14%] Built target opencv_core
[ 18%] Built target opencv_imgproc
[ 18%] Built target opencv_imgcodecs
[ 19%] Built target opencv_videoio
[ 19%] Built target opencv_highgui
[ 20%] Built target opencv_ts
[ 22%] Built target opencv_perf_core
[ 23%] Built target opencv_test_core
[ 23%] Built target opencv_flann
[ 23%] Built target opencv_test_flann
[ 24%] Built target opencv_perf_imgproc
[ 26%] Built target opencv_test_imgproc
[ 26%] Built target opencv_intensity_transform
[ 26%] Built target example_intensity_transform_intensity_transform
[ 26%] Built target opencv_test_intensity_transform
[ 26%] Built target opencv_ml
[ 27%] Built target opencv_test_ml
[ 27%] Built target opencv_phase_unwrapping
[ 27%] Built target example_phase_unwrapping_unwrap
[ 27%] Built target opencv_test_phase_unwrapping
[ 27%] Built target opencv_photo
[ 27%] Built target opencv_perf_photo
[ 28%] Built target opencv_test_photo
[ 28%] Built target opencv_plot
[ 28%] Built target example_plot_plot_demo
[ 29%] Built target opencv_quality
```

Afin de vérifier que vous avez bien installé OpenCV, vous faite exécuter la commandes suivantes :

```
Pkg-config --libs --cflags opencv4
```

Création d'un projet

1 – Créer un dossier et nommez le, par exemple : TP<X> où vous remplacer le X par le numéro du TP

2 – Créer 2 fichiers : **main.cpp** et **CMakeLists.txt**

```
#include <stdio.h>
#include <opencv2/opencv.hpp>
using namespace cv;
int main()
{
   Mat image;
   image = Mat::zeros(512, 512, CV_8UC1);
   if ( image.empty() )
   {
       printf("No image data \n");
       return -1;
   }
   namedWindow("Display Image", WINDOW_AUTOSIZE );
   imshow("Display Image", image);
   waitKey(0);
   return 0;
}
```

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

Avant de compiler le projet, créez un dossier **build** et déplacez vous vers lui.

```
mkdir build && cd build
cmake ..
make
./DisplayImage
```