Computer Vision, Lab 01: OpenCV and Octave

Today we'll learn how to perform some useful computer vision tasks with OpenCV (C++ and Python) and Octave.

Install OpenCV

In this lab, we give you multiple options for installing OpenCV and a C++/Python IDE.

Different projects will typically use different versions of OpenCV. On Ubuntu, you will therefore generally want to install each version you use in `/usr/local` and pick the precise version you need for a specific project at build time. I'm not sure what you should do to maintain different versions on Windows!

If you haven't already, go ahead and install OpenCV 4.5.5 according to the instructions below. It will take 20 minutes plus. If you're on Windows, skip to the Windows intructions, where you'll have a choice between Visual Studio Community and Visual Studio Code.

Apple Mac people!

We don't provide instructions here. Please see the instructions here (https://thecodinginterface.com/blog/opencv-cpp-vscode/).

Linux

Dependencies for Ubuntu 20.04

I'm not sure these are all the dependencies you will need. Let me know if some are missing!

```
$ sudo apt-get install cmake build-essential git unzip python3 libjpeg-dev libpng-dev \
    python3-pip wget libtbb2 libtbb-dev libtiff-dev libqt5opengl5-dev libqt5gui5 libqwt-qt5
-dev \
    libqt5test5 libqt5concurrent5 libopenblas-dev libopenblas64-dev liblapack-dev liblapack
64-dev \
    liblapacke-dev libeigen3-dev libavfilter-dev libvtk6-dev libwebp-dev libopenexr-dev lib
gdal-dev \
    libavutil-dev libavcodec-dev libavformat-dev libswscale-dev libavresample-dev libavdevi
ce-dev \
    libatlas-base-dev libvtk7-dev libv4l-dev libeigen3-dev libatlas-base-dev libatlas-cpp-
0.6-dev \
    libgstreamer1.0-dev gstreamer1.0-plugins-{base,good,bad,ugly}
```

Download

The current stable version of OpenCV is 4.5.5. Download the source from the OpenCV GitHub page (https://github.com/opencv/opencv/archive/4.5.5.zip) and unpack it somewhere:

```
$ cd ~/Desktop
    $ wget https://github.com/opencv/opencv/archive/4.5.5.zip
$ unzip 4.5.5.zip
    $ rm 4.5.5.zip
$ wget https://github.com/opencv/opencv_contrib/archive/4.5.5.zip
    $ unzip 4.5.5.zip
$ rm 4.5.5.zip
    $ cd opencv-4.5.5
$ mkdir build
    $ cd build
```

Build

I like to build and install the library in such a way that each project can point to its preferred version of the library. I use the following build definitions:

```
$ cmake -Wno-dev \
        -DCMAKE_BUILD_TYPE=RELEASE \
        -DOPENCV_EXTRA_MODULES_PATH=$HOME/Desktop/opencv_contrib-4.5.5/modules \
    -DCMAKE_INSTALL_PREFIX=/usr/local/opencv-4.5.5 \
    -DBUILD_opencv_java=OFF -DWITH_OPENGL=ON \
    -DWITH_OPENCL=ON -DWITH_IPP=OFF -DFORCE_VTK=ON \
    -DWITH_GDAL=ON -DWITH_XINE=ON \
    -DENABLE_PRECOMPILED_HEADERS=OFF \
    -DENABLE_AVX=ON -DWITH_TBB=ON -DWITH_EIGEN=ON \
    -DWITH_V4L=ON -DWITH_LAPACK=ON -DWITH_QT=ON \
    -DWITH_FFMPEG=ON -DBUILD_TESTS=OFF \
    -DBUILD_PERF_TESTS=OFF -DOPENCV_ENABLE_NONFREE=ON ..
$ # Check that the dependencies you want are found by cmake
    $ make -j8
$ sudo make install
```

This gives us QT windows with OpenGL support, as well as non-free modules like SIFT. You'll have to do more work to get CUDA support (if you have an Nvidia graphics card).

Sit back and relax while it compiles. You can install VSCode while OpenCV is building (see below)

Windows Method 1 (Visual Studio C++)

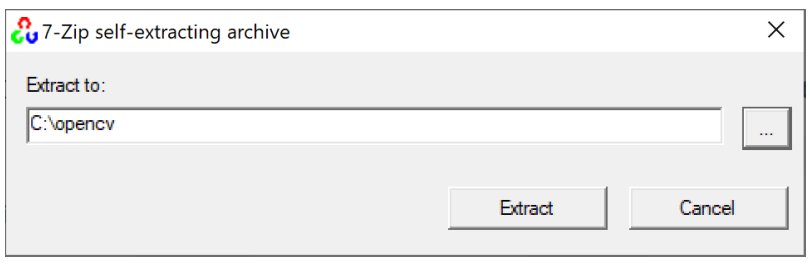
These instructions were written by Alisa Kunapinun.

In Windows, we have many options for how to install. Here we give some pointers for Visual Studio Community on Windows, for C++ and Python. Later, we'll give instructions for Visual Studio Code on Windows.

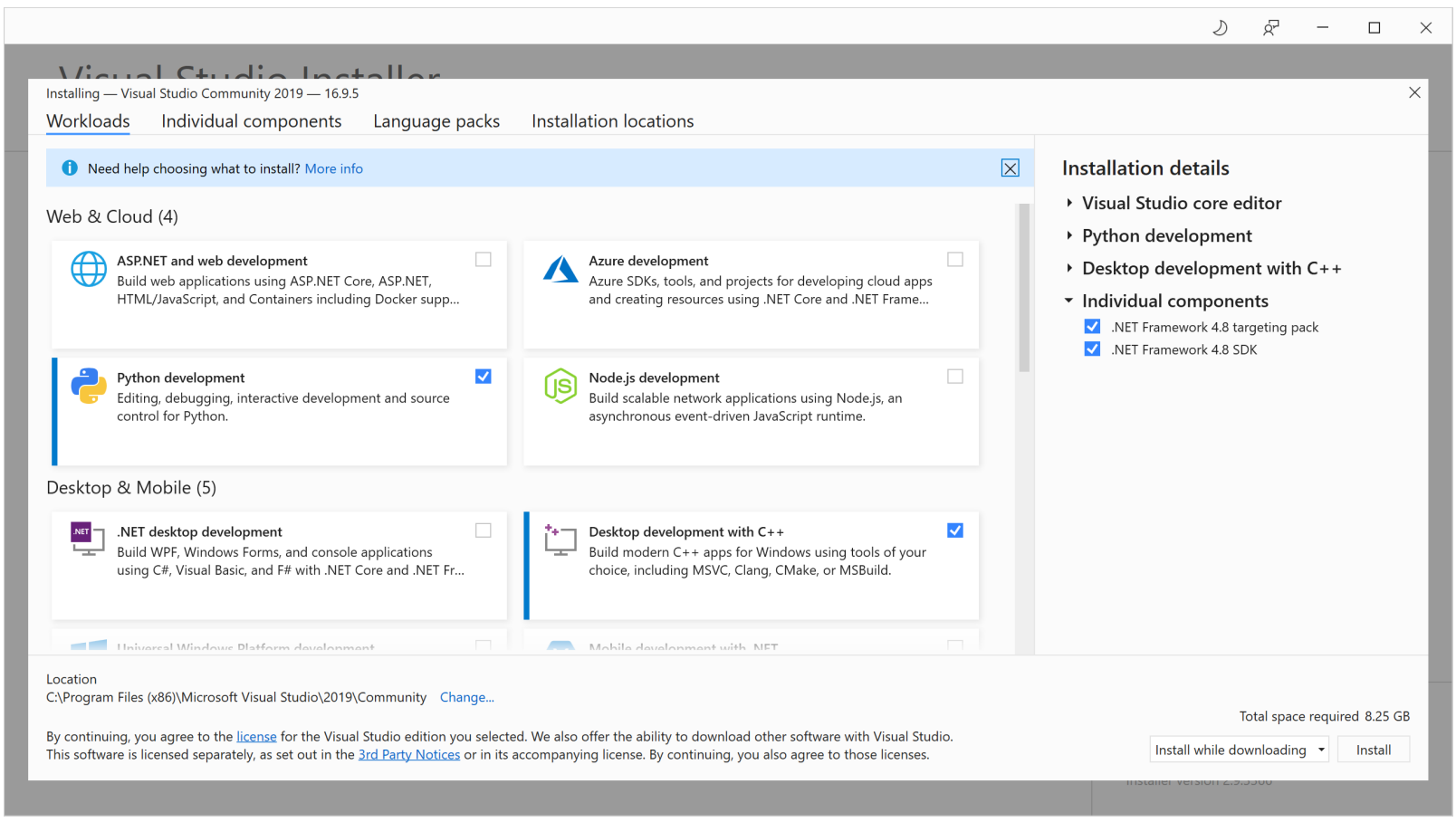
Here are some references:

- https://demystifymachinelearning.wordpress.com/2018/08/30/installing-opencv-on-windows-using-anaconda/ (https://demystifymachinelearning.wordpress.com/2018/08/30/installing-opencv-on-windows-using-anaconda/)
- https://cv-tricks.com/how-to/installation-of-opencv-4-1-0-in-windows-10-from-source/ (https://cv-tricks.com/how-to/installation-of-opencv-4-1-0-in-windows-10-from-source/)
- https://docs.opencv.org/master/d5/de5/tutorial_py_setup_in_windows.html (https://docs.opencv.org/master/d5/de5/tutorial_py_setup_in_windows.html)
- https://medium.com/@kswalawage/install-python-and-jupyter-notebook-to-windows-10-64-bit-66db782e1d02 (https://medium.com/@kswalawage/install-python-and-jupyter-notebook-to-windows-10-64-bit-66db782e1d02)
- https://cuda-chen.github.io/programming/image%20processing/2020/01/21/vscode-with-opencv-cpp-on-windows10-explained.html (https://cuda-chen.github.io/programming/image%20processing/2020/01/21/vscode-with-opencv-cpp-on-windows10-explained.html)
- https://www.pranav.ai/cplusplus-for-jupyter (https://www.pranav.ai/cplusplus-for-jupyter)
- https://learnopencv.com/tag/xeus-cling/ (https://learnopencv.com/tag/xeus-cling/)
- https://sourceforge.net/projects/opencvlibrary/files/opencv-win/ (https://sourceforge.net/projects/opencvlibrary/files/opencv-win/)
- https://www.youtube.com/watch?v=FCzMpHWUUKg (https://www.youtube.com/watch?v=FCzMpHWUUKg)

1. Download latest OpenCV release from **OpenCV site (https://opencv.org/releases/)** site
    - Click on the **Windows** of the latest version on openCV.
    - Extract it to the desire location: In this step, I choose `C:\opencv`
2. Clone OpenCV-Contribute at **GitHub (https://github.com/opencv/opencv_contrib.git)**, and extract it at the same location.



3. Install Visual Studio Community from **here (https://visualstudio.microsoft.com/vs/community/)**
4. While setting up visual studio installation, select 2 workloads as below:
    - Python Development
    - Desktop Development with C++

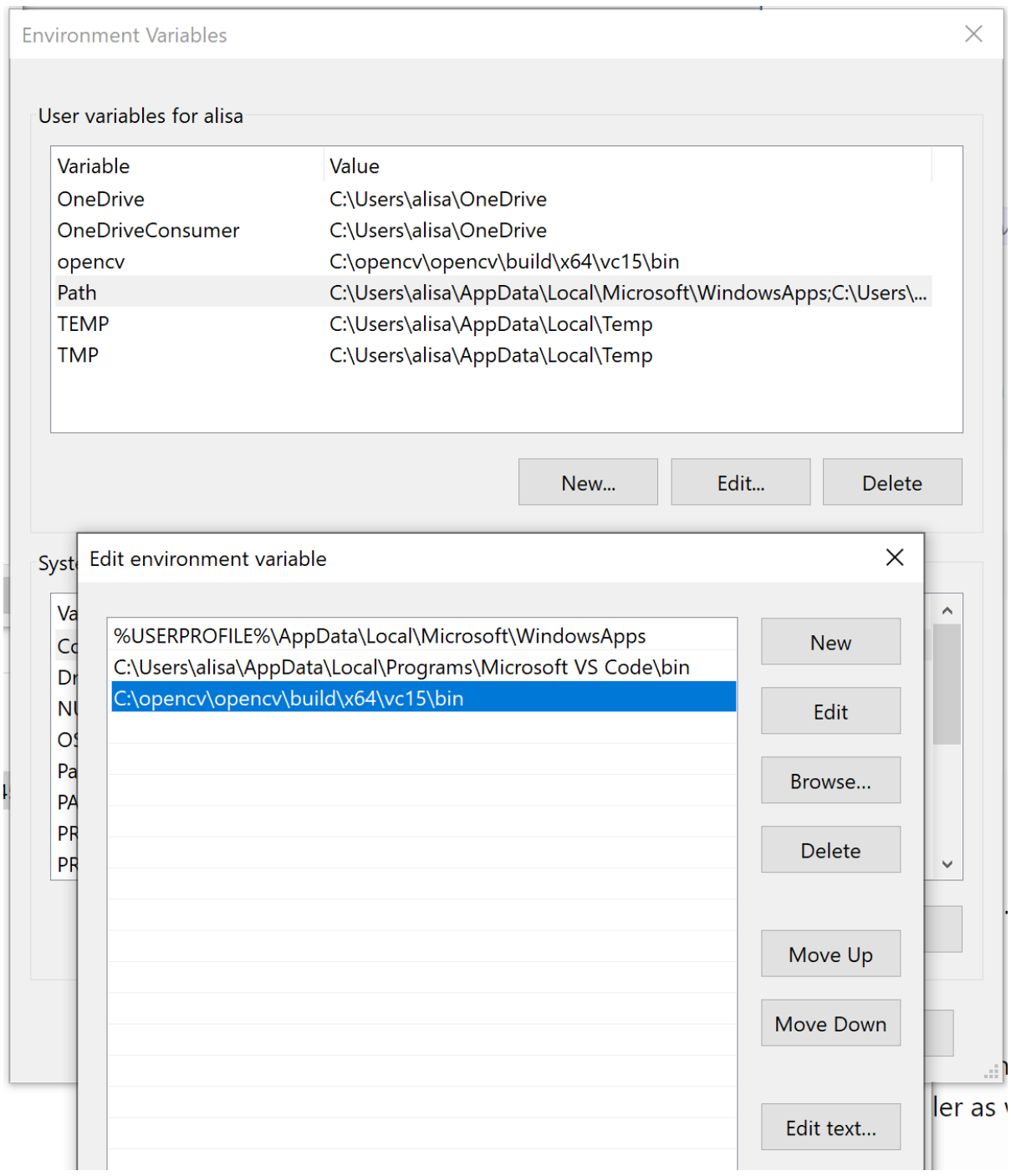


5. Go to System Properties → *Environment Variables*

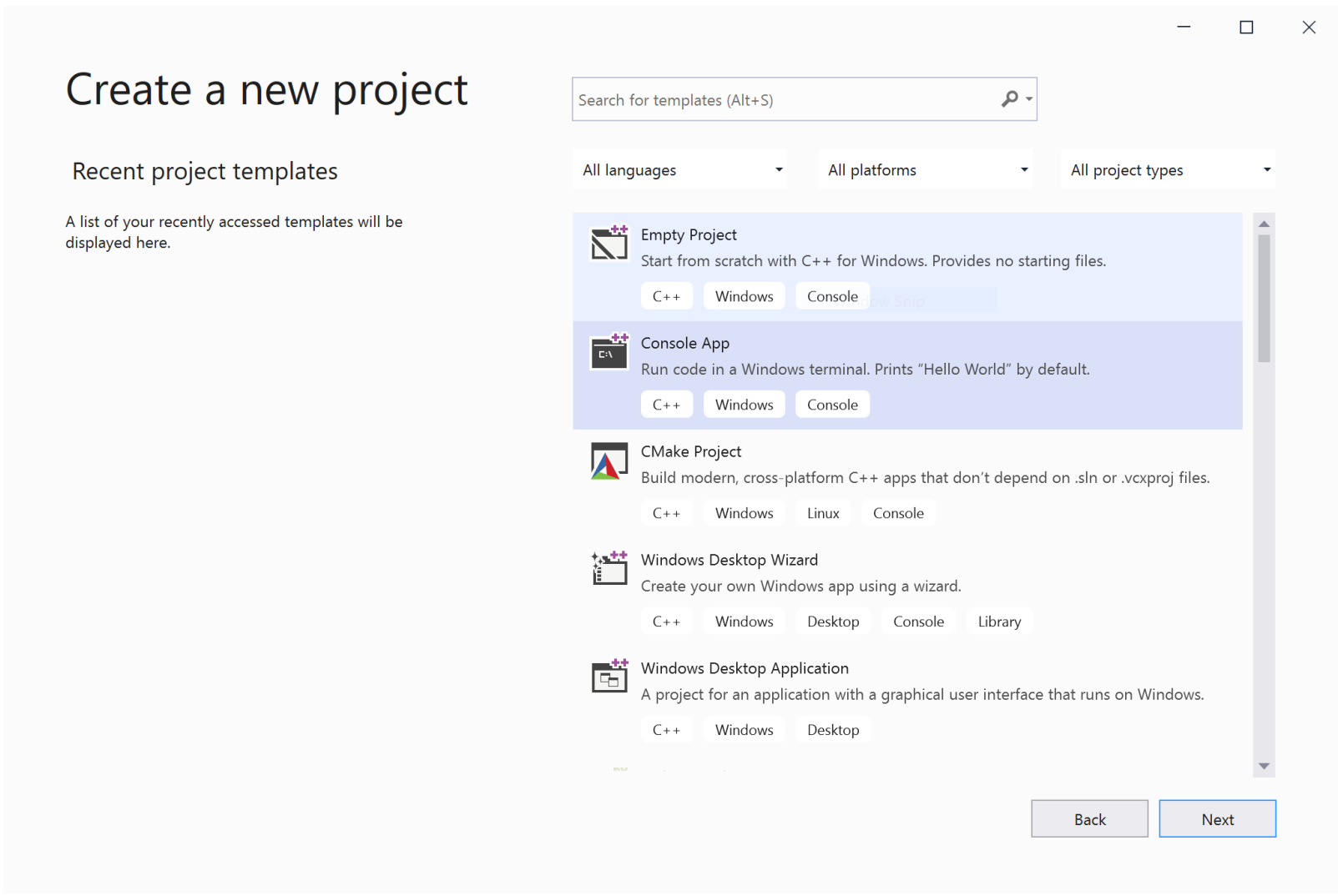
    Note: In Windows10, you can go type "Environment Variables" to search System Properties

    - At the User variables → Path tab select → *Edit..*
    - Press *New* and input

- Variable value: `C:\opencv\opencv\build\x64\vc15\bin` (Your location)

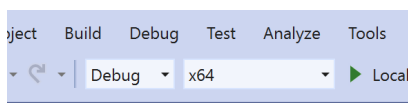


7. After installation done, the Visual Studio require restart. After restart, open Visual studio
8. Create a new project as **Console App** (C++)

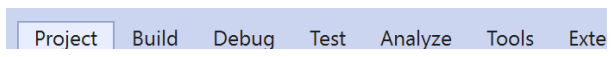


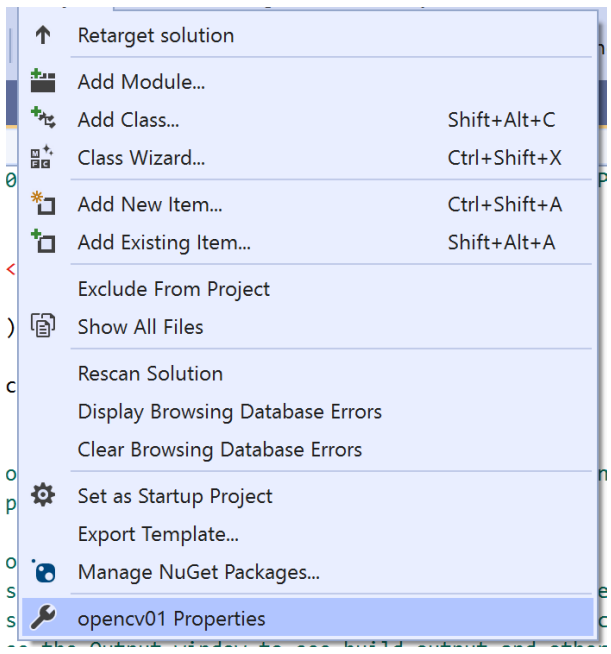
Setting up OpenCV in Visual Studio C++

1. Set platform target to x64

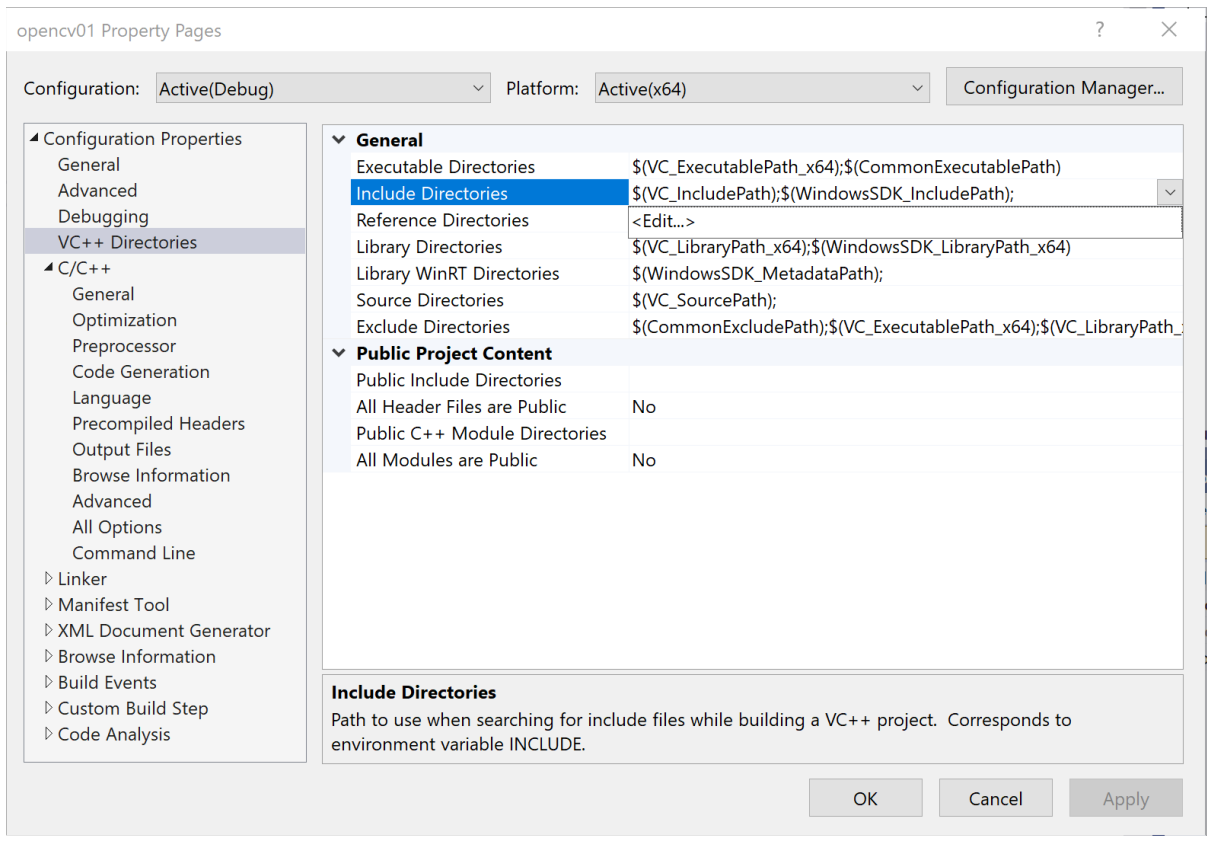


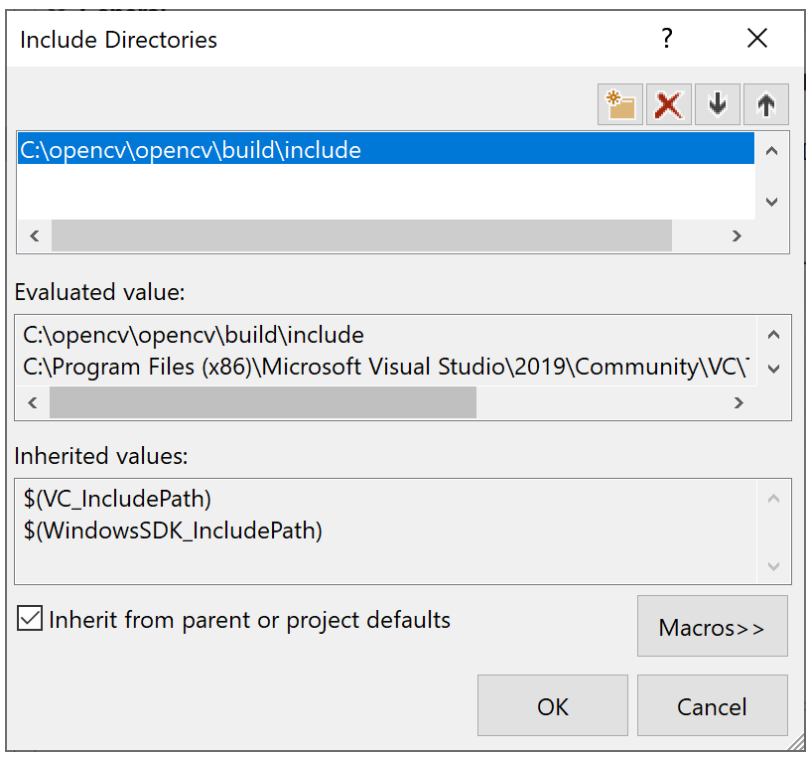
2. Go to *Project* → [project name] Properties



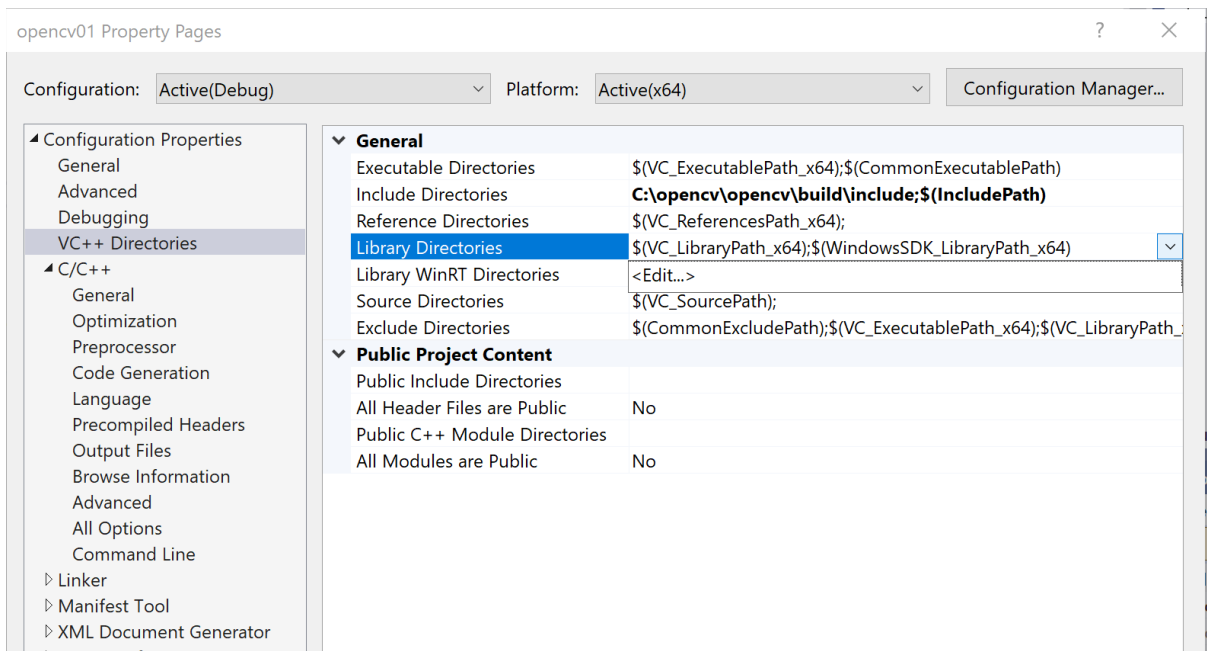


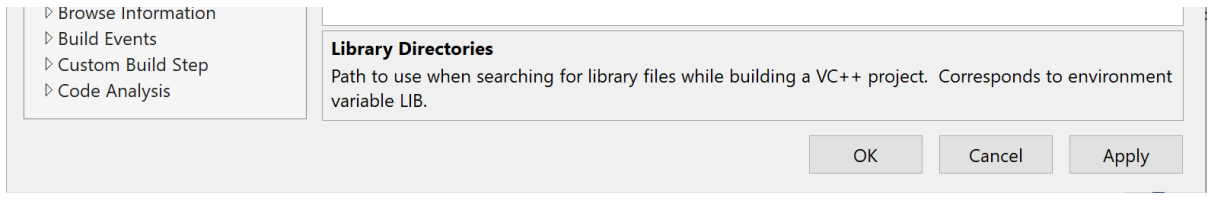
3. At VC++ Directories tab → Include Directories → click Edit and add opencv/include path

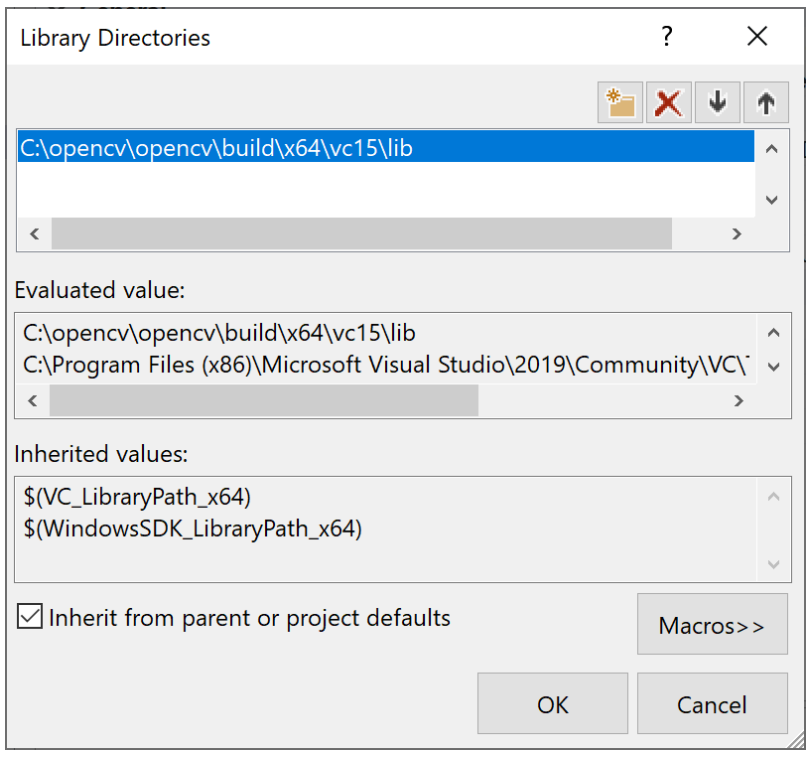




4. At VC++ Directories tab → Library Directories → click Edit and add opencv/lib path

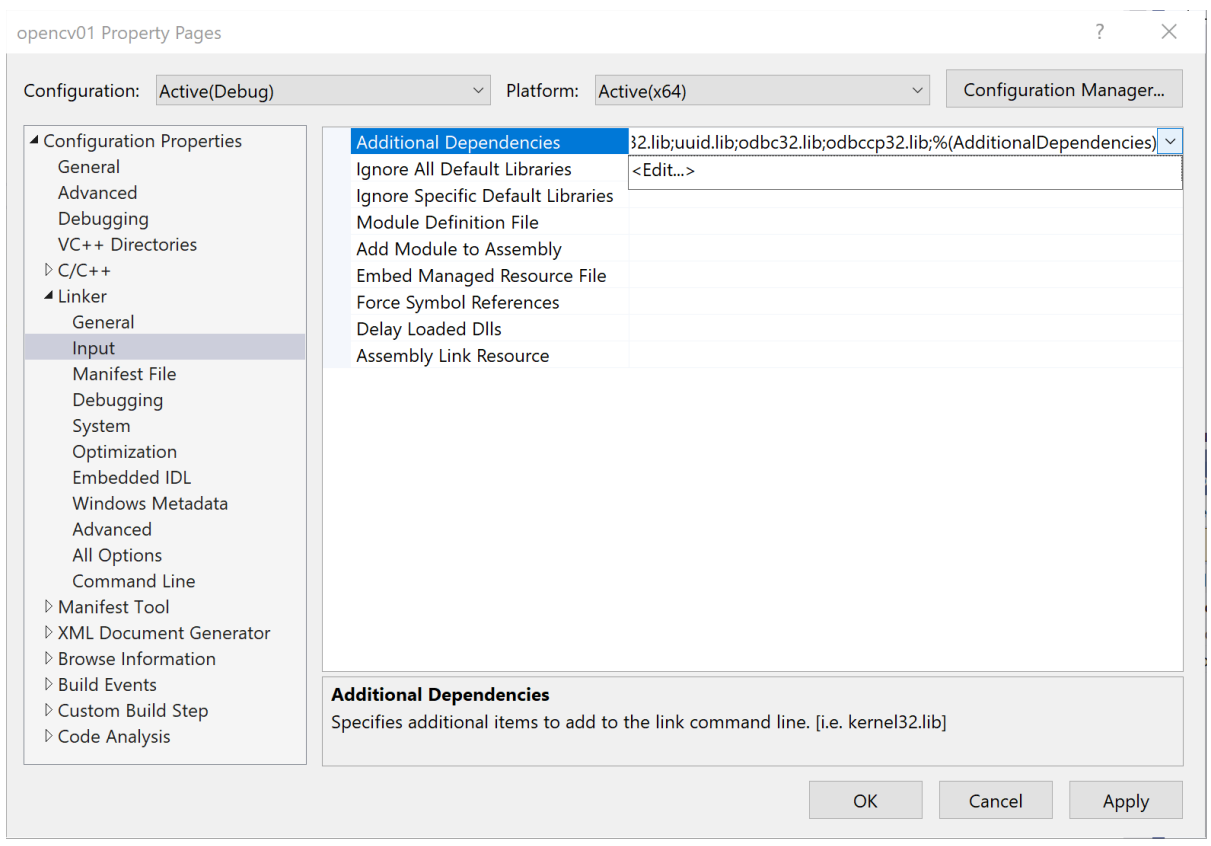


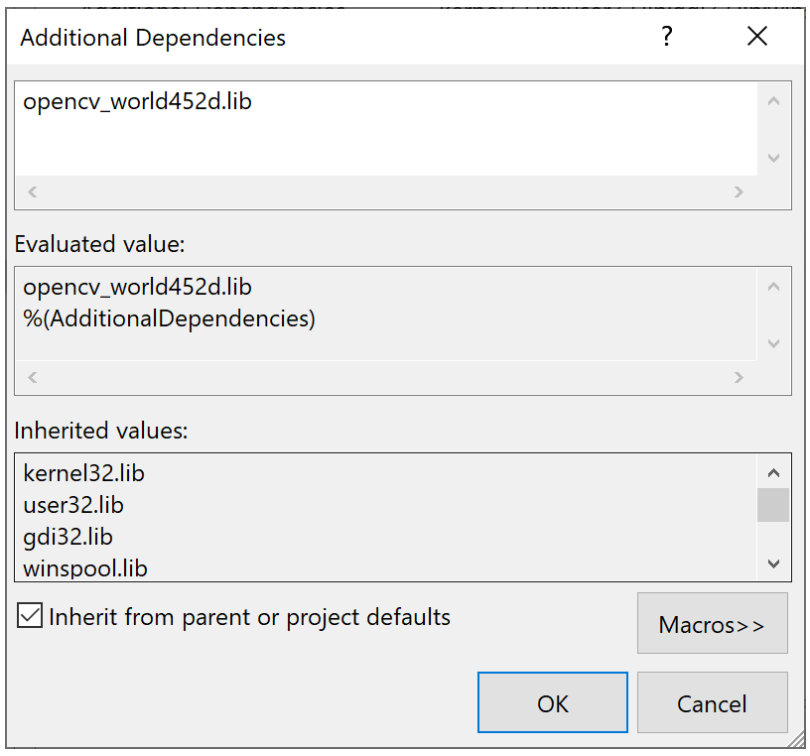




5. At Linker/Input tab → Additional Dependencies → click Edit and add `opencv_worldxxxd.lib` file

   Note: You can see the library file name in `C:\opencv\opencv\build\x64\vc15\lib`



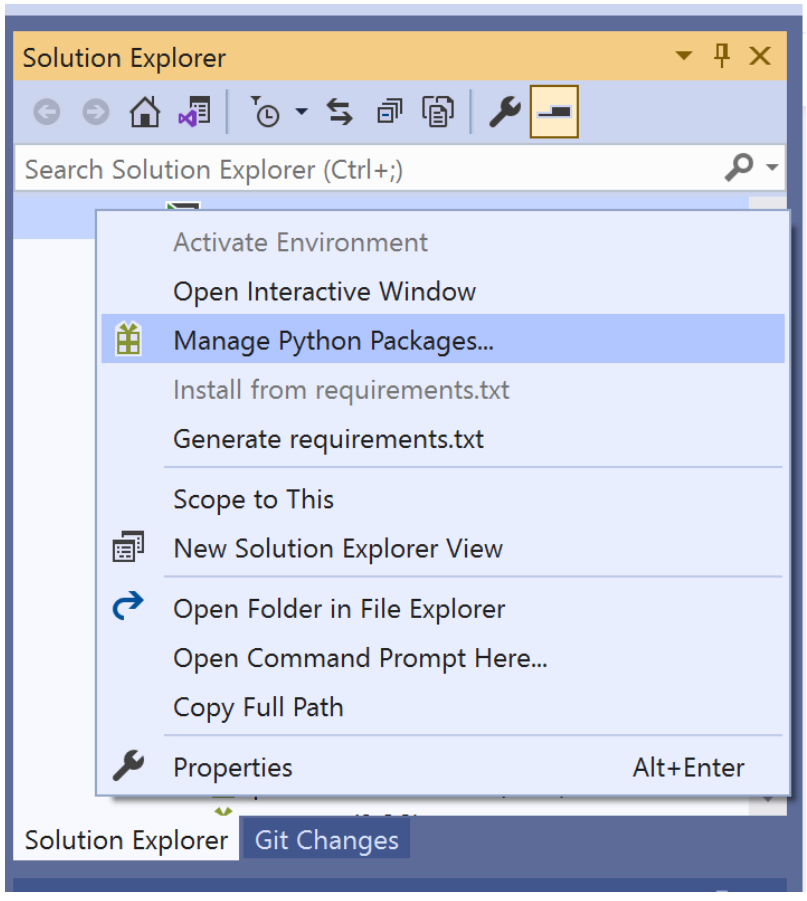


6. Try to write the code as below.

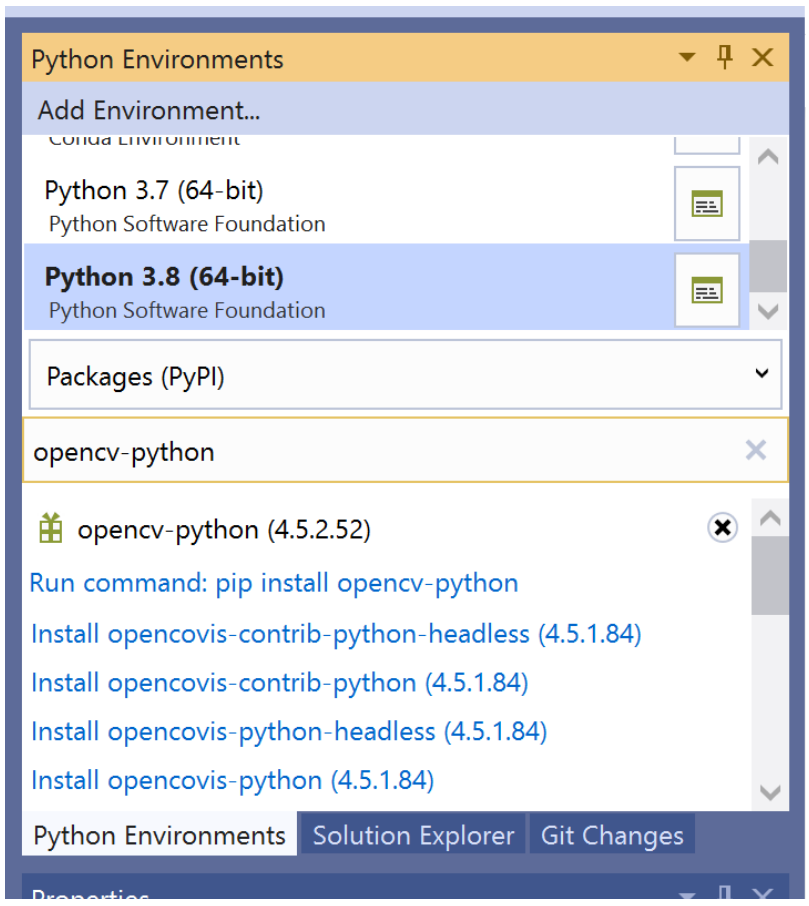
   Note: To run as debug, press F5. To run without debugging, press Ctrl+F5

Set up OpenCV Python in Visual Studio

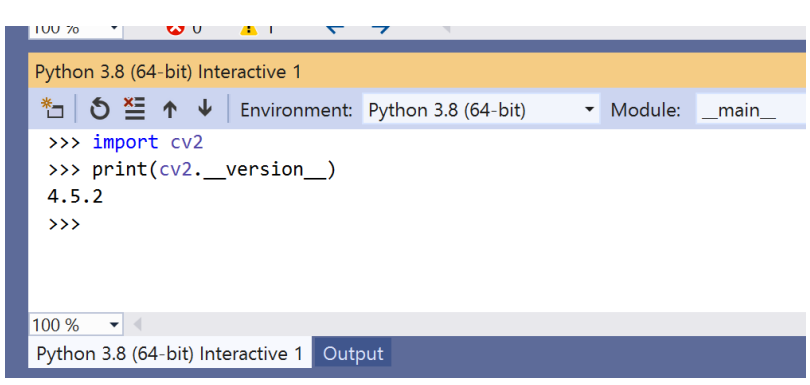
1. Create a python project in Visual studio
2. At the solution explorer, right click at **Python 3.x** → Manage Python package



3. Select Package PyPI → type **opencv-python** → double click at *install opencv-python*



4. Go to **Tools** → **Python** → *Python Interactive Wimdow*
5. Try to type anything of opencv code



You can check your version of OpenCV with code like this:

```
import cv2
print(cv2.__version__)
```

In [ ]:
```
import cv2
import numpy as np

print(cv2.__version__)
size = 300, 600, 3
image = np.zeros(size, dtype=np.uint8)
cv2.circle(image, (250,150), 100, (0,255,128), -100)
cv2.circle(image, (350,150), 100, (255,255,255), -100)
cv2.imshow("", image)
cv2.waitKey(0)

# If this says 4.3.0 or anything besides 4.5.5, you don't have the latest OpenCV!
```

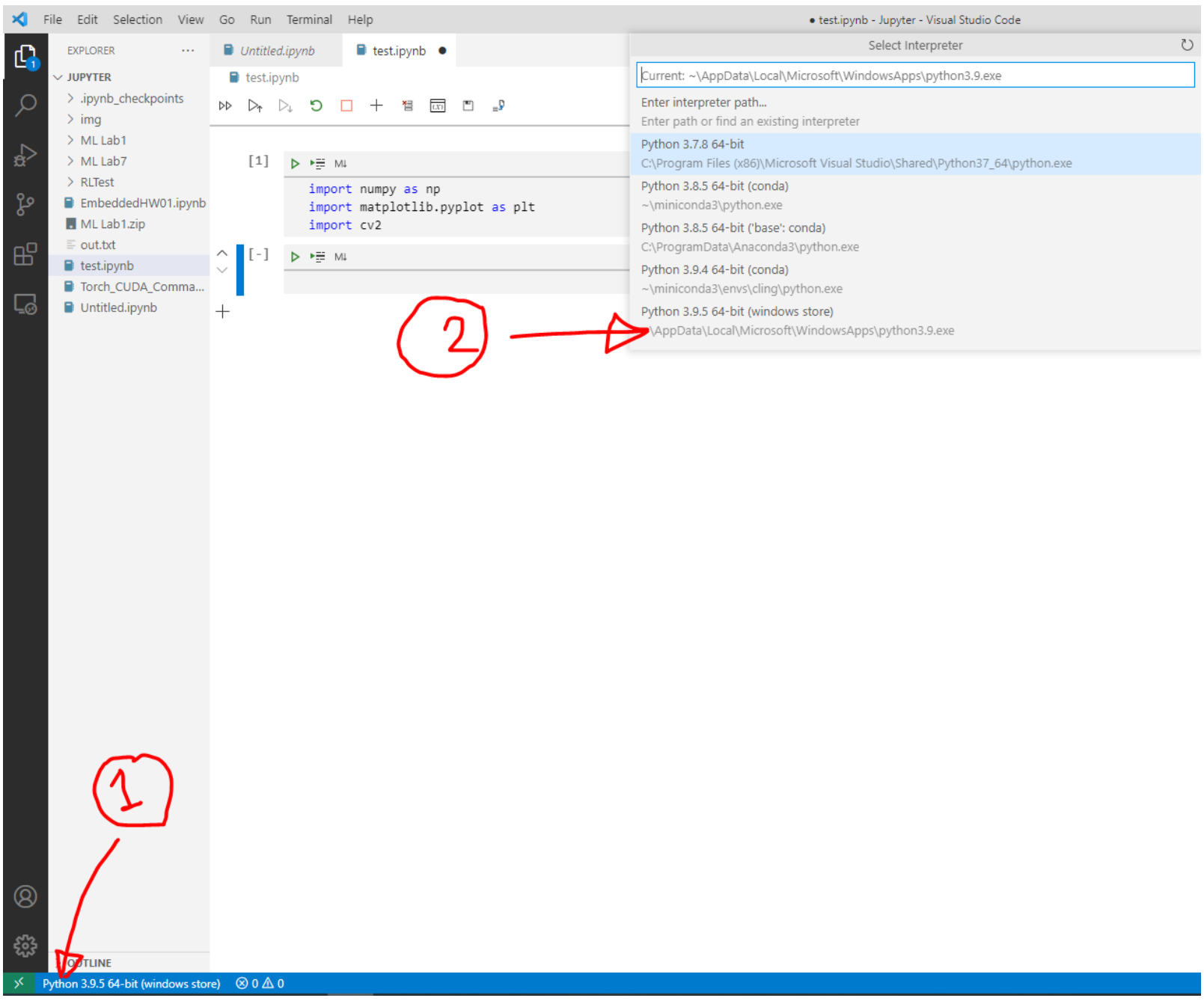
```
4.3.0
```

Windows Method 2 (Visual Studio Code)

Python

First, the Python installation:

1. Download Python from Microsoft Store (***Recommend***:by type **python** in command line) or [Python website (https://www.python.org/downloads/)](https://www.python.org/downloads/) and install it.
2. Set **PythonXX** path and **PythonXX/Scripts** path to environment variable (No need for microsoft store install)
    - **PythonXX** path - `C:\Users\[username]\AppData\Local\Programs\Python\Python39`
    - **PythonXX/Scripts** path - `C:\Users\[username]\AppData\Local\Programs\Python\Python39\Scripts`
3. Download Visual Studio Code from [here (https://code.visualstudio.com/download)](https://code.visualstudio.com/download) and install it.
4. Open command line:
    - Check python version `python --version`
    - Install virtual environment `pip install virtualenv`
    - Create a virtual environment called opencv `python -m virtualenv opencv`
    - Go to opencv environment, activate the virtual environment.
        - `cd opencv/Scripts`
        - `activate.bat`
    - Install numpy `pip install numpy`
    - Install MatPlotLib `pip install matplotlib`
    - Install OpenCV `pip install opencv-python`
    - Install Jupyter `python -m pip install jupyter`
    - Try to open Jupyter Notebook `jupyter notebook`
    - Cancel Jupyter notebook using Ctrl-C
5. Open Visual Studio Code and select Python interpreter (at left bottom of the screen).



6. Try to create a jupyter program or python program.

For jupyter notebook, you cannot use imshow directly. Please use pyplot to show image.

import cv2 import numpy as np import matplotlib.pyplot as plt

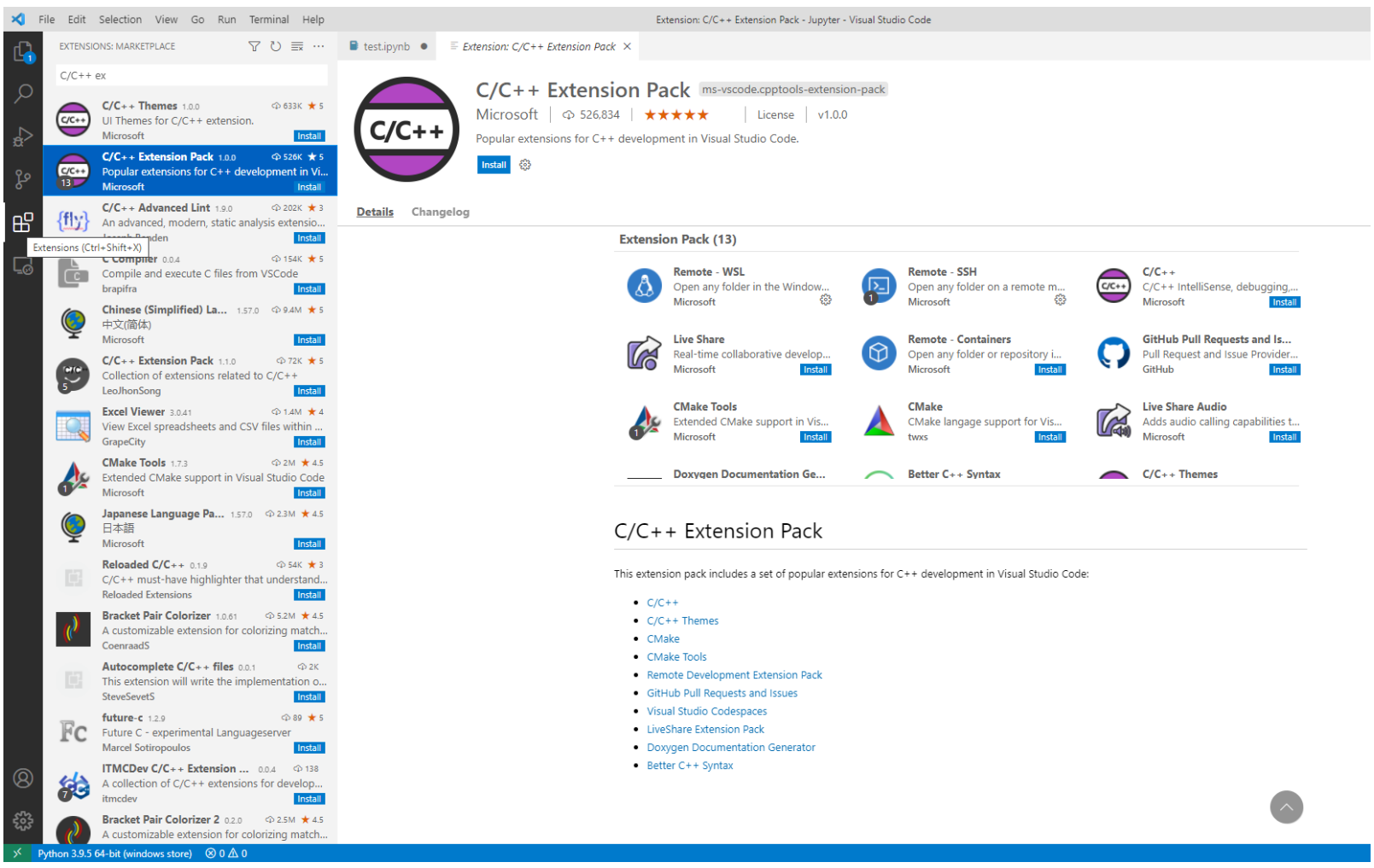
print(cv2.**version**) size = 300, 600, 3 image = np.zeros(size, dtype=np.uint8) cv2.circle(image, (250,150), 100, (0,255,128), -100) cv2.circle(image, (350,150), 100, (255,255,255), -100)

def imshow(image): img2 = image[:,:,::-1] #change color from BGR color to RGB color plt.imshow(img2) plt.title('image') plt.show()

imshow(image)

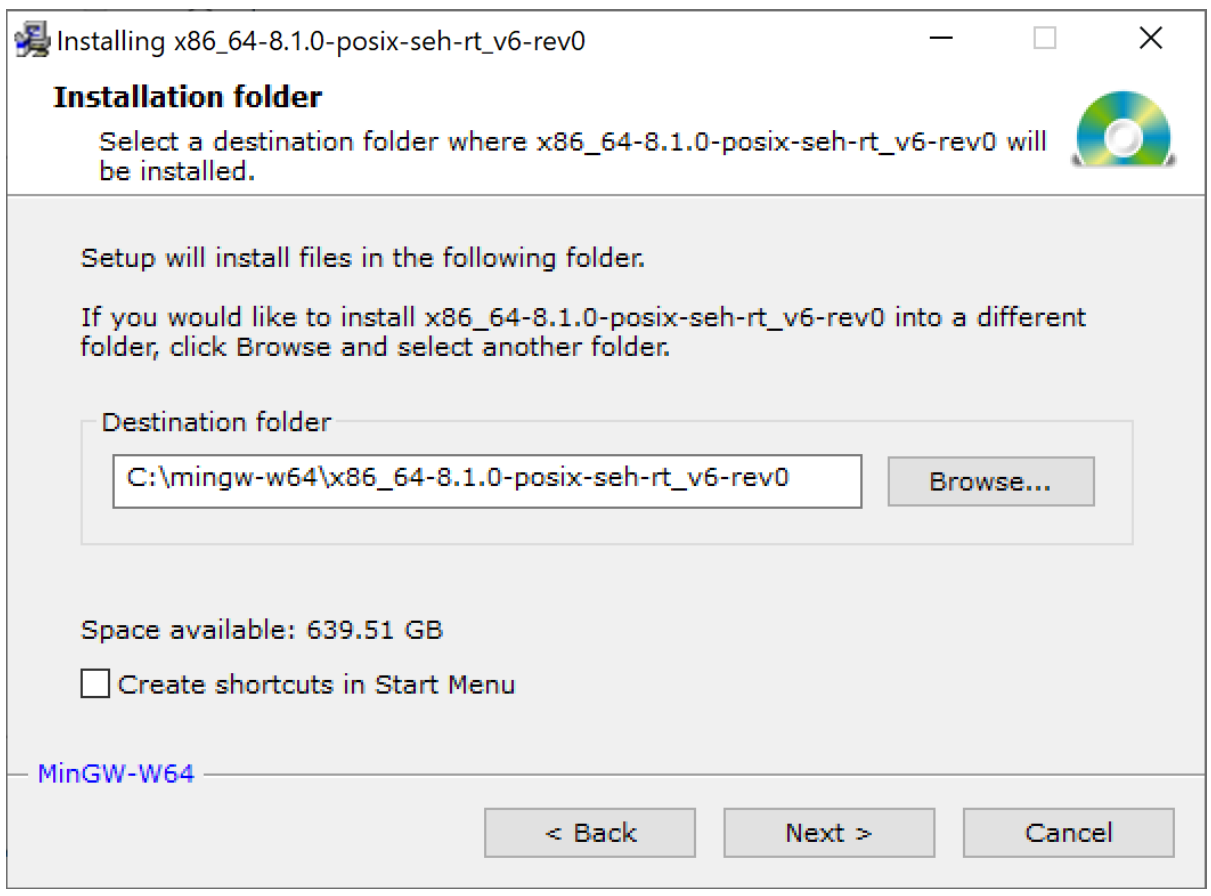
C++

1. After install VS code, Install VS Code C/C++ extensions (ms-vscode.cpptools)



2. Download and install MinGW-w64 from **here (https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win32/Personal%20Builds/mingw-builds/installer/mingw-w64-install.exe/download)**. Setting while install as:

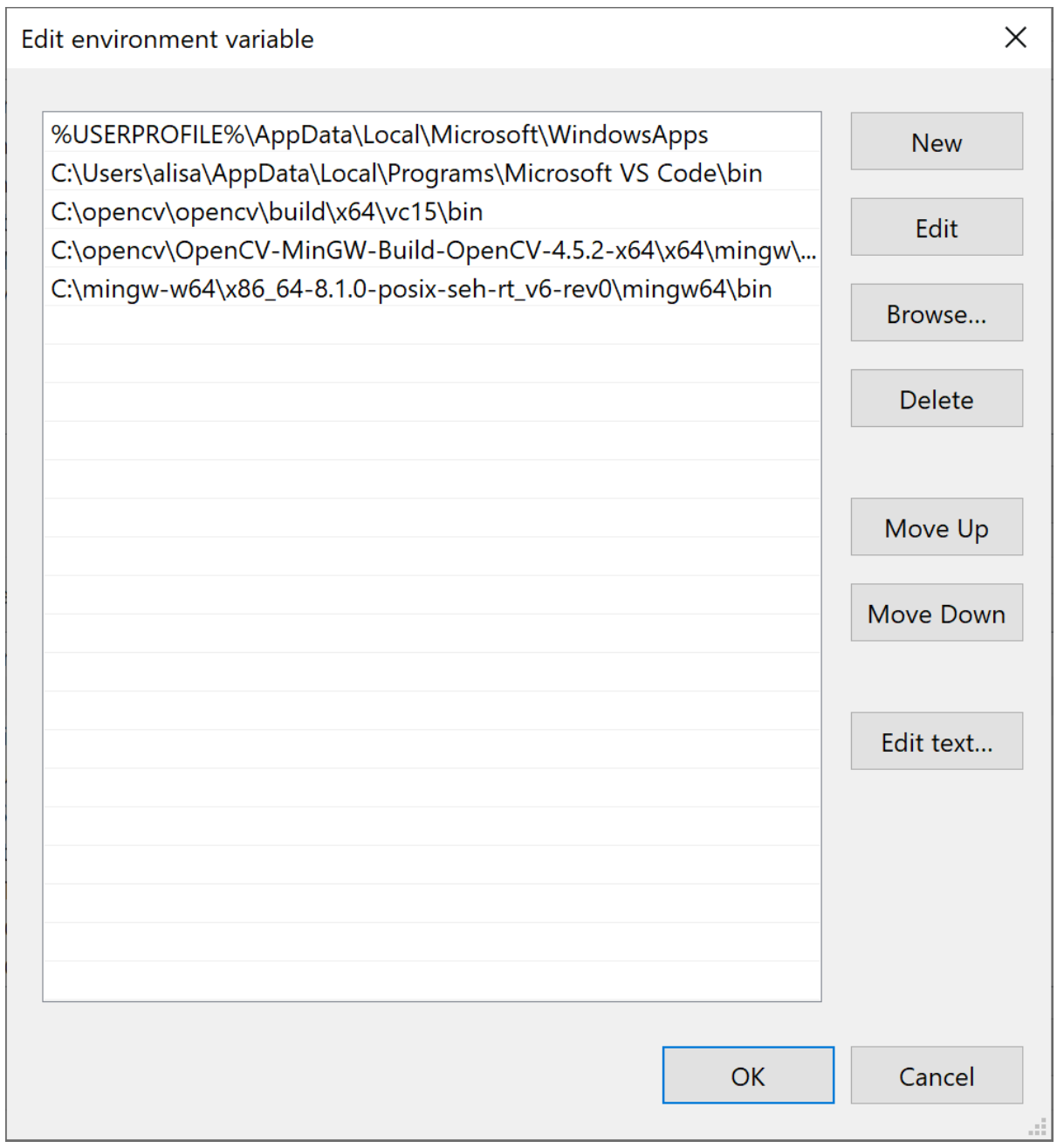
   - Version 8.1.0
   - Architecture x86-64
   - posix
   - Exception seh
   - Build revision 0
   - Set path of compiler as: `C:\mingw-w64\x86_64-8.1.0-posix-seh-rt_v6-rev0`



3. Download [OpenCV MinGW (https://github.com/huihut/OpenCV-MinGW-Build)](https://github.com/huihut/OpenCV-MinGW-Build) in zip file. unzip it and put into `C:\opencv\OpenCV-MinGW-Build-OpenCV-4.5.5-x64`
4. Go to System Properties → *Environment Variables*

   Note: In Windows10, you can go type "Environment Variables" to search System Properties

   - At the User variables → Path tab select → *Edit..*
   - Press *New* and input
     - Binaries OpenCV path: `C:\opencv\OpenCV-MinGW-Build-OpenCV-4.5.5-x64\x64\mingw\bin`
     - Binaries MinGW path: `C:\mingw-w64\x86_64-8.1.0-posix-seh-rt_v6-rev0\mingw64\bin`



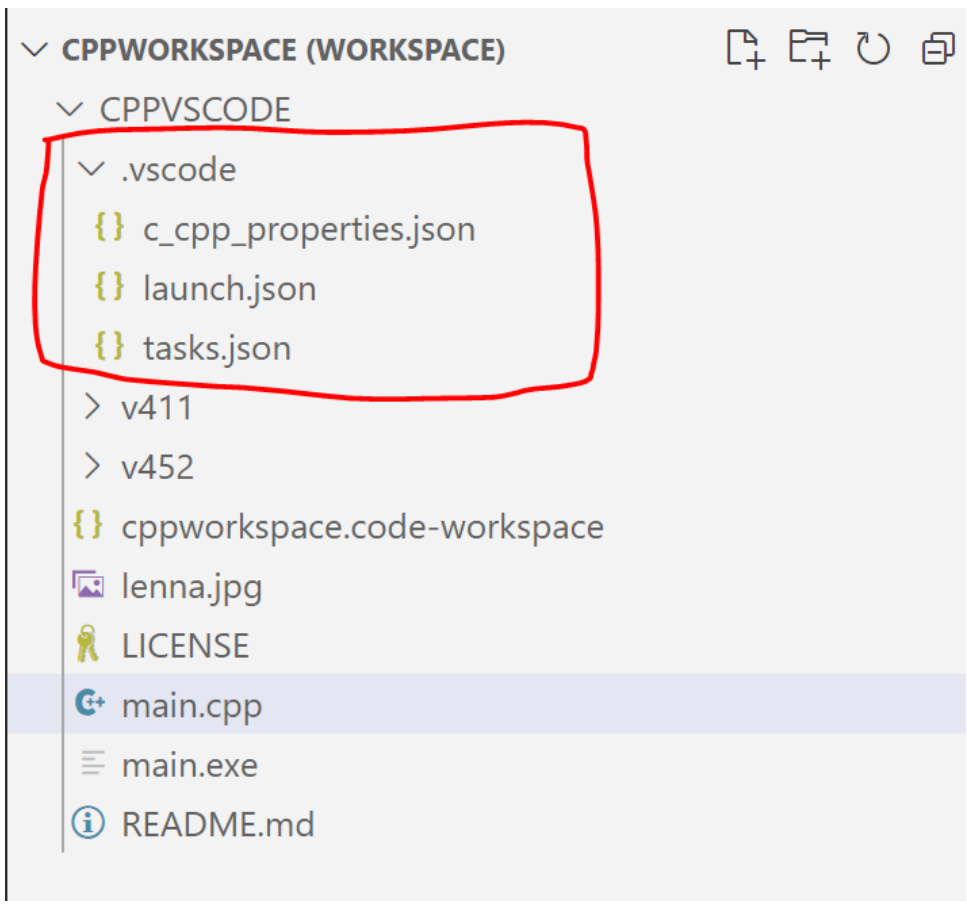
Note: If you don't do this step, you will find *gdb* error in last step

5. Configure file in vscode

   5.1. Create a workspace in vs code.

   5.2. Create a main.cpp and try to run it.

   5.3. in folder .vscode, you will see `tasks.json` , `launch.json` , and `c_cpp_properties.json` . The code will be looked like [repo (https://github.com/Cuda-Chen/opencv-config-with-vscode)](https://github.com/Cuda-Chen/opencv-config-with-vscode)



5.4. Modify all 3 files as below (You can copy)

tasks.json

```
{
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: g++.exe build active file",
            "command": "C:\\mingw-w64\\x86_64-8.1.0-posix-seh-rt_v6-rev0\\mingw64\\bin\\g++.exe",
            "args": [
                "-g",
                "${file}",
                "-o",
                "${fileDirname}\\${fileBasenameNoExtension}.exe",
                "-I","C:\\opencv\\OpenCV-MinGW-Build-OpenCV-4.5.5-x64\\include",
                "-L","C:\\opencv\\OpenCV-MinGW-Build-OpenCV-4.5.5-x64\\x64\\mingw\\bin",
                "-l","libopencv_calib3d452",
                "-l","libopencv_core452",
                "-l","libopencv_dnn452",
                "-l","libopencv_features2d452",
                "-l","libopencv_flann452",
                "-l","libopencv_highgui452",
                "-l","libopencv_imgcodecs452",
                "-l","libopencv_imgproc452",
                "-l","libopencv_ml452",
                "-l","libopencv_objdetect452",
                "-l","libopencv_photo452",
                "-l","libopencv_stitching452",
                "-l","libopencv_video452",
                "-l","libopencv_videoio452"
            ],
            "options": {
                "cwd": "C:\\mingw-w64\\x86_64-8.1.0-posix-seh-rt_v6-rev0\\mingw64\\bin"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "detail": "Task generated by Debugger."
        }
    ],
    "version": "2.0.0"
}
```

launch.json

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "g++.exe - Build and debug active file",
            "type": "cppdbg",
            "request": "launch",
            "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
            "args": [],
            "stopAtEntry": true,
            "cwd": "C:\\mingw-w64\\x86_64-8.1.0-posix-seh-rt_v6-rev0\\mingw64\\bin",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "miDebuggerPath": "C:\\mingw-w64\\x86_64-8.1.0-posix-seh-rt_v6-rev0\\mingw64
\\bin\\gdb.exe",
```

```
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "C/C++: g++.exe build active file"
        }
    ]
}
```

c_cpp_properties.json

```
{
    "configurations": [
        {
            "name": "Win32",
            "includePath": [
                "${workspaceFolder}/**",
                "C:\\opencv\\OpenCV-MinGW-Build-OpenCV-4.5.5-x64\\include"
            ],
            "defines": [
                "_DEBUG",
                "UNICODE",
                "_UNICODE"
            ],
            "compilerPath": "C:\\mingw-w64\\x86_64-8.1.0-posix-seh-rt_v6-rev0\\mingw64
\\bin\\gcc.exe",
            "cStandard": "c11",
            "cppStandard": "c++17",
            "intelliSenseMode": "clang-x64"
        }
    ],
    "version": 4
}
```

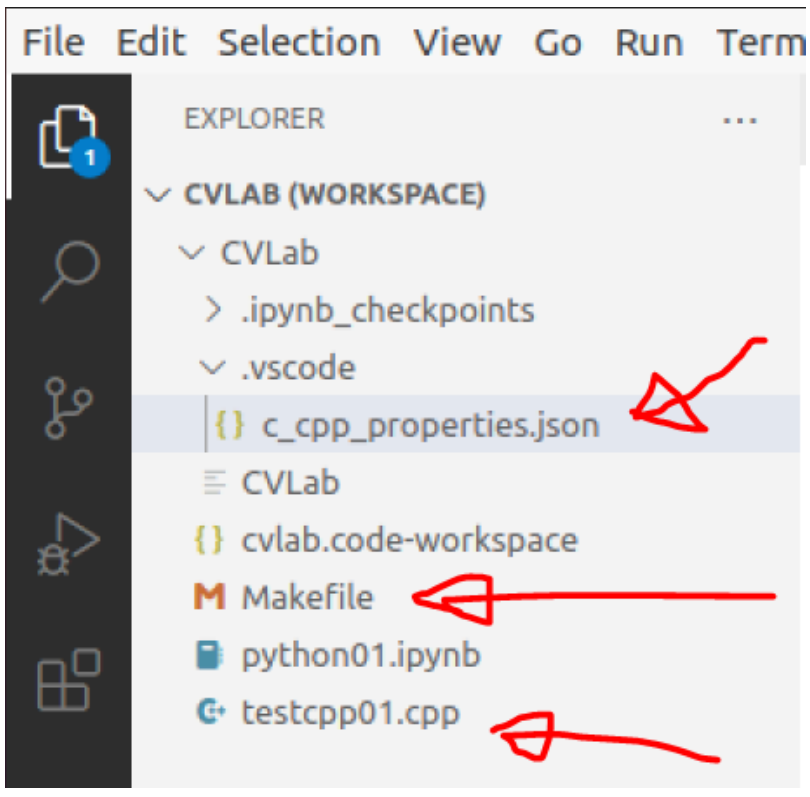
6. Write some code to test:

```
#include

#include <opencv2/opencv.hpp>


using namespace cv;


int main( int argc, char** argv )
{
    std::cout << "aa" << std::endl;
```

```
    Mat image = Mat::zeros(300, 600, CV_8UC3);
    circle(image, Point(250, 150), 100, Scalar(0, 255, 128), -100);
    circle(image, Point(350, 150), 100, Scalar(255, 255, 255), -100);
    imshow("Display Window", image);
    waitKey(0);
    std::cout << "bb" << std::endl;

    // In case of load image file
    std::string img = "C:\\Users\\alisa\\OneDrive\\Documents\\CPPVSCODE\\lenna.jpg";
    Mat srcImage = imread(img);
    if (!srcImage.data) {
        return 1;
    }
    imshow("srcImage", srcImage);
    waitKey(0);

    Mat greyMat;
    cv::cvtColor(srcImage, greyMat, COLOR_BGR2GRAY);
    imshow("greyImage", greyMat);
    waitKey(0);

    return 0;

}
```

Additional OpenCV C++ setup for Windows VS code

1. After setting OpenCV Python completed, open VS code.
2. Create a workspace, then go to **.vscode** folder create a file named **c_cpp_properties.json**



3. input data as below

```
{
 "configurations": [
     {
         "name": "Win32",
         "includePath": [
             "${workspaceFolder}/**",
             "/home/alisa/anaconda3/include/opencv4/**" <<< Your opencv4 path
         ],
         "defines": [
             "_DEBUG",
             "UNICODE",
             "_UNICODE"
         ],
         "compilerPath": "/usr/bin/g++",
         "cStandard": "c11",
         "cppStandard": "c++17",
         "intelliSenseMode": "clang-x64"
     }
 ],
 "version": 4
}
```

```
    "configurations": [
        {
            "name": "Win32",
            "includePath": [
                "${workspaceFolder}/**",
                "/home/alisa/anaconda3/include/opencv4/**"
            ],
            "defines": [
                "_DEBUG",
                "UNICODE",
                "_UNICODE"
            ],
            "compilerPath": "/usr/bin/g++",
            "cStandard": "c11",
            "cppStandard": "c++17",
            "intelliSenseMode": "clang-x64"
        }
    ],
    "version": 4
}
```

4. At line `"/home/alisa/anaconda3/include/opencv4/**"` , search the path yourself, and modify it.
5. Create a cpp file which has opencv code.
6. Create **Makefile** which has detail below.

CVLab > Makefile

```
CC = "g++"
PROJECT = CVLab
SRC = testcpp01.cpp

LIBS = `pkg-config opencv4 --cflags --libs`

$(PROJECT) : $(SRC)
    $(CC) $(SRC) -o $(PROJECT) $(LIBS)
```

```
// make file

CC = "g++"
PROJECT = CVLab  // your project name
SRC = testcpp01.cpp  // cpp file


LIBS =  pkg-config opencv4 --cflags --libs


/$(PROJECT) : /$(SRC)
    $(CC) $(SRC) -o $(PROJECT) $(LIBS)
```

7. Open terminal, run `make`
8. Enable library path: `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/`
9. Run application project file `./CVLab`

Python - Get Started | testcpp01.cpp

CVLab > testcpp01.cpp > main()

```
#include <iostream>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main()
{
    cout << "Hello World!" << endl;

    Mat image = Mat::zeros(300, 600, CV_8UC3);
    circle(image, Point(250, 150), 100, Scalar(0, 255, 128), -100);
    circle(image, Point(350, 150), 100, Scalar(255, 255, 255), -100);
    imshow("Display Window", image);
```

Alternative method for installing OpenCV on Ubuntu 20.04 with VS Code

I like the custom build of OpenCV for my C++ development as described in the beginning. If you aren't too picky, though, you can skip the custom build of OpenCV and just go with the libraries that come with Python and use VSCode for your dev environment, similar to Windows:

1. Download VS code from **here (https://code.visualstudio.com/download)** and install it.
2. Open terminal and do step:
   - Update and Refresh Repository lists:
     - `sudo apt-get update`
     - `sudo apt-get upgrade`
   - Install Python: `sudo apt-get install build-essential cmake python3-numpy python3-dev python3-tk libavcodec-dev libavformat-dev libavutil-dev libswscale-dev libavresample-dev libdc1394-dev libeigen3-dev libgtk-3-dev libvtk7-qt-dev`
   - Upgrade pip: `sudo -H pip3 install --upgrade pip`

- Install virtual environment: `sudo -H pip3 install virtualenv`
- Create virtual environment: `virtualenv opencv`
- Activate virtual environment: `source opencv/bin/activate`
- Install Jupyter: `pip install jupyter`
- Install OpenCV: `sudo apt-get install python3-opencv`
- Install OpenCV-dev: `sudo apt-get install libopencv-dev` (for C++ use)
- Run Jupyter Notebook: `jupyter notebook`

3. Open Visual Studio Code and select Python interpreter (at left bottom of the screen).
4. Try to code OpenCV for check the error.

Start an OpenCV C++ project (no IDE, for Linux)

If you're on Windows, you've followed the instructions above and can already compile and run C++ programs. Right?

If you're on Linux, let's start our first project. In some directory on your file system, create files `main.cpp` and `CMakeLists.txt`.

For `main.cpp`, start out with a "Hello World!" message:

```
#include <iostream>
using namespace std;

int main(void)
{
  cout << "Hello, World!" << endl;
  return 0;
}
```

In `CMakeLists.txt`, add some minimal definitions to generate the `Makefile`:

```
cmake_minimum_required(VERSION 3.12)

project(hello)
set(CMAKE_CXX_STANDARD 14)

add_executable(hello main.cpp)
```

To build, make a build directory, run `cmake`, then run `make`:

```
$ cd /home/$USER/CV/hello
$ mkdir build-debug
    $ cd build-debug
$ cmake -DCMAKE_BUILD_TYPE=Debug ..
    $ make
$ ./hello
```

Note that here we're asking `cmake` to build a "debug" executable. To run the program in the debugger and step through it:

```
$ sudo apt-get install gdb
    $ gdb ./hello
(gdb) break main
Breakpoint 1 at 0x11a9: file /home/mdailey/CV/hello/main.cpp, line 5.
(gdb) run
Starting program: /home/mdailey/CV/hello/build-debug/hello
(gdb) list
1
2       #include <iostream>
3       using namespace std;
4
5       int main() {
6           cout << "Hello, World!" << endl;
7           return 0;
8       }
(gdb) next
6           cout << "Hello, World!" << endl;
(gdb) next
Hello, World!
7           return 0;
(gdb) next
```

If you want your code to run as fast as possible with optimization, you have to use "release" mode for the build:

```
$ cd /home/$USER/CV/hello
$ mkdir build-release
    $ cd build-release
$ cmake -DCMAKE_BUILD_TYPE=Release ..
    $ make
$ ./hello
```

You can still run the executable in the debugger, but you'll only see the assembly language instructions instead of the original source code.

OK, but what about OpenCV? Let's do something that uses the library. Add necessary definitions to your `CMakeLists.txt`:

```
find_package(OpenCV 4.5.5 REQUIRED)
target_include_directories(hello PUBLIC ${OpenCV_INCLUDE_DIRS})
    target_link_libraries(hello ${OpenCV_LIBS})
```

If all goes well, when you run `cmake`, now you should see a message like

```
-- Found OpenCV: /usr/local/opencv-4.5.5 (found suitable version "4.5.5", minimum required
is "4.5.5")
```

If you don't see "Found OpenCV: ..." or have some other error, you have probably installed OpenCV in a place that cmake can't find it. Try giving it a hint:

```
find_package(OpenCV 4.5.5 REQUIRED PATHS /home/$USER/wherever/opencv/is)
```

In our simple program, let's get the singular value decomposition of a matrix. You'll need to include OpenCV's core functionality:

```
#include <opencv2/core.hpp>
```

then, in the `main()` function:

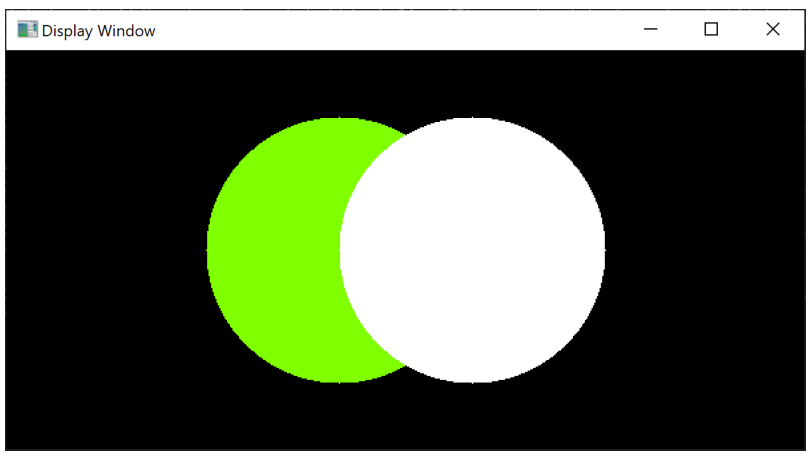
```
double adData[] = { 3, 2, 4, 8, 4, 2, 1, 3, 2 };
cv::Mat matA( 3, 3, CV_64F, adData );
cout << "A:" << endl << matA << endl;
cv::SVD svdA( matA, cv::SVD::FULL_UV );
cout << "U:" << endl << svdA.u << endl;
cout << "W:" << endl << svdA.w << endl;
cout << "Vt:" << endl << svdA.vt << endl;
```

Try it! To verify correctness of the result, we'll use Octave in the next part below.

While we're here, though, let's try another program that uses some graphics:

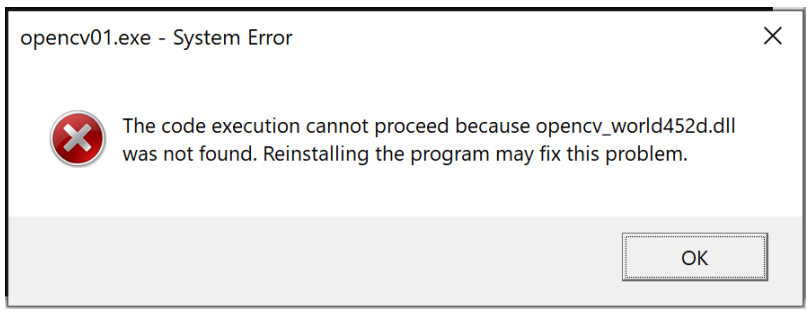
```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main()
{
    Mat image = Mat::zeros(300, 600, CV_8UC3);
    circle(image, Point(250, 150), 100, Scalar(0, 255, 128), -100);
    circle(image, Point(350, 150), 100, Scalar(255, 255, 255), -100);
    imshow("Display Window", image);
    waitKey(0);
    return 0;
}
```

Do you get a result like this?



Windows tip if it does not work

If you found this error:



The easiest way to solve the problem is copy `opencv_world452d.dll` from `C:\opencv\opencv\build\x64\vc15\bin`

Python OpenCV Programs

Regardless of how you installed, you should be able to run a Python program using OpenCV.

Note: if you've compiled OpenCV from source and want Python to use the version you've compiled rather than the default version from pip, you need to set the `PYTHONPATH` either in your environment setup file (`$HOME/.bashrc` in Ubuntu) or at the command line:

```
$ cat > myprog.py
    import cv2
    print(cv2.__version__)
    $ python3 myprog.py
4.3.0
$ export PYTHONPATH=/usr/local/opencv-4.5.5/lib/python3.8/dist-packages
    $ python3 myprog.py
4.5.5
```

In [ ]:
```
# Python SVD example

import numpy as np
import cv2

matA = np.array([[3.0,2.0,4.0],[8.0,4.0,2.0],[1.0,3.0,2.0]])
w, u, vt = cv2.SVDecomp(matA)
print("A:")
print(matA)
print("U:")
print(u)
print("W:")
print(w)
print("Vt:")
print(vt)
```

IDE or no IDE?

Some experienced programmers say that IDEs are bad for you, because they get between you and your code.

My view is that it's important to understand how to do things without an IDE, so that you don't depend on it. Then, if you use an IDE, you should take care to understand precisely what it is doing for you and how.

That's the beauty of development on Linux. Since everything is open and nothing is hidden from you, you have full control over everything you're doing, and you can go as deep as you want to to understand what your system is doing. IDEs that work well in Linux are just wrappers around the same ordinary tools that we would use from the command line.

So I don't think it's harmful to use a good IDE for C++ development on Linux. One of the best I've seen is CLion by JetBrains. Since JetBrains gives educational licenses for all of its IDEs, we can use it freely, and once you have a real job, you can convince your boss to buy licenses.

Note, however, that CLion can be heavy and with all the indexing and code completion it's doing, you'll burn through your battery charge on your laptop pretty quick! You'll probably have to plug in or use the PowerSave mode.

If you want to, go ahead and download CLion from [the JetBrains CLion download page (https://www.jetbrains.com/clion/download/#section=linux)](https://www.jetbrains.com/clion/download/#section=linux) and set it up. Repeat the `CMakeLists.txt` and `main.cpp` setup from the first part of the lab, and get your OpenCV program running at the console and in the debugger.

Alternatively, we can use Visual Studio Code, which is not quite as capable as CLion, especially when it comes to automatically refactoring code, but is more lightweight and lets you move around between Python and C++ and other languages very easily. It also supports remote development over a SSH connection. If you want to use VSCode, go to [the VSCode home page (https://code.visualstudio.com/)](https://code.visualstudio.com/) and download the package for your OS (64-bit .deb if you're on Ubuntu).

Octave

Developing OpenCV code in C++ can be tedious sometimes, especially when the program is not quite working and you're not sure where the mistake in your calculations is.

One way to develop more quickly would be to use Python instead of C++ for the development. This may be more productive, but you won't learn as much about the OpenCV core, internals will be more mysterious, and your code won't (usually) be as fast as it could be.

An even better way to develop *algorithms*, however, is to code them in a much higher level language such as Matlab or Octave, verify correctness, then port the algorithm to OpenCV, either in C++ or Python.

OK, so let's install Octave:

```
$ sudo apt-get install octave
```

I am old fashioned and don't really care for Octave's GUI, so I run like this:

```
$ octave --no-gui
GNU Octave, version 5.2.0
...
octave:1> A = [ 3, 2, 4; 8, 4, 2; 1, 3, 2 ]
A =

   3   2   4
   8   4   2
   1   3   2

octave:2> [U, W, Vt] = svd(A)
U =

  -0.455849   0.637848  -0.620767
  -0.844431  -0.530370   0.075129
  -0.281315   0.558442   0.780387

W =

Diagonal Matrix

  10.6578         0         0
        0    3.2594         0
        0         0    1.6696

Vt =

  -0.78856  -0.54334  -0.28802
  -0.48165   0.25451   0.83859
  -0.38234   0.80000  -0.46240

octave:3>
```

Now you can verify whether the OpenCV code and the Octave code give the same result. First, you will probably see differences in the sign of the basis vectors in U and Vt but that's not important. More importantly, however, notice that the result for the third matrix is transposed.

This can be confusing. The SVD factors a matrix A as a product of two orthogonal matrices and a diagonal matrix. We usually write the svd as U W V' = A. But some libraries will give you V, and some will give you V transposed.

It is easy to check which is which in Octave:

```
octave:3> U * W * Vt
ans =

   3.22601   2.33969   3.62198
   7.88153   4.55033   1.08442
   0.98938   3.13467   1.78744
```

Since we didn't get our original A back, we would hypothesize that Octave is giving us V, not V transposed:

```
octave:4> U * W * Vt'
ans =
   3.00000   2.00000   4.00000
   8.00000   4.00000   2.00000
   1.00000   3.00000   2.00000
```

So now we know that our Octave code should have read

```
octave:5> [U, W, V] = svd(A);
octave:6> U*W*V'
ans =

   3.00000   2.00000   4.00000
   8.00000   4.00000   2.00000
   1.00000   3.00000   2.00000
```

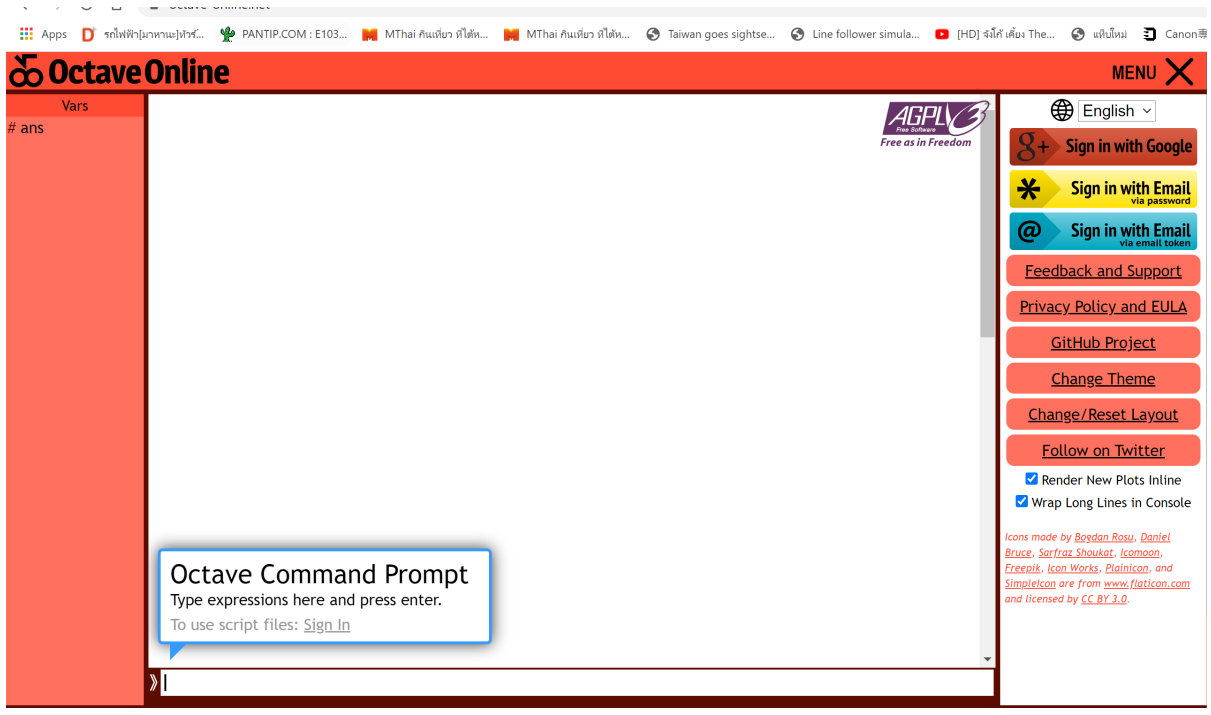
and we can also check the result in the C++ code:

```
cout << "Reconstruction of A from its SVD:" << endl << svdA.u * cv::Mat::diag(svdA.w) * svd
A.vt << endl;
```

Octave Online?

If you don't want to install Octave on your machine, you may want to try it online.

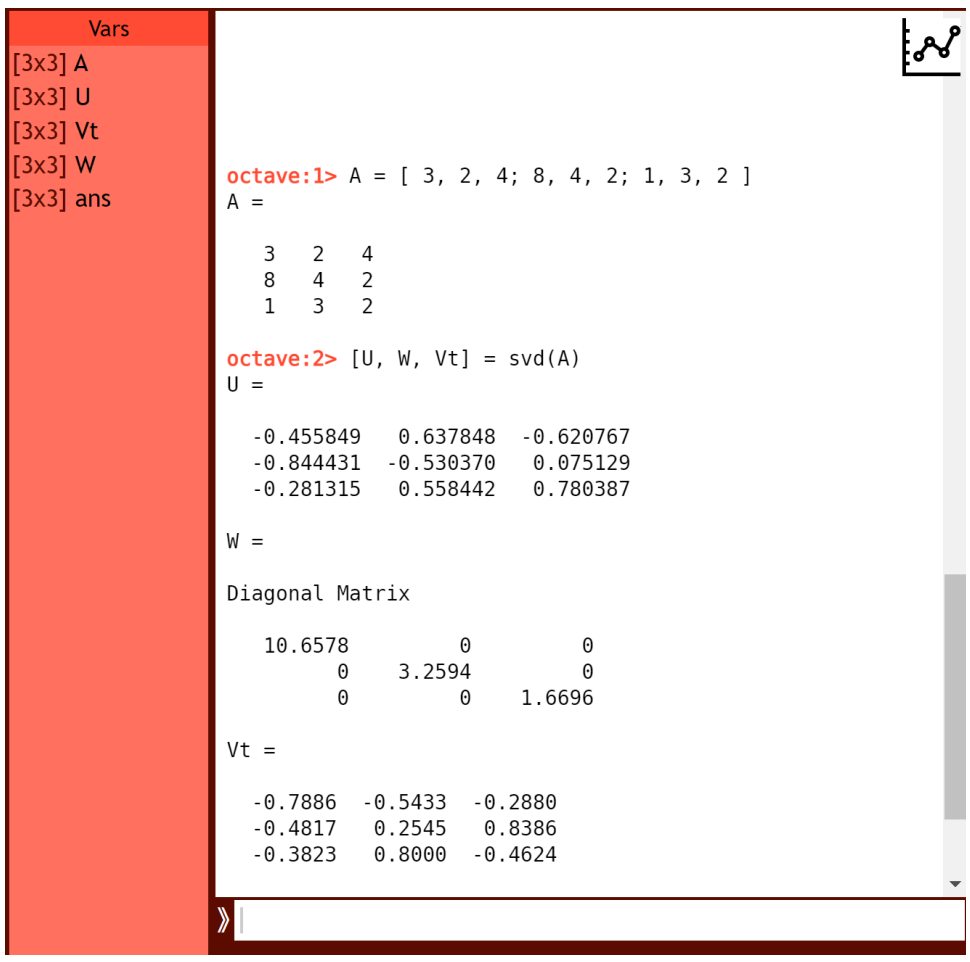
1. Go to https://octave-online.net/ (https://octave-online.net/) for run octave online (Need internet)

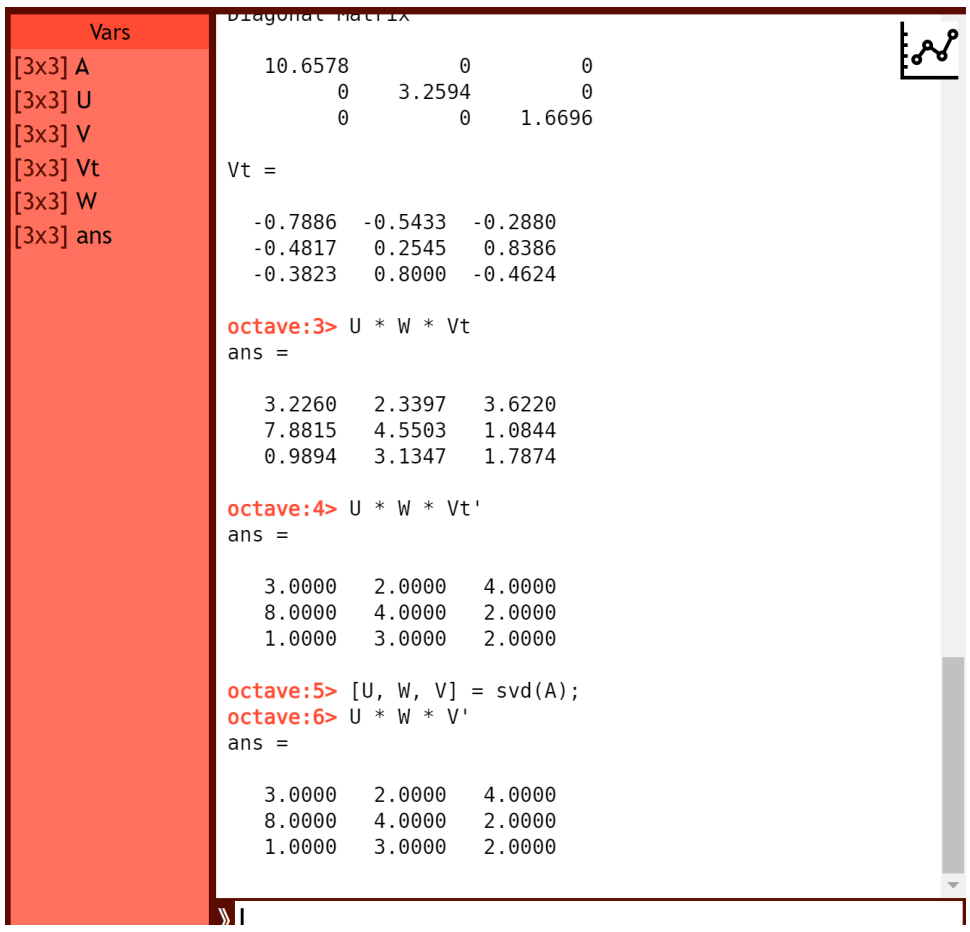


2. Click at ***Menu*** → Sign in with google → Use youre e-mail
3. You can use it as Matlab or octave.
4. At the left side, you can create m-files for do your homework.
5. Here's the same SVD example:

```
octave:1> A = [ 3, 2, 4; 8, 4, 2; 1, 3, 2 ]
A =

   3   2   4
   8   4   2
   1   3   2


octave:2> [U, W, Vt] = svd(A)
U =


  -0.455849   0.637848  -0.620767
  -0.844431  -0.530370   0.075129
  -0.281315   0.558442   0.780387


W =


Diagonal Matrix


  10.6578         0         0
        0    3.2594         0
        0         0    1.6696


Vt =

  -0.78856  -0.54334  -0.28802
  -0.48165   0.25451   0.83859
  -0.38234   0.80000  -0.46240


octave:3> U * W * Vt
ans =


   3.22601   2.33969   3.62198
   7.88153   4.55033   1.08442
   0.98938   3.13467   1.78744


octave:4> U * W * Vt'
ans =


   3.00000   2.00000   4.00000
   8.00000   4.00000   2.00000
   1.00000   3.00000   2.00000


octave:5> [U, W, V] = svd(A);
```

```
octave:6> U * W * V'
ans =

   3.00000   2.00000   4.00000
   8.00000   4.00000   2.00000
   1.00000   3.00000   2.00000
```





Exercises to do in lab

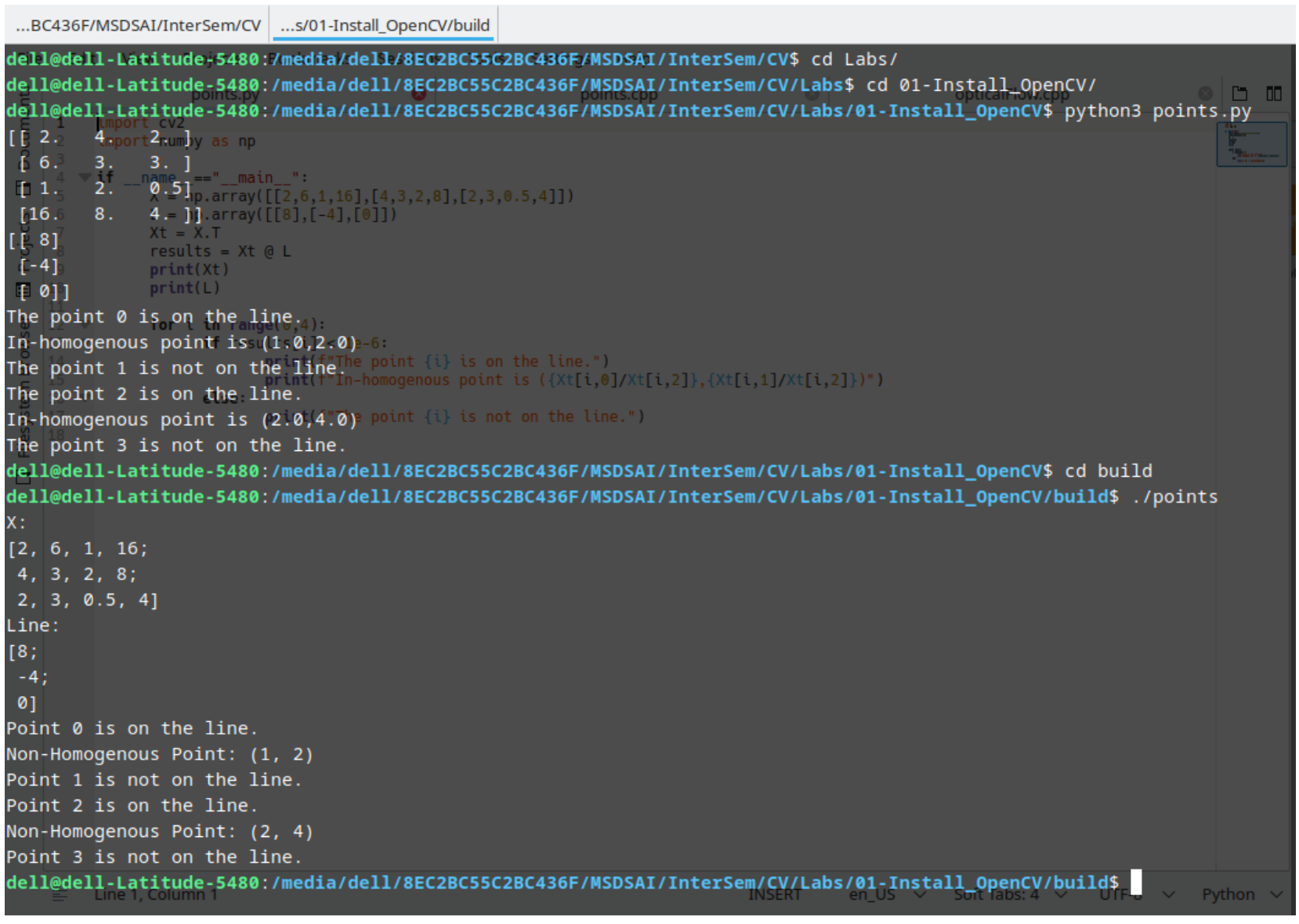
1. Write OpenCV C++, OpenCV Python, and Octave code to test which, if any, of the homogeneous 2D points (2, 4, 2), (6, 3, 3), (1, 2, 0.5), and (16, 8, 4) are on the homogeneous 2D line (8, -4, 0). Output the inhomogeneous representations of the points that are on the line.
2. Figure out how to plot the 2D line (8, -4, 0) and the four homogeneous 2D points from exercise 1 above in Octave (you will need to install gnuplot).

Exercise 1

In [ ]:
```
// C++ Implementation

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>

using namespace cv;
using namespace std;
```

```
int main()
{
    double adDataX[] = {2, 6, 1, 16, 4, 3, 2, 8, 2, 3, 0.5, 4};
    cv::Mat matX(3, 4, CV_64F, adDataX);
    cout << "X: " << endl << matX << endl;
    double adDataL[] = {8, -4, 0};
    cv::Mat matL(3,1,CV_64F, adDataL);
    cout << "Line: " << endl << matL << endl;
    cv::Mat Xt = matX.t();
    cv::Mat matDotProds = Xt * matL;

    for (int i=0; i<4; i++) {
        if(fabs(matDotProds.at<double>(i,0)) < 1e-6) {
             cout << "Point " << i << " is on the line." << endl;
             cout << "Non-Homogenous Point: (" << Xt.at<double>(i,0)/Xt.at<double>(i,2) << ", " <<
        }
        else {
             cout << "Point " << i << " is not on the line." << endl;
        }
    }
    return 0;
}
```

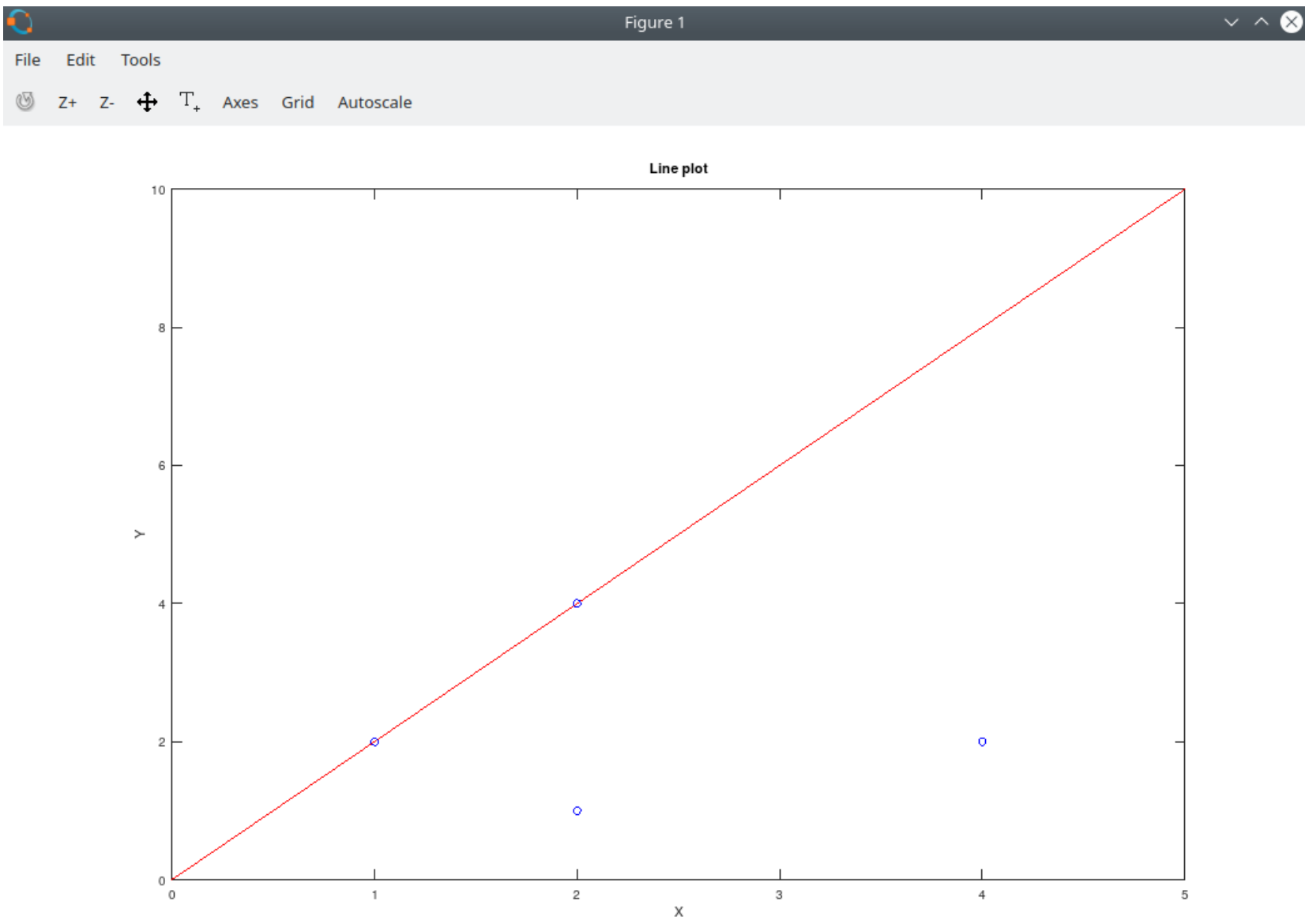
In [ ]:
```
# Python Implementation

import cv2
import numpy as np

if __name__=="__main__":
      X = np.array([[2,6,1,16],[4,3,2,8],[2,3,0.5,4]])
      L = np.array([[8],[-4],[0]])
      Xt = X.T
      results = Xt @ L
      print(Xt)
      print(L)

      for i in range(0,4):
            if results[i] < 1e-6:
                  print(f"The point {i} is on the line.")
                  print(f"In-homogenous point is ({Xt[i,0]/Xt[i,2]},{Xt[i,1]/Xt[i,2]})")
            else:
                  print(f"The point {i} is not on the line.")
```



Exercise 2

In [ ]:
```
% Data
X=[2, 4, 2; 6, 3, 3; 1, 2, 0.5; 16, 8, 4];
X=X'
```

```
L=[8; -4; 0];

result = X'*L

% Plot the line
p1x=0;
p1y=(-L(1)*p1x - L(3))/L(2);

p2x=5;
p2y=(-L(1)*p2x - L(3))/L(2);

plot([p1x, p2x], [p1y, p2y], 'r-')

% Plot the points
hold on;
for i=1:4
   plot([X(1,i)/X(3,i)], [X(2,i)/X(3,i)], 'bo')
end

title('Line plot')
xlabel('X')
ylabel('Y')

waitfor(gcf)
```




```
(4.8031, 9.433)
```

Exercises to do on your own

Here's a very basic mobile robot I built with scrap metal, two DC motors from cordless drills, 3D printed motor mounts, miscellaneous hardware and caster wheels, hobby RC brushed motor speed controllers and radio receiver, and an Arduino for Nano for control.

For vision, I added an NVIDIA Jetson Nano Development Kit (https://th.rs-online.com/web/p/processor-development-tools/1999831/?cm_mmc=TH-PLA-DS3A-_-google-_-PLA_TH_THB_Fallback-_-All+Products-_-1999831&matchtype=&aud-827186183886:pla-293946777986&gclid=Cj0KCQjw--GFBhDeARIsACH_kdbBNFCyMXmHOx-PvOxQZwIi09Re8pic9SLaIQWj3FbCVZr0hEwK-FgaArvUEALw_wcB&gclsrc=aw.ds) and an 8MP camera for the Jetson Nano (https://www.lazada.co.th/products/8mp-imx219-jetson-nano-200-fov-3280x246415-i2298839807-s7747540313.html?spm=a2o4m.searchlist.list.1.547e4cc8uIGag2&search=1&freeshipping=1).

We will use something similar but a bit smaller and easier to assemble, the Jetbot (https://jetbot.org/master/), to experiment with autonomous control using computer vision!

Download this video acquired from the mobile robot (https://drive.google.com/file/d/1K2EjcMJifDUOkSP_amlg8wcHmv_jh44V/view?usp=sharing).

1. Figure out how to read and display the video in an OpenCV window (using C++!). You may find the `VideoCapture` tutorial (https://docs.opencv.org/4.5.5/d8/dfe/classcv_1_1VideoCapture.html) useful.
2. Do the same thing using OpenCV from Python.
3. Get the sparse optical flow example from the OpenCV optical flow tutorial page (https://docs.opencv.org/4.5.5/d4/dee/tutorial_optical_flow.html) working on this video. Try both C++ and Python.

The report

Turn in a brief report in PDF format describing your experience in the lab and the results of the two sets of exercises using Google Classroom before the next lab.

Exercise 1

C++ and Python Code for reading a video file and displaying it using OpenCV.

In [ ]:
```
#include <opencv2/core.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <iostream>
#include <stdio.h>

using namespace cv;
using namespace std;

int main(int, char**)
{
    Mat frame;
    VideoCapture cap;
    // char **filename = "Lab01-robot-video.qt";
    int deviceID = 0;
    int apiID = cv::CAP_ANY;
    // cap.open(deviceID, apiID);
    cap.open("../Lab01-robot-video.qt", apiID);

    if (!cap.isOpened()) {
        cerr << "Error! Unable to open file\n";
        return -1;
    }

    cout << "Start grabbing" << endl << "Press any key to terminate" << endl;
```

```
    for(;;) {
        cap.read(frame);
        if (frame.empty()) {
             cerr << "Error! Blank frame grabbed.";
             break;
        }

        imshow("Live", frame);
        if (waitKey(5)>=0) {
             break;
        }
     }

     return 0;
}
```

Exercise 2

In [ ]:
```
import cv2
import numpy as np

if __name__=="__main__":
     cap = cv2.VideoCapture("./Lab01-robot-video.qt")
     if (cap.isOpened() == False):
          print("Error opening Video file")

     while (cap.isOpened()):
          ret, frame = cap.read()
          if ret==True:
               cv2.imshow("Frame", frame)
               if cv2.waitKey(25) & 0xFF == ord('q'):
                    break
          else:
               break
     cap.release()
     cv2.destroyAllWindows()
```

Exercise 3

OpticalFlow Video for Lab - https://youtu.be/6O_EoVOHI8o (https://youtu.be/6O_EoVOHI8o)

Optical Flow is the pattern of apparent motion. Optical flow calculates a 2D Vector field, where each vector is a displacement vector showing the movement of points from one frame to the next. Application areas of Optical Flow include 1) Structure from Motion 2) Video Compression 3) Video Stabilization

There are two assumptions behind the optical flow algorithm.

1) Pixel intensities of an object donot change between consecutive frames.
2) Pixels neighboring each other have similar motion

Let us consider a pixel with intensity $\mathbf{I}$, which moves by a distance of $\mathbf{dx}$ and $\mathbf{dy}$ between consecutive frames. The time taken for transition between one frame to the next is denoted by $\mathbf{dt}$.

Since the Intensities of the object pixels doesnot change between consecutive frames, as per our stated assumption, we can write,

$$\mathbf{I(x, y, t) = I(x + dt, y + dy, t + dt)}$$

Using Taylor series, we get

$$f_x\mathbf{u} + f_y\mathbf{v} + f_t = 0$$

$$f_x = \frac{\partial f}{\partial x} \; ; \; f_y = \frac{\partial f}{\partial y}$$

$$u = \frac{\partial x}{\partial t} \; ; \; v = \frac{\partial y}{\partial t}$$

Where f_x and f_y are image gradients which can be calculated; and f_t is the gradient along time dimension. u and v are unknowns which need to be calculated using the **Lucas-Kanade Method**.

Lucas-Kanade Method

A stated assumption behind the optical flow algorithm is the similarity of motion among neighboring pixels. Lucan-Kanade method uses this assumption by taking a 3x3 patch around the central point and assuming similarity of motion between all the nine pixels.

All of these nine points have a common movement vector, denoted by **(u, v)**. We get nine equations which can be solved to get the movement vector for the selected patch. OpenCV calculates optical flow using the least square fit method.

Lucas-Kanade method works only for small motions. In case of large motions, image pyramids are used to convert large motions to small ones and a scaling factor is used to adjust accordingly.

In [ ]:
```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
```

```
#include <opencv2/videoio.hpp>
#include <opencv2/video.hpp>

using namespace cv;
using namespace std;

int main()
{
       string filename = "../Lab01-robot-video.qt"; // Name of video file to open
       VideoCapture cap(filename);     // declaring a capture object to grab video frames
       if (!cap.isOpened()) {          // check if capture object can access the file
            cerr << "Unable to open file!" << endl;
            return 0;
       }

       // Create some random colors
       vector<Scalar> colors;
       RNG rng;
       for (int i=0; i<100; i++) {
            int r = rng.uniform(0,256);
            int g = rng.uniform(0,256);
            int b = rng.uniform(0,256);
            colors.push_back(Scalar(r,g,b));
       }

       Mat old_frame, old_gray; // Declare matrices to hold previous frames (color and grayscale)
       vector<Point2f> p0, p1;  // Declare two point vectors (float dtype)

       cap >> old_frame; // Copy from video stream to old frame
       cvtColor(old_frame, old_gray, COLOR_BGR2GRAY);  // convert to grayscale
       goodFeaturesToTrack(old_gray, p0, 100, 0.3, 7, Mat(), 7, false, 0.04);  // get corner featur

       Mat mask = Mat::zeros(old_frame.size(), old_frame.type());  // declare a zero matrix of same

       while(true) {
            Mat frame, frame_gray;  // Declare current frame for every loop
            cap >> frame;           // Get current frame from the video stream

            if (frame.empty()) {   // Check if frame is empty
                break;
            }

            cvtColor(frame, frame_gray, COLOR_BGR2GRAY);  // convert to grayscale

            vector<uchar> status;
            vector<float> err;     // Error vector
            TermCriteria criteria = TermCriteria((TermCriteria::COUNT) + (TermCriteria::EPS), 10, 0
            // Calculate optical flow between successive frames
            calcOpticalFlowPyrLK(old_gray, frame_gray, p0, p1, status, err, Size(15,15), 2, criteri

            vector<Point2f> good_new;
            for (uint i=0; i < p0.size(); i++) {
                if (status[i]==1) {
                    good_new.push_back(p1[i]);
                    line(mask, p1[i], p0[i], colors[i], 2);
                    circle(frame, p1[i], 5, colors[i], -1);
                }
            }

            Mat img;
            add(frame, mask, img);
            imshow("Frame", img);

            int keyboard = waitKey(30);
            if (keyboard == 'q' || keyboard == 27) {
                break;
            }

            old_gray = frame_gray.clone();
            p0 = good_new;
       }
}
```

In [ ]:
```
import numpy as np
import cv2 as cv
import argparse

VIDEO_FILE = "Lab01-robot-video.qt"

cap = cv.VideoCapture(VIDEO_FILE)
# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 100,
                       qualityLevel = 0.3,
                       minDistance = 7,
                       blockSize = 7 )
# Parameters for lucas kanade optical flow
lk_params = dict( winSize  = (15, 15),
                  maxLevel = 2,
                  criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 0.03))
```

```
# Create some random colors
color = np.random.randint(0, 255, (100, 3))
# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY)
p0 = cv.goodFeaturesToTrack(old_gray, mask = None, **feature_params)
# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)
while(1):
    ret, frame = cap.read()
    if not ret:
        print('No frames grabbed!')
        break
    frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    # calculate optical flow
    p1, st, err = cv.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)
    # Select good points
    if p1 is not None:
        good_new = p1[st==1]
        good_old = p0[st==1]
    # draw the tracks
    for i, (new, old) in enumerate(zip(good_new, good_old)):
        a, b = new.ravel()
        c, d = old.ravel()
        mask = cv.line(mask, (int(a), int(b)), (int(c), int(d)), color[i].tolist(), 2)
        frame = cv.circle(frame, (int(a), int(b)), 5, color[i].tolist(), -1)
    img = cv.add(frame, mask)
    cv.imshow('frame', img)
    k = cv.waitKey(30) & 0xff
    if k == 27:
        break
    # Now update the previous frame and previous points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)
cv.destroyAllWindows()
```

Type *Markdown* and LaTeX: α^2