COMP27112
Introduction to Visual Computing
Coursework Assignment 3
Image Processing Exercise 1

Tim Morris

Introduction

The aim of this exercise is to get you started with OpenCV and to do some very simple image processing.

OpenCV (opencv.org) is an open source computer vision and machine learning library that's released under the BSD licence. It can be used freely for academic and commercial purposes. The first version was released in 2001, it now has a user community of over 47,000 users and the libraries have been downloaded over 18 million times.

Since it's a well-established and widely used library, there are a lot of resources on the internet. If you run into problems with the coursework, this should be one of the first places to seek help. Other useful websites are listed in Blackboard.

This piece of coursework is worth 7.5% of the unit's assessment, so you should spend no more than about 7.5 hours on it.

Getting Started

Follow the instructions on the Coursework area of Blackboard to download and install the version of OpenCV for your operating system. In this lab, you'll write two simple programmes, one to prove to yourself that you've installed OpenCV correctly, the second to perform thresholding.

Hello OpenCV

To check that everything is working correctly, print out the installed OpenCV major and minor versions in the console. To do that, OpenCV provides two macros representing the two integers: CV_MAJOR_VERSION and CV_MINOR_VERSION.

Use your favourite text editor to create a new C++ file. (Although this is technically a C++ file, you're using the C subset only.) Type the following code:

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

int main(int argc, char *argv[])
{
    //Print the OpenCV version
    printf ("OpenCV version: %d.%d\n", CV_MAJOR_VERSION,
CV_MINOR_VERSION);
    return 0;
}
```

Now compile it by running:

```
g++ HelloCV.cpp –o HelloCV ’pkg-config --cflags --libs
opencv4’
```

(If you copy and paste, then be aware of the problem with quotation marks.) Run the output programme, you should get:

```
OpenCV version 4.5
```

You can, of course, make editing and compiling much easier by using an IDE, instructions for doing that can be found on the web.

Using OpenCV

The first part of this exercise is to load an image file into memory and display it in a window using OpenCV. Along with the standard input/output library, and core.hpp that was included in the example you will also need to include the **highgui.hpp** headers. core.hpp provides basic OpenCV structures and operations, while highgui.hpp gives you the ability to create and work with windows to display information to the user.

First, you will need to provide the code with an image path, either by using command-line arguments, user input in the terminal or any other way that you can think of. Make sure to check that the path has been provided, if it hasn’t you should exit the program. Now that you have a path to the image, you need to load the image. For this, OpenCV has a function named **imread()** in the cv namespace which takes the path as the first argument and some flags that determine how the data is interpreted as the second (it’s safest to use IMREAD_UNCHANGED). You can find all the details in the OpenCV documentation, links are given below. This function loads the image into a **cv::Mat** object. Now that you have the image (you should check that the image is loaded, if imread fails, it will return NULL), you need a window to display it in. OpenCV has a function called **cv::namedWindow()** which does this for you. This takes the window name as the first argument and the window flags as the second argument. You want the window size to match the image size, so use CV_WINDOW_AUTOSIZE as the flag. All that’s left is to add the image to the window. **cv::imshow()** does just that. The first argument is the name of the window you want to add the image to and the second argument is a pointer to the image. At this point, if you

compile and run the program you will notice that nothing shows (try it). That is because your program terminates and closes the window before you can see anything. A useful command in this case is **cv::waitKey(0)**. The function waits for the user to press a key in order to move on with the code. The numerical argument defines how long the function will wait, look up the documentation to see how it's interpreted. Compile and run the code. A window should pop up with the title that you gave and the image.

Note that you must also include the **imgcodecs.hpp** header to use imread.

To summarise, the steps involved are:

1. Load an image using **imread()**
2. If imread has returned NULL, the reading has failed, so exit the programme.
3. Create a display window using **namedWindow**
4. Load the image into that window using **imshow**
5. Use **waitKey** to pause the execution until a key is pressed

The final step in this lab is to threshold the image and display the result. If you loaded a colour image, you must first convert it to an 8-bit grey scale image using **cv::cvtColor** which takes four arguments: the input and output images, a flag that indicates the conversion being used (use COLOR_BGR2GRAY) and the last one indicates the number of channels in the output (use 0). Thresholding is achieved using the function **cv::threshold**. You will need to include **imgproc.hpp** to use threshold.

Threshold takes five arguments: the input and output images, a value for the threshold, a maxval and a flag to indicate the type of thresholding being applied. You will probably use THRESH_OTSU as the flag – in this case the function returns the value of the threshold that's computed and maxval is ignored. If you set the flag to THRESH_BINARY, then

$$\mathrm{dst}(x,y) = \begin{cases} \mathrm{maxval} & \text{if } \mathrm{src}(x,y) > \mathrm{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Finally, save the threshold image using **cv::imwrite()**

You might want to try to add a trackbar to control the thresholding to see the effect of different thresholds.

Finally, apply the thresholding programme to the sample images that are provided.

Queries

Experiment with your code to find the answers to the following questions. Write your answers in a separate document.

1. Is Otsu's method successful in thresholding all the images?
2. How would you modify the thresholding algorithm to address any problems?
3. What metrics are there for assessing the success of thresholding?

Submission

Once you have a working solution, simply ZIP the C++ file and your answer document and submit it to the Lab 3 area in Blackboard.

Marking Scheme

Exit correctly if no valid input	2 marks
Imshow displays images correctly	2 marks
Otsu is correctly called	2 marks
Thresholded image is displayed	2 marks
Meaningful names to windows	1 mark
Trackbar	2 marks
Interactive thresholding	2 mark
Question 1	1 mark
Question 2	3 marks
Question 3	3 marks

Total	20 marks

Links to OpenCV documentation

Colour space conversion
https://docs.opencv.org/4.5.1/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab

Colour space conversion flags
https://docs.opencv.org/4.5.1/d8/d01/group__imgproc__color__conversions.html#ga4e0972be5de079fed4e3a10e24ef5ef0

Thresholding
https://docs.opencv.org/4.5.1/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57

imread
https://docs.opencv.org/4.5.1/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56

imshow
https://docs.opencv.org/4.5.1/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563

imwrite
https://docs.opencv.org/4.5.1/d4/da8/group__imgcodecs.html#gabbc7ef1aa2edfaa87772f1202d67e0ce

Trackbar
https://docs.opencv.org/4.5.1/d7/dfc/group__highgui.html#ga004b512c3d09aa53f20b050b1d01fab1