COMP27112
Computer Graphics and Image Processing
Laboratory Exercise 3
Measuring Noise

Dr Tim Morris

2018-2019

1 Introduction

This lab exercise builds on what you've learned last time, so make sure that you have a good understanding of how you load images into memory and display them.

This week, you are going to learn how to measure the level of noise in an image. There are multiple ways of doing this but for simplicity, you are going to reduce the noise in the image by blurring the image and then take the difference between the original and the blurred versions (you have to add an offset to ensure that the results are all positive, in our case we will add 128). This difference image will mostly highlight noise, so its histogram should look like a narrow peak with a mean of 128. (In greyscale, we measure "colour" as brightness with values ranging from 0 to 255.)

In theory, the original image is the "true" image plus some amount of noise and the blurred image has the noise reduced. The difference then represents the amount of noise that has been removed through smoothing. Basically you're implementing the methods we've talked about in lectures.

2 Setting up

In your COMP27112 directory, create a subdirectory called **lab3**. The code from Coursework 4 will give you a good starting point for this lab, so icopy it.

3 Let's begin

Unlike the last lab, you will not be guided through the code line-by-line. The general concept is:

1. the program should load an image,
2. convert it to greyscale (if it isn't greyscale already),
3. blur it according to some values defined by a slider,
4. compute the difference between the original and blurred images and add 128 to it (because we don't want negative values),
5. generate the histogram
6. compute the standard deviation and
7. write it in the histogram window
8. save the difference image and the histogram image.

In order to process images, you will have to include the imgproc library available in OpenCV2.

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

As a general rule, RGB images have three different channels, the values in the Mat being represented with the type CV_8UC3. Here, 8 represents the depth (8 bits = 256 values), u means unsigned (positive values), c (char) and 3 is the number of channels.

If you read a colour image (you can check this by looking at the nChannels value), you'll have to convert it to greyscale.

When you subtract the two images, you will end up with negative numbers that will be truncated to 0 (remember, the numbers are unsigned), so you will need to add 128 to the values after subtraction, but before the assignment.

One more thing you will need is to draw the histogram. This is the code to draw the histogram mat.

Listing 1: C++ code needed to print the versions

```cpp

//This method takes a histogram cv::Mat and returns a cv::Mat
// containing the actual drawing of the histogram
cv::Mat drawHist(cv::Mat hist, int scale)
{
  double mx = 0;
  cv::minMaxLoc(hist, 0, &mx, 0, 0);


  cv::Mat result = cv::Mat::zeros(256 * scale, 256 * scale, CV_8UC1);

  for(int i = 0; i < 255; i++)
  {
    //Get the histogram values
    float histValue = hist.at<float>(i, 0);
    float nextValue = hist.at<float>(i+1, 0);

    //Create 4 points for the poly
    cv::Point p1 = cv::Point(i * scale, 256 * scale);
    cv::Point p2 = cv::Point(i * scale + scale, 256 * scale);
    cv::Point p3 = cv::Point(i * scale + scale, (256-nextValue*256/mx)*scale);
    cv::Point p4 = cv::Point(i * scale, (256-nextValue*256/mx) * scale);

    //Draw the poly(Ending in p1)
    int numPoints = 5;
    cv::Point points[] = {p1, p2, p3, p4, p1};
    cv::fillConvexPoly(result, points, numPoints, cv::Scalar::all(256), 0, 0);
  }

  return result;
}
```

It might be helpful to declare the mats globally (outside of the main scope) so that they can be accessed by other functions. You may also find it useful to split your code into different functions so that you don't copy and paste code (avoid doing that as much as possible). You can create one slider to adjust the blur radius, or two sliders that adjust the blur radius and the sigma value and make sure the images update accordingly. (See the definition of GaussianBlur in the OpenCV documentation and below.)

Useful functions:

1. **cv::Mat(numberOfRows, numberOfColumns, type)** - Constructor for cv::Mat objects. The type represents the kind of values that are expected, eg.: CV_8UC1, CV_8UC3, CV_16UC1.

2. **cv::cvtColor(source, destination, conversionType)** - converts between colour spaces. The conversionType is the kind of conversion that the function needs to do, eg.: from RGB to 8-bit greyscale (CV_RGB2GRAY). You can find more types in the OpenCV documentation.

3. **cv::GaussianBlur(source, destination, kernelSize, sigmaX, sigmaY)** - Applies a Gaussian blur operation to the source mat, saving the output to the destination mat. The kernel size is represented by a **cv::Size** object. The sigmaX and sigmaY are the gaussian standard deviations, requiring odd values. If the sigmaY is 0, it will be set to the same value as sigmaX, if both are zero their values are computed from kernelSize.width and kernelSize.height.

4. **cv::calcHist(sourceAddress, noImages, channels, mask, destination, dimensions, sizes, ranges)** - Calculates all the details of a histogram for an image. The sourceAddress is the memory address of the image. noImages is the number of images at that address. The channels is an integer array, indexing each channel in an image. The mask is a cv::Mat object (can be an empty mat for no mask). The destination object is a Mat that will hold the histogram data. The dimensions is an integer representing the number of dimensions for the values in the image. The sizes is an array of integers representing the size of each dimension. The ranges object is a matrix of floats, each column representing the min and max values for each dimension.

5. **cv::createTrackbar(sliderName, windowName, valueVariable, maxValue, callback, userdata)** - Creates and adds a slider to the specified window. The valueVariable must be an integer pointer - this is the value that gets changed when the slider moves. The parameter maxValue is an integer that sets the maximum value that the slider can be set to. The callback is a function that needs to be created separately - this will be called every time the slider changes its value. The callback must be a void and have only two parameters (int pos, void* userdata) which is the current value of the slider. The userdata can be NULL.

4 Submit

You now have one code file and two images. Zip them and **submit**.

5 Marking scheme

Read image and convert to greyscale	2 marks
Correct blurred image	2 marks
Correct slider	2 marks
Correct noise image	2 marks
Draw correct histogram	2 marks