Image Processing Practical 1

Aims: A first program using OpenCV; familiar with operations on images.

1. Create a new OpenCV project in Visual Studio using the instructions in note and then
2. Load an image and show the image (download the 'Lena') – see example in Page 15 of the note.
3. Look up the entry for the **GaussianBlur** function in the OpenCV manual. Explore the effects of varying either of the mask (i.e. kernel) parameters from 5.

   Try larger and smaller values (which must always be odd). What happens? How does this affect the direction of blurring?
4. Investigate the use of the following OpenCV functions by replacing the GaussianBlur() operation in the above code:

   - **flip()**
   - **canny()**

   You will need to first convert the image to grayscale using:

   ```
   cvtColor(inputImage, grayImage, CV_BGR2GRAY);
   ```

   N.B. Suitable threshold parameters to choose for the canny operator would be 10 and 200. You can leave the other parameters at the suggested defaults from the manual (you may wish to vary the apertureSize if you wish also).

   - **resize**()

   N.B. You can just specify fx and fy resize scales and let the function compute the required destination image size.

   Alternatively you can use *Mat::create()* a method within the OpenCV C++ image object *Mat* to create an empty image of a given size as the destination then set *fx* and *fy* to 0 to have the scaling factors automatically calculated to fit into the destination image.

   Investigate the effects of changing type of interpolation used (for example between CV_INTER_NN and CV_INTER_LINEAR.

   Use the OpenCV manual for reference. Experiment with different parameters and different images.

Image Processing Practical 2

Aims: Interacting with image data

1. Repeat the example shown in Page 20-22 of the note.
2. Change the above code so that a right mouse click changes the current pixel colour to:

    a. Black
    b. Red
    c. Green
    d. Blue
    e. Light Gray
    f. Dark Gray
    g. Yellow
    h. Purple

    Record the numerical colour values that you use.

3. Change the above code to fill the 5x5 pixel neighbourhood red around the location at which the mouse is clicked.
4. Extract the 100x100 pixel area around the location of a mouse clicks (i.e. "Region of Interest" - ROI) and create and display this sub-part of the original image in a separate (new) window. You may find the Mat class method Mat(const Mat& m, const Rect& roi) and the region of interest examples in the C++ Cheatsheet section of the manual helpful.

    Beware of the use of pointers inside the C callback function:

    Mat roi = Mat ((Mat *)img, Rect (...));*

5. Use the function cvtColor() to adapt the program to first convert the colour image loaded to grayscale using the built-in OpenCV conversion. Set the code paramete of this function as CV_BGR2GRAY to cope with the BGR channel ordering.

    Adapt the pixel access using the Mat::at() method to now report the single channel colour values (see previous page).

6. Investigate the use of the rectangle() function (Drawing Function section of core section of manual) to draw a 10x10 rectangle around a image point selected with a mouse click. You will need to investigate the OpenCV datatype Scalar to specify an (BGR) colour in which to draw it.

Image Processing Practical 3

Aims: Writing images to file

1. Repeat the example shown in Page26.
2. With reference to the manual entry for imwrite() experiment with changing both the CV_IMWRITE_JPEG_QUALITY parameter when saving JPEG ".jpg" images and with CV_IMWRITE_PNG_COMPRESSION when saving PNG ".png" images. What do you notice about the image quality and file sizes ?
3. Use the provided program to invert a photographic image (e.g. "original.jpg") and save the result out to file in JPEG format (call the result "inverted.jpg"). Now use this inverted image as a new file input to the same program to produce a secondary output image "inverted_back.jpg" (again use JPEG).

   Now onvestigate the use of the absdiff() function to write your own program that computes the difference between one image and another and displays the result. Try running this program on your original image ("original.jpg") and the image "inverted_back.jpg". You should see just a black image (as the inverted back image looks the same as the original) but are all the pixel values actually all zero. Use the earlier examples of accessing pixel elements to investigate this.

   You could also try using the equalizeHist() function to visualise the difference of the two images but first you will need to convert this "difference image" to grayscale as equalizeHist() only operates on grayscale images.

   Do the same but this time using PNG format images. What is the difference? Why does this occur?

Image Processing Practical 4

Aims: Capture image and video from a camera

1. Capture image from a camera following the instructions in 1.4.4
2. Capture a video from a camera following the instruction in 1.4.5
3. Experiment with the image processing functions from earlier exercises to display live "processed" video. Try the following:
   - GaussianBlur() to produce a live blurred video
   - bitwise_not() to display a live inverted video
   - canny() to display live edge detection

   Remember you need to do a grayscale conversion first for canny() like last time.
4. Extend the above program to display only the difference between subsequent image frames captured from the camera (Hint: use absdiff()).
5. Extend the above program to save a captured video frame to file when the key "c" is pressed by the user.
6. Extend the above program to capture a single of frames (to numbered files) every:
   - 3 seconds
   - 10 seconds
   - 1 minute

   What would be a sensible choice of image file format given that we may now be collecting a large quantity of data?
   alternatively look at the getTickCount() and getTickFrequency() OpenCV functions)
7. Use your program to capture two successive frames of a static scene in your work area (i.e.without changes in camera position or any movement in the scene).
   Based on your earlier program to compare the original and inverted images use absdiff() to compute the difference between these two images.
   What do you expect the result to be? What is the actual result that occurs? Is there really no difference in the images? - use the earlier program example of querying individual pixel elements to investigate.
8. Repeat task 7 but this time use two captured image frames where you have changed the lighting conditions in the scene. You don't need to turn the lab lights off!!! - just cast a body shadow over the scene (staying out of shot) or shine a mobile phone screen/light into the shot.
   How does this result differ from that in task 5?
9. Use circle() to draw a circle overlay around the centre of your live image capture.
   N.B. Note that the y-axis appears to be inverted for drawing functions following the computer graphics convention of a bottom-left origin.

Image Processing Practical 5

Aims: Writing video to file

1. Practice the example shown in 1.4.6
2. Extend your programs from Tasks 3 and 4 of the Practical 4 on capturing live video from a camera to write these results to a video file.
   Beware of creating large files.
3. Investigate the use of the VideoCapture() object in OpenCV to alter your earlier programs so that instead of operating on images captured from a camera they now operate on images frames from a video file.

Image Processing Practical 6

Aims: Image representation

1. Write you own grayscale quantization program that loads a grayscale image and reduces the number of gray levels to a specified number.

   Don't consider changing the bit representation, just the number of levels actually used.

2. Write your own image measuring program that outputs the Euclidean distance between two points in a displayed image.

   Use the mouse to select the first and second point.

3. ***If you have time:*** Experiment with the OpenCV resize() functions (with different interpolation routines) and dilate() / erode functions with different parameters. We will discuss erosion and dilation later in the IA courses.

Image Processing Practical 7

Aims: Image Transformation

1. Use the webcam to display a difference image between consecutive live video frames that relate to what has changed within the scene.

   You may want to experiment with changing the interval between the captured frames that are differenced.

2. Use the webcam to display a "vapour trails" type image by blending the current image frame with the previous N image frames.

   Experiment with equal weights between images and weights that decrease with the age (in time) of the frame (but ensure your weights always sum to 1).

3. Use the webcam to capture a picture of a dark object on the plain desk background (e.g. pen, keys, mobile phone). Investigate producing the grayscale histogram of this image and selecting a suitable threshold for foreground and background separation in a binary image.

   Using your signal processing knowledge and experience determine a method to automatically extract a suitable threshold from the histogram.

4. Develop your answer to exercise 3 to use adaptive thresholding.

   How well does it adapt to changing lighting conditions?

5. Use the webcam to capture a live image and display the amplitude spectrum in real-time.

   Investigate using low-pass filtering in Fourier space to (look at examples in notes) to produce a blurred image, displayed in real-time from the camera.

   Compare the speed of this with your earlier spatial convolution based approach.

6. *If you have time:* Use the webcam to develop a real-time colour-slicing technique that isolates a particular colour range in the scene (by thresholding the H channel of the HSV image representation) and overlays the corresponding H (or RGB value) onto the otherwise grayscale image?

   There are various stages to this (in brief summary, to help you out):

   - Convert captured frame to HSV space.
   - Threshold H channel of HSV image.
   - Convert captured frame to grayscale.
   - Produce an RGB output image where:
     - If thresholded H pixel == 0 then RGB output channels = grayscale channel
     - Else RGB output channels = corresponding input pixel

   This "chromo-keying" technique is a common video effect (common in 1980s music videos!)

Image Processing Practical 8

Aims: Image Enhancement

1. Investigate the use of histogram equalisation in OpenCV and apply it to live images captured from the webcam.

2. Implement contrast stretching in OpenCV and compare (visually and experimentally) the different effects this operator has on an image compared to your earlier histogram equalisation implementation.

3. Expand your earlier histogram equalisation implementation to perform histogram specification from:

   - a separate source image
   - a user selected region of the current image

4. From the range of techniques specified (in this chapter and in others) identify the following:

   - A suitable technique for noise removal for the webcam captured images.
   - A suitable technique for edge enhancement for the webcam captured images.

   Implement each separately and evaluate its performance:

   What are the noise characteristics of these cameras?

   What techniques are suited to these characteristics?

5. Implement the adaptive ***unsharp*** filtering using the kernels specified in the notes.

   Evaluate its performance against conventional ***unsharp*** filtering.