Computer Vision – Lab Exercises

Lab-0: Basic Operations (IPA course refresher)

- Read in a colour image file of your choice.
- Remove a 100x100 pixel section from the middle of the image and display this in a new window.
- Invert the removed section and reinsert into the original image.
- Save the resulting image to storage with a new filename.

Lab-1: Image/Video Compression

- Checking the quality of compression
  - Implement MSE and PSNR error metric functions and apply them to an image and the same image that has been compressed with jpeg algorithm.
  - Repeat your tests on the “compression example images” on Canvas. Do you see a correlation between compression error and content?
  - Try varying the quality of the JPEG output (IMWRITE_JPEG_QUALITY) and note the change in error metrics.
  - Try saving the files in PNG format and varying the compression level. What can you say about the errors when you compare the original input images and the output images?
  - Divide your images into sections and calculate the error metrics for each sub-image. Find the mean and standard deviation for each complete image. Apply this to the previous test images and investigate how the values change with different image content.
- Run Length Encoding (RLE)
  - Implement a lossless compression method using Run Length Encoding to encode your input image. Convert your input image to greyscale before applying the RLE compression.
  - Think about how you will represent the sequence of similar values as a data structure. One approach would be to use C++ containers to store a run of similar values for each line. The encoded file will consist of a container for each line:
    - Use a pair data structure to capture a sequence of identical values. This will include the greyscale value of pixels in the sequence and the number of identical values in the sequence.
    - Use a vector data structure to capture all of the sequences in a single line.
    - Use an additional vector data structure to represent all of the lines in the image.
  - You will also need to think about how to serialise the encoded data structures to disk and how to read them back into your program.
  - Compare the size of the compressed file with the original, remembering to use a binary representation to improve efficiency of the file storage.

Lab-2: Image Restoration - filtering

- Use the Gaussian, Median and Bilateral filters to blur an image. Show the output from each filter side-by-side with the original image. Compare the results.

- Write a direct inverse filtering function and use the Gaussian blurred image as an input for testing.
    - Check the difference between the restored image and the original by using the MSE/PNSR metrics. Are you able to see a difference by inspection?
    - Try different levels of Gaussian blurring to check how effective the restoration technique is.
    - Add noise to the input image and investigate the performance of the inverse filtering technique.
- The Lucy-Richardson algorithm is a common technique used in image processing for deblurring images where you have limited knowledge about the original blurring of the image. Implementing this algorithm in OpenCV using C++ involves several steps. Below is a basic guide for you as a reference:
    - **Include Necessary Headers:** Begin by including the necessary OpenCV headers and the standard C++ headers.
    - **Read the Image:** Load the image that you want to deblur.
    - **Define the Point Spread Function (PSF):** The Lucy-Richardson algorithm requires initial knowledge of the point spread function (PSF) that describes the blurring. This could be a known function or estimated from the image.
    - **Implement the Lucy-Richardson Algorithm:** Implement the iterative process of the Lucy-Richardson algorithm. This involves estimating the latent image and updating it iteratively.
    - **Display or Save the Result:** After the iterations, display or save the deblurred image.

Lab-3: Morphological Operators

- Investigate the use of Dilation and Erosion operators using the openCV functions or coding your own functions.
- Following the Lab-1, investigate the use of Opening and Closing operators using the openCV functions or coding your own functions.
- Investigate the use of Thinning and Thickening operators using the openCV functions or coding your own functions.
- Count the number of the cells in the image that can be downloaded from Canvas (cell.jpg)

Lab-4: Feature Processing - Edge, Regions

This tutorial builds on the work we looked at in the CV course regarding edge feature detection. First re-read the relevant section of the CV course notes to re-familiarise your-self with this topic. All of the code examples in this tutorial are available in Canvas.

- Download and compile the example `generic_interface.cpp` from the examples in Canvas. (Using instructions for Visual Studio as per IP course notes Chapter 1).
    - It would be useful to use a webcam for this exercise but if you don't have access to one, the same code can be used to load and process a video file instead. The example code is a generic interface to cameras/images/videos, all contained in a single course file.

- Add Canny edge detection to this example using the `canny()` operator from the OpenCV manual. By default, the OpenCV implementation of canny edge detection does not do the first stage of Gaussian smoothing.
- Investigate `GaussianBlur()` operator in OpenCV to add this as a pre-processing stage to your canny detection to obtain stable edges within the scene. You may wish to look at using the `createTrackbar()` operator to create window trackbars for your varying your parameters (an example is shown in the butterworth_lowpass.cpp example).

- Based on your work for part 1 investigate the use of the `HoughLines()` and `HoughLinesP()` operators for the detection of straight lines in the image. The earlier C example hough_lines.cc may help you with some aspects of this (esp. drawing the lines) but you will need to convert the OpenCV C operators code into C++ operators and look at the differences (in the manual) between the two interfaces. Can you obtain useful straight edges from an object?
- Based on your work for part 1 investigate the use of the `FindContours()` operators for the detection of connected contours within the image (use the version without the hierarchy parameter). Experiment with approximation method parameters (but use CV_RETR_LIST mode for simplicity). You may find the `drawContours()` operator similarly useful. Instead of `canny()` you may find it easier to do adaptive thresholding on the input image to this approach (`adaptiveThreshold()`). Can you extract the contour with the largest area `contourArea()` and either a) draw it a different colour (perhaps filled in) or b) separate it out into another image (hint: `minAreaRect()`) ?
- From your work in part 3 and with reference to the "collection mode" and "recognition mode" of the histogram_based_recognition.cpp example you encountered earlier construct a program that extracts the Hu moments (`HuMoments()`) from the largest contour in a sample image and then (in recognition mode) compares this vector of moments to those of the largest contour in the current image. When is it seeing the same contour (object)? How are you doing the comparison? Is there a better way to do it?
- Investigate the use of Mean Shift Segmentation using the `pyrMeanShiftFiltering()` operator from OpenCV as an input to your earlier C++ examples.

Lab-5: Feature Processing - Histograms

This tutorial builds on the work we looked at in the IP course regarding using statistical histogram comparison for the recognition of objects and scenes. First re-read section 4.2.5 of the IP course notes to re-familiarise your-self with this topic.

- Download and compile the example `histogram_based_recognition.cpp` from the examples in Canvas. (Using instructions for Visual Studio as per IP course notes Chapter 1). You can use a webcam to capture live video but an existing video file will work just as well. This example first stores several grayscale image histograms ("Sample Collection") mode and then (by pressing "m") moves to recognition mode whereby any new selected image is recognized (pressing "r" for recognition) based on closest match to the set of stored image histograms. Remind yourself how it

works and try it on some example images. Make a copy of the executable file (and the code) for later comparison.

- Using this example (`histogram_based_recognition.cpp`) and additionally the `sobel()` operator from the OpenCV manual alter the histogram based recognition to now perform histogram comparison based recognition based on the histogram of the Sobel edge detector response. Compare its performance to the earlier grayscale colour-based version of histogram comparison-based recognition from stage 1. How does it compare? Where does it perform better or worse?

- Now using the example histogram_based_recognition_colour.cpp as a base repeat stage 2 for R,G, B colour but now using the Sobel response on each of the R, G, and B colour channels. Compare its performance to the earlier grayscale colour-based version of histogram comparison-based recognition from stage 1. How does it compare? Where does it perform better or worse?
- Investigate the use of the `divide()` operator to investigate building histograms of gradient orientations ? (e.g. dividing Gy/Gx or similar and computing the tangent to get the angle at every point) ? (for Gx / Gy components see the xorder / yorder parameters of `sobel()` How well does this work for recognition compared to the earlier approaches ?
- ADVANCED: Investigate how `histogram_based_recognition_colour.cpp` combines the colour information in the measure.
    - Can you produce a version that combines both colour channel and Sobel edge or gradient orientation information? Does it perform any better than before?
    - Look at the `cvtColor()` operator to convert RGB to HSV colour (or BGR to HSV! Remember BGR channel ordering). Produce a version of `histogram_based_recognition_colour.cpp` that uses HSV colour with and without sobel edge information. How do they perform in comparison? (N.B. what are the HSV value ranges?)
    - Would it make sense just to use a subset of the HSV channels? Try it and see.