COMP27112: Visual Computing

Lab 4

Lecturer: Terence Morley

1 Introduction

For this practical assignment you should use C/C++ and OpenCV to develop the code. Your code, results and comments **MUST** be submitted in a **single PDF file**. Only submit the PDF file.

You should use the supplied images for your processing and include them in your report.

For ease of marking, please lay out your report in sections using the titles given in this document.

You will probably need to refer to the OpenCV documentation website: https://docs.opencv.org/4.8.0/.

2 Intended Learning Outcomes

By the end of this assignment, you should be able to:

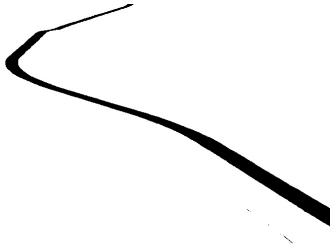
- Implement image processing code using C/C++ and OpenCV
- Create an image processing function that you can add to your own library
- Choose combinations of techniques in order to solve an image processing problem (that is, a image processing pipeline, or workflow)

3 Image Histogram and Segmentation [10 Marks]

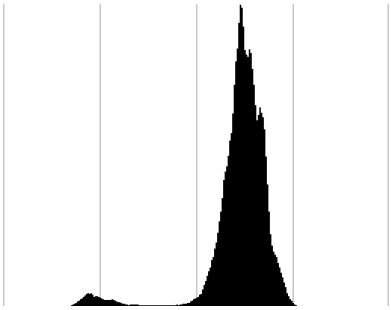
3.1 Histogram

Image segmentation is the process of partitioning an image into distinct parts (regions or objects). Thresholding is a simple way to do this, and the result is a binary image (that is, one that consists of two levels: black and white for example).

Imagine that you want to write software to help a drone navigate in a desert by following roads. You might want to segment the drone camera image into sand and road. This is shown in the following image where sand is labelled white and the road is labelled black.



The grey-level threshold that was used in the above thresholding problem was 110. But how do you decide which threshold value to use? It is often done by examining a histogram of the image, as shown below.



In this histogram image, a vertical bar is drawn in black to represent the number of pixels in the image with a particular grey level (0–255). The grey vertical gridlines mark 0, 64, 128, 192, and 255. Two peaks can be seen in this image: a large one at the lighter end (the sand), and a small one at the darker end (the road). A suitable threshold value, between the two peaks, can be estimated at around 110.

★ TASK

Write a function that creates a histogram image like the one shown above. The image must be 400 pixels high and 512 pixels wide. Because it is 512 pixels wide and the horizontal axis should represent 256 grey levels (0–255), each bar should be two pixels wide. You can use the supplied `histogram.cpp` as a starting point.

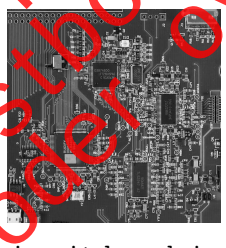
The highest count in the histogram should be drawn to the full height of the image. If the count for a grey-level is zero, no bar should be drawn.

You **must** write your own code to count the number of occurrences of each grey level, and to scale the counts to fit the image. You must **not** use the OpenCV `calcHist()` function nor something similar from other libraries.

You can draw the bars by setting pixels in the image or by drawing lines or rectangles with OpenCV functions.

Drawing the gridlines is *optional*, but if you do add them, it will make your function more helpful. The gridlines in the above image were drawn in light grey to avoid them being mistaken for bars, and the bars were drawn over the top of the gridlines (that is, the gridlines were drawn first).

Your **REPORT** should contain the source code for your histogram function and the histogram image for these two supplied images:



`circuit_board.jpg`

`science_person.jpg`

NOTE If you use LaTeX to write your report, you might include your code using the `listings` package:

```
\usepackage{listings}
...
\begin{lstlisting}[language=C++]
int x = 123;
\end{lstlisting}
```


3.2 Thresholding

For this part, you should use your histogram function to help you choose suitable threshold values. If you failed to do that part of the assignment, you can use an image manipulation program such as **gimp** that provides a

histogram display. The gimp program is available for Linux, Windows and macOS.

You may also use the programs that you created in the previous lab for doing OTSU thresholding and interactive thresholding.

★ TASK

Choose an appropriate threshold value for each of the following problems, and give a brief description of any problems encountered or any observations.

Filename	Problem
`fundus.tif`	This is taken from a set of images used to train and test algorithms for recognising the effects of diabetes on the retina. The aim of the processing is to identify the blood vessels
`glaucoma.jpg`	This is an image taken from a set used to train opthalmologists to recognise glaucoma. The aim of processing is to find the diffuse bright region towards the middle and the brighter area inside it.
`optic_nerve_head.jpg`	This was an image captured by a bespoke device that gives a tightly framed image of the optic nerve head. The aim of processing is to find the outlines of the large, slightly bright area and the smaller brighter area inside it.
`motorway.png`	This is an image from the internet of a motorway destination sign. Car manufacturers have deployed systems that read speed limit signs. A next step would be to read these signs. So the aim would be to find the white text.

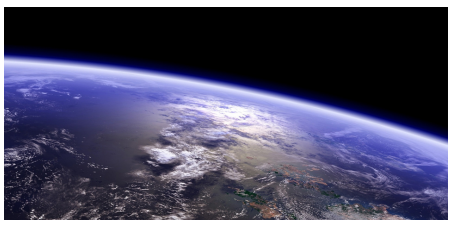
In your **REPORT**, include the following for each problem:

- the original image
- the image histogram
- thresholded image
- threshold value that was chosen
- a brief description of any observations you made

4 Horizon Detection [10 Marks]

This section asks you to find, and plot, a polynomial that represents the horizon in the following images:

horizon1.jpg



horizon2.png

horizon3.jpg

4.1 Processing Pipeline

You will need to do some form of edge detection that finds the edge between the Earth and space in two of the images, and sea and sky in the final one. You might think that you could do binary thresholding first and then

do edge detection, but you will find that there are lots of areas of misclassification that you will need to deal with.

This lab asks you to use the Canny edge detector and the Hough line transform in your processing pipeline. You should attempt to process the images in the following way:

- Convert the image into greyscale (if necessary)
- Apply a Canny filter on the image, leaving us with an image of the edges
- Apply a probabilistic Hough transformation that will return a list of pairs of Points defining the start and end coordinates for line segments.
- Filter out the short lines, use Pythagoras to compute the lines' lengths.
- Filter out the vertical lines. You could do that by either calculating the inverse tangent of each line (use `atan2`), finding its angle from the horizontal, or check whether the x co-ordinates of the segment's endpoints are similar.
- Now that you are left with all the (nearly) horizontal lines' points, find a curve that best fits all those points. This is called polynomial regression. It takes some points and calculates the best polynomial of any order that you choose that fits all the points. Be careful not to overfit the points though; since the horizon curve best matches a quadratic function choosing a higher order polynomial can give you unstable results, i.e. a very wavy line.

> Your program will need to use a number of parameters. It would be ideal if a single set of parameters could be used to process all images of this type, but that often isn't possible.
> You should allow for your program to use different parameter values. This might be achieved by passing them in as command-line parameters, using a switch-statement, or just by using three set of constants, two of which being commented out for each image.

NOTE You are supplied with some code to help you with this lab, see `horizon.cpp`. The contents of this file are shown below.

The `fitPoly` function, shown below, accepts a list of points. It calculates a line (curve) of best fit through those points. You also specify an order for the poylonimial, n (if you are expecting a straight line, you would set n to 1, for example). The function returns the polymomial as a vector of doubles, the order of which is the order of coefficients: $a + bx + cx^2 + ...$ You might want to set up a vector with a few points and try out this function so that you are clear on its operation.

```
1  //Polynomial regression function
2  std::vector<double> fitPoly(std::vector<cv::Point> points, int n)
3  {
4    //Number of points
5    int nPoints = points.size();
6
7    //Vectors for all the points' xs and ys
8    std::vector<float> xValues = std::vector<float>();
9    std::vector<float> yValues = std::vector<float>();
10
11   //Split the points into two vectors for x and y values
12   for(int i = 0; i < nPoints; i++)
13   {
14     xValues.push_back(points[i].x);
15     yValues.push_back(points[i].y);
16   }
17
18   //Augmented matrix
19   double matrixSystem[n+1][n+2];
20   for(int row = 0; row < n+1; row++)
```

```
  {
    for(int col = 0; col < n+1; col++)
    {
      matrixSystem[row][col] = 0;
      for(int i = 0; i < nPoints; i++)
        matrixSystem[row][col] += pow(xValues[i], row + col);
    }

    matrixSystem[row][n+1] = 0;
    for(int i = 0; i < nPoints; i++)
      matrixSystem[row][n+1] += pow(xValues[i], row) * yValues[i];

  }

  //Array that holds all the coefficients
  double coeffVec[n+2] = {}; // the "= {}" is needed in visual studio, but not in Linux

  //Gauss reduction
  for(int i = 0; i <= n-1; i++)
    for (int k=i+1; k <= n; k++)
    {
      double t=matrixSystem[k][i]/matrixSystem[i][i];

      for (int j=0;j<=n+1;j++)
        matrixSystem[k][j]=matrixSystem[k][j]-t*matrixSystem[i][j];

    }

  //Back-substitution
  for (int i=n;i>=0;i--)
  {
    coeffVec[i]=matrixSystem[i][n+1];
    for (int j=0;j<=n+1;j++)
      if (j!=i)
        coeffVec[i]=coeffVec[i]-matrixSystem[i][j]*coeffVec[j];

    coeffVec[i]=coeffVec[i]/matrixSystem[i][i];
  }

  //Construct the vector and return it
  std::vector<double> result = std::vector<double>();
  for(int i = 0; i < n+1; i++)
    result.push_back(coeffVec[i]);
  return result;
}
```

As part of this lab, you will be asked to draw the detected horizon from the polynomial onto the image. The function `pointAtX()`, shown below, will help you with that. If you provide it with an x-coordinate and the polynomial coefficients, it will return a point (x, y) (that is, it will calculate the y-coordinate for you and return it as a point that you might use in an OpenCV function).

```
//Returns the point for the equation determined
//by a vector of coefficents, at a certain x location
cv::Point pointAtX(std::vector<double> coeff, double x)
{
  double y = 0;
  for(int i = 0; i < coeff.size(); i++)
  y += pow(x, i) * coeff[i];
  return cv::Point(x, y);
}
```

★ TASK

Write a program to perform the processing pipeline described above. Use the supplied `fitPoly()` and `pointAtX()` as well as the OpenCV functions `Canny()` and `HoughLinesP()`.

You will need to experiment with the parameters for the Canny and Hough functions and other processing that you do. You will probably need different values for each image that you process, so make them easier to work with in your program. That is, don't just hard-code values into your function calls.

Make sure that you understand the purpose of the Canny parameters `lowerThreshold` and `upperThreshold`; and the Hough parameters `rho`, `theta`, `threshold`, `minLen`, `maxGap`.

Once you have obtained a polynomial that follows the line of the horizon, draw the line/curve on the original colour image. Make it stand out by drawing it in a bright colour and don't draw the line too narrow (nor so thick that it disguises an inaccurate detection). You might draw this line by setting pixels in the image, or by using the OpenCV functions `circle()` or `line()`.

Your **REPORT** should include your code.

4.2 Processing

Now that you have written your program to detect the horizon in an image, use it on the provided horizon images.

★ TASK

Run your program on each of the three supplied horizon images. You may need to choose different parameter values to get a good result in each image.

You might have trouble with noise in a couple of the images. If that is the case (and you can't select parameter values to achieve the aim), try adding some extra processing to your pipeline, but make sure that the horizon is detected accurately.

In your **REPORT**, include the following for each horizon image:

- The original image
- The Canny edge image
- The image with all of the probabilistic Hough lies drawn
- The image with the short lines removed
- The image with only the (approximately) horizontal lines
- The image with the horizon drawn
- The set of parameter values with an obvious name for the parameter
- If you needed to add extra processing, give a brief description

NOTE Draw your Hough lines and the final horizon in colour onto the original colour image.

Optional: The horizon in the oil-rig image needs rotating clockwise to make it horizontal. Calculate the angle that it needs to be rotated by (in degrees). Hint: use your calculated polynomial. Show your working and state an OpenCV function that can rotate the image. Rotate the image to make the horizon horizontal. Do you have any observations on your result?

5 Marking Scheme

3.1	Write a function to create a histogram image.	5 marks
3.1	Show histogram for the images `circuit_board.jpg` and `science_person.jpg`.	1 marks
3.2	Threshold the images `fundus.tif`, `glaucoma.jpg`, `optic_nerve_head.jpg`, and `motorway.png`. One mark for each image for supplying the output image, threshold value, and observations.	4 marks
4.1	Write code that performs the stated horizon-detection processing pipeline.	4 marks
4.2	Process the three images (horizon1, horizon2, horizon3) and draw the detected horizon from the polynomial on the original colour image. Include the requested images at the intermediate stages as well as the parameter values used. Two marks for each horizon image.	6 marks
	Total	**20 marks**

Check-list

Have you:

- Created a well-formatted report?
- Included input, intermediate, and output images with useful labels?
- Chosen a size for your images so that the details can be seen, but not so big that the document covers too many pages? (See the images in this document.)
- Included the code?

Submit your report as a **single pdf document** on Blackboard.
Do **not** include any other files or zip it – just submit a pdf.