COMP27112
Introduction to Visual Computing
Laboratory 4
Horizon Detection

Dr Tim Morris

2020-2021

1 Introduction

Today's lab exercise is your second (and last) marked lab for the image processing part of the unit. We will be taking a look at detecting the horizon in images. A number of concepts will be covered, including edge detection using the Canny transformation and polynomial regression.

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.

2 Let's dive right in

You'll have created a basic image processing programme in Lab 3, use it as a starting point for this exercise.

First and foremost, you need to load an image. For each image, you will have to go through a series of transformations to end up with a line separating the Earth from the sky. Let's take a closer look at what you need to do for each image:

1. Convert the image into greyscale (if necessary)
2. Apply a Canny filter on the image, leaving us with an image of the edges
3. Apply a probabilistic Hough transformation that will return a list of pairs of **Points** defining the start and end coordinates for line segments.
4. Filter out the short lines, use Pythagoras to compute the lines' lengths.
5. 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.
6. Now that we're left with all the (nearly) horizontal lines' points, we need to draw 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.

You'll need to save some sample intermediate images for your submission: a Canny edge image, an image with the probabilistic Hough lies drawn in, an image with the short lines removed, an image with only the (approximately) horizontal lines and an image with the horizon.

Since polynomial regression is outside the scope of this course, you have been provided with two functions that do this for you. **fitPoly** returns a vector of coefficients for the polynomial and **pointAtX** returns the point on the polynomial at a certain x position.

Listing 1: C++ code to compute a polynomial line's coefficients and points on a polynomial line

```
1
2  //Polynomial regression function
3  cv::vector<double> fitPoly(cv::vector<cv::Point> points, int n)
4  {
5    //Number of points
6    int nPoints = points.size();
7
8    //Vectors for all the points' xs and ys
9    cv::vector<float> xValues = cv::vector<float>();
10   cv::vector<float> yValues = cv::vector<float>();
11
12   //Split the points into two vectors for x and y values
13   for(int i = 0; i < nPoints; i++)
14   {
15     xValues.push_back(points[i].x);
16     yValues.push_back(points[i].y);
17   }
18
19   //Augmented matrix
20   double matrixSystem[n+1][n+2];
21   for(int row = 0; row < n+1; row++)
22   {
23     for(int col = 0; col < n+1; col++)
24     {
25       matrixSystem[row][col] = 0;
26       for(int i = 0; i < nPoints; i++)
27         matrixSystem[row][col] += pow(xValues[i], row + col);
28     }
29
30     matrixSystem[row][n+1] = 0;
31     for(int i = 0; i < nPoints; i++)
32       matrixSystem[row][n+1] += pow(xValues[i], row) * yValues[i];
33
34   }
35
36   //Array that holds all the coefficients
37   double coeffVec[n+2];
38
39   //Gauss reduction
40   for(int i = 0; i <= n-1; i++)
41     for (int k=i+1; k <= n; k++)
42     {
43       double t=matrixSystem[k][i]/matrixSystem[i][i];
44
45       for (int j=0;j<=n+1;j++)
46         matrixSystem[k][j]=matrixSystem[k][j]-t*matrixSystem[i][j];
47
48     }
49
50   //Back-substitution
```

```
51    for (int i=n;i>=0;i--)
52    {
53      coeffVec[i]=matrixSystem[i][n+1];
54      for (int j=0;j<=n+1;j++)
55        if (j!=i)
56          coeffVec[i]=coeffVec[i]-matrixSystem[i][j]*coeffVec[j];
57
58      coeffVec[i]=coeffVec[i]/matrixSystem[i][i];
59    }
60
61    //Construct the cv vector and return it
62    cv::vector<double> result = cv::vector<double>();
63    for(int i = 0; i < n+1; i++)
64      result.push_back(coeffVec[i]);
65    return result;
66  }
67
68  //Returns the point for the equation determined
69  //by a vector of coefficents, at a certain x location
70  cv::Point pointAtX(cv::vector<double> coeff, double x)
71  {
72    double y = 0;
73    for(int i = 0; i < coeff.size(); i++)
74    y += pow(x, i) * coeff[i];
75    return cv::Point(x, y);
76  }
```

For convenience, this listing is provided as a separate text file.

Here are some of the things that you'll need. As always, look up anything you don't understand in the online documentation before asking a GTA in the lab:

1. **cv::vector<type>** is a vector that can hold multiple values of the same type (cf array)
2. **cv::Canny(source, destination, lowerThreshold, upperThreshold, aperture, L2Gradient)** is a function that applies the Canny transformation on an image
3. **cv::HoughLinesP(sourceMat, destVector, rho, theta, threshold, minLen, maxGap)** detects the line segments in the sourceMat and adds them to the vector as **cv::Vec4i** objects, containing the coordinates for the segment (x1, y1, x2, y2). Rho is the distance resolution of the accumulator (in pixels). Theta is the angle resolution of the accumulator (in radians). The threshold is the accumulator threshold parameter. Only the lines that get enough votes (>threshold) are returned. minLen is the minimum line length, while maxGap is the maximum allowed gap between line segments.
4. **cv::circle(destination, point, radius, colour)** draws a circle on a cv::Mat. The point parameter is a cv::Point containing the coordinates of the circle and the colour is a cv::Scalar containing the three BGR values (0-255).

5. **cv::line(destination, pt1, pt2, color, ...)** draws a line on a cv::Mat.

6. **cv::moveWindow(name, x, y)** moves a window to the specified x and y coords. Not necessary, but it saves you having to arrange the windows every time you run the program.

You should experiment with

1. the parameters of the Canny function

2. the parameters of the Hough function

to check your understanding of the effect of these parameters.

3 Questions

1. Canny has two thresholds that control the edge thresholding process. What is their purpose?

2. What is the purpose of the *aperture* parameter? What is the result of changing it from 3 to 5, 7, 9 or greater?

3. The Hough transform has two parameters that specify the resolution of the accumulator. Their default values are 1 and $\pi/180$. What is the effect of increasing the first and reducing the second?

4. The Hough transform has a pair of parameters that determine the minimum length of a line that can be accepted, and the maximum gap between two segments if they are to be considered part of the same line. What is the effect of changing these values?

5. How close are the computed horizons to where you think the horizon should be? What might cause any discrepancy?

4 Submit

You now have one code file and four sample images for each source image you've processed and a pdf of your answers to the questions. Zip these and submit them to the Lab4 area of Blackboard.

5 Marking Scheme

You're assessed on five criteria judged by the correctness of your code and the result images, and your answers to the questions:

	1 mark	2 marks
Canny	functioning code	correct output
Probabilistic Hough	functioning code	correct output
Filtering short edge segments	functioning code	correct output
Filtering oriented edge segments	functioning code	correct output
Identify and draw horizon correctly	can draw a horizon	can draw correct horizon
Question 1		a correct answer
Question 2		a correct answer
Question 3		a correct answer
Question 4		a correct answer
Question 5		a correct answer

References

[1] OpenCV website, https://opencv.org/about.html

[2] OpenCV documentation, https://docs.opencv.org/2.4/index.html