COMP27112 Visual Computing

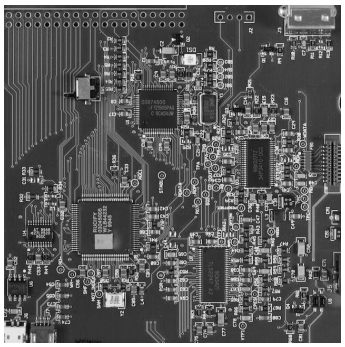
LAB 4 REPORT

April 29, 2024

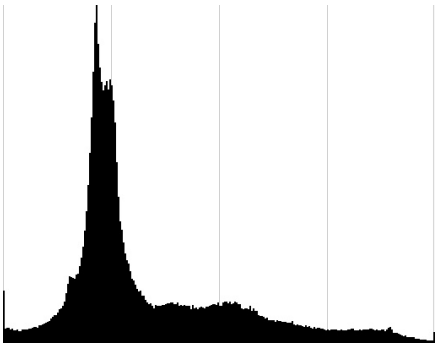
Joseph Rhodes
University of Manchester

Image Histogram and Segmentation

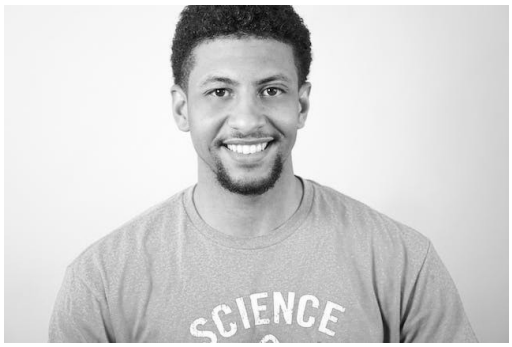
Histograms



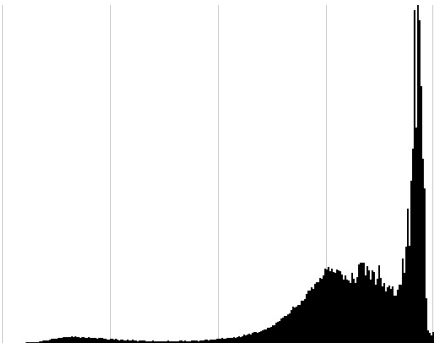
Circuit Board



Circuit Board Histogram



Science Person



Science Person Histogram

Code 1: Source Code for Histogram.cpp

```
#include <stdio.h>
#include <opencv2/core/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace cv;

const int HIST_IMG_HEIGHT = 400;
const int HIST_IMG_WIDTH = 512;
const int BAR_WIDTH = 2;
const int GRID_LINES[] = {0, 64, 128, 192, 255};

void createHistogram(Mat& img, Mat& hist)
{
    long counts[256] = {};


```

```
19    // Create a white image for the histogram
20    hist = Mat(HIST_IMG_HEIGHT, HIST_IMG_WIDTH, CV_8UC1, Scalar
          (255));
21
22    // Draw grid lines
23    for (int i = 0; i < sizeof(GRID_LINES) / sizeof(GRID_LINES
          [0]); ++i) {
24        int grid_line = GRID_LINES[i] * BAR_WIDTH;
25        line(hist, Point(grid_line, 0), Point(grid_line,
              HIST_IMG_HEIGHT - 1), Scalar(200));
26    }
27
28    // Counts of each gray level
29    for (int y = 0; y < img.rows; ++y) {
30        for (int x = 0; x < img.cols; ++x) {
31            uchar pixel = img.at<uchar>(y, x);
32            counts[pixel]++;
33        }
34    }
35
36    // Max count
37    long maxCount = 0;
38    for (int i = 0; i < 256; ++i) {
39        if (counts[i] > maxCount) {
40            maxCount = counts[i];
41        }
42    }
43
44    // Bars of the histogram
45    for (int i = 0; i < 256; ++i) {
46        int barHeight = static_cast<int>((counts[i] *
              HIST_IMG_HEIGHT) / maxCount);
47
48        if (barHeight > 0) {
49            int startX = i * BAR_WIDTH;
50            int endX = (i + 1) * BAR_WIDTH - 1;
51
52            rectangle(hist, Point(startX, HIST_IMG_HEIGHT -
                  barHeight),
53                      Point(endX, HIST_IMG_HEIGHT - 1), Scalar(0)
                          , FILLED);
54        }
55    }
56 }
57
58 int main(int argc, char* argv[])
```

```
59 {
60     Mat img;
61     Mat hist;
62 
63     img = imread(argv[1], IMREAD_GRAYSCALE);
64 
65     // Check if the image was successfully loaded
66     if (img.empty()) {
67         printf("Failed to load image '%s'\n", argv[1]);
68         return -1;
69     }
70 
71     // Create image histogram
72     createHistogram(img, hist);
73 
74     namedWindow("Histogram", WINDOW_NORMAL);
75     imwrite("hist.jpg", hist);
76 
77     imshow("Histogram", hist);
78 
79     // Wait for a key press before quitting
80     waitKey(0);
81 
82     return 0;
83 }
```

Thresholding

All of the Figures are below the explanations

fundus.tif

The threshold value I have chosen for the Fundus image is **95**. I chose this value because it allows for the identification of blood vessels without disruption from darkness as the threshold value increases. Lowering the threshold value would result in the blood vessels not being visible.

glaucoma.jpg

The threshold value I have chosen for the Glaucoma image is **137**. I chose this value because it allows for the identification of the bright region towards the middle, and the dark spot within represents the brighter area inside it. Increasing the threshold value will darken the inner area, making it difficult to identify the brighter area inside. Decreasing the threshold value will combine the two bright areas, making it impossible to identify the brighter area inside.

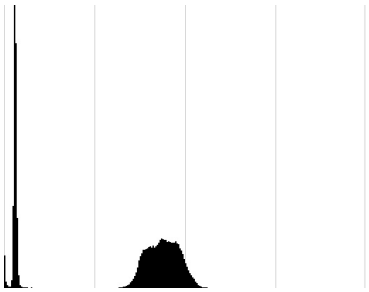
optic_nerve_head.jpg

The threshold value I have chosen for the Optic Nerve Head image is **104**. The threshold value of 104 allows for the identification of the outlines of the large, slightly bright area along with the smaller, brighter area inside it. Increasing the threshold value darkens the image, while decreasing the threshold makes it difficult to identify the large, slightly bright area because it starts to highlight the surrounding area.

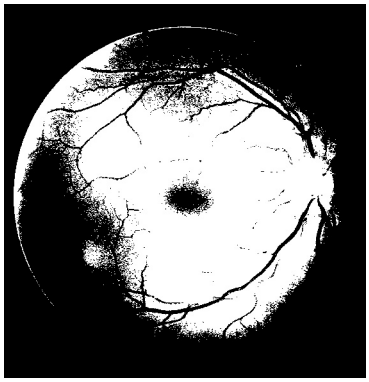
motorway.png

The threshold value I have chosen for the Motorway image is **175**. This one was a little more challenging because, while the text is very distinct at some threshold values, it also highlights the road. At the threshold value of 175, the image still highlights the road somewhat, but you can still read the sign information on the front signs. However, it remains difficult to read the signs at the back without excessively highlighting the road. Increasing the threshold would render the signs unreadable, while decreasing the threshold would highlight the road more than the signs.

(a) Fundus Original Image

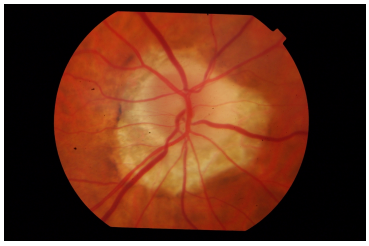


(b) Fundus Histogram

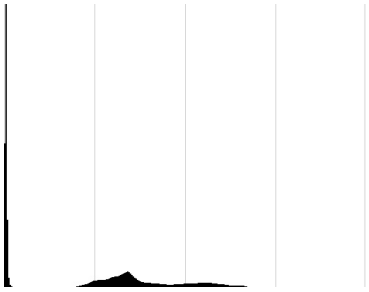


(c) Threshold Fundus Image

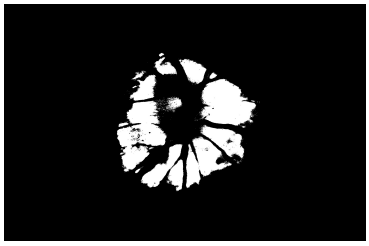
Figure 1: fundus.tif



(a) Glaucoma Original Image



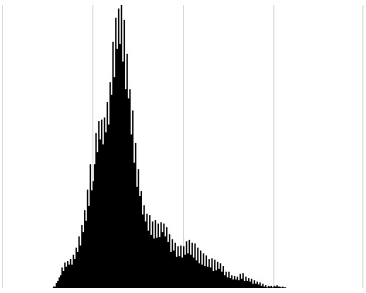
(b) Glaucoma Histogram



(c) Threshold Glaucoma Image

Figure 2: glaucoma.jpg

(a) Optic Nerve Head Original Image

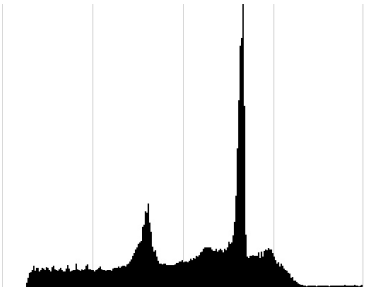


(b) Optic Nerve Head Histogram

(c) Threshold Optic Nerve Head Image

Figure 3: optic_nerve_head.jpg

(a) Motorway Original Image



(b) Motorway Histogram

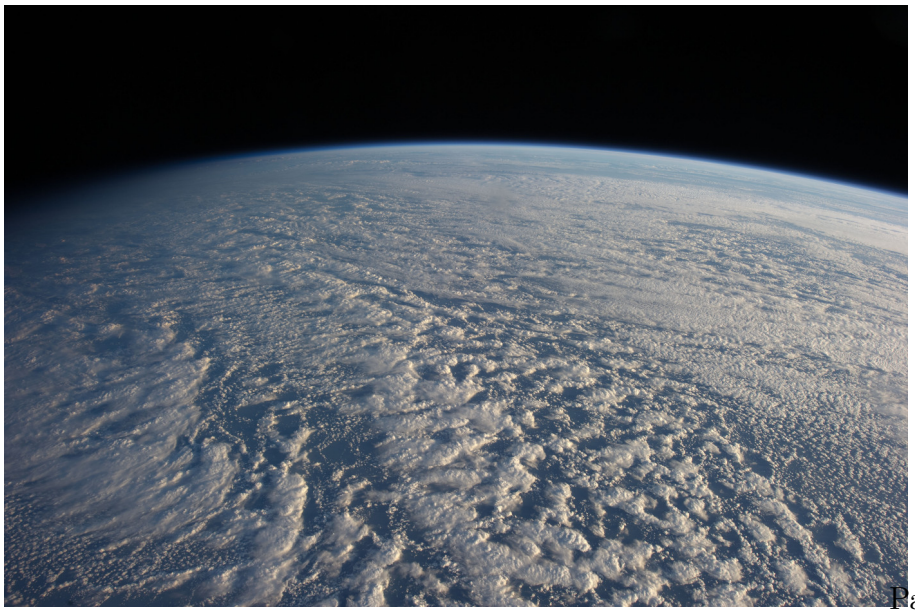
(c) Threshold Motorway Image

Figure 4: motorway.png

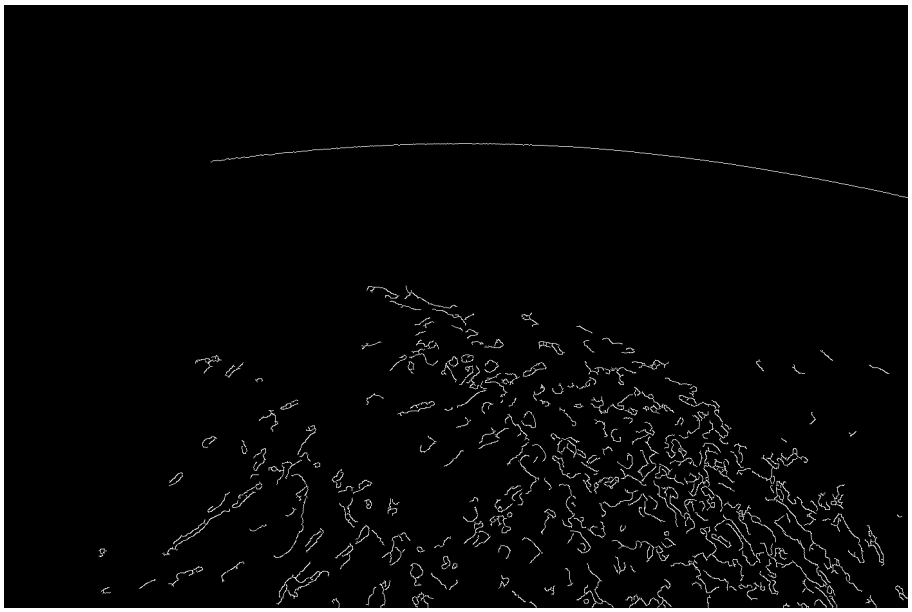
Horizon Detection

Horizon 1

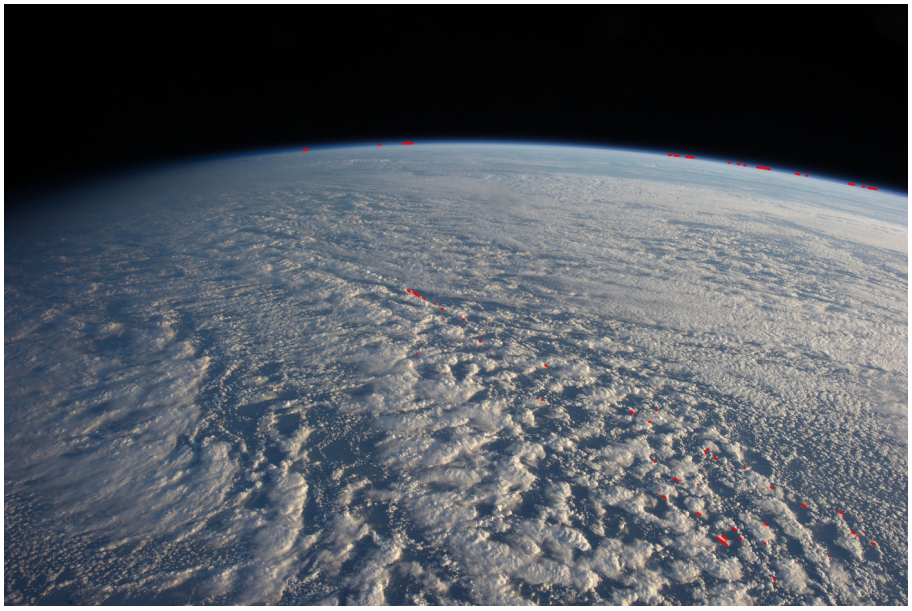
Horizon 1 Original Image



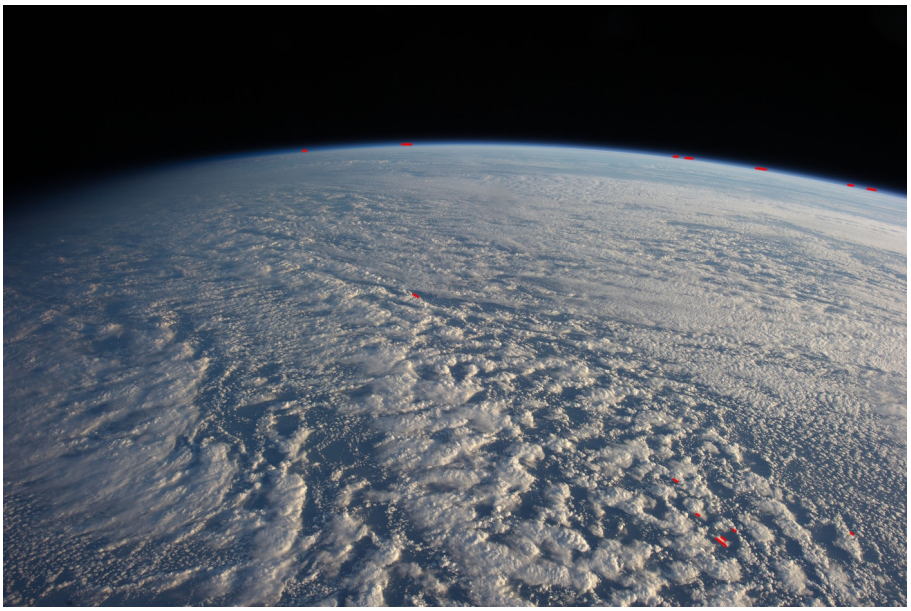
Horizon 1 Canny Edge



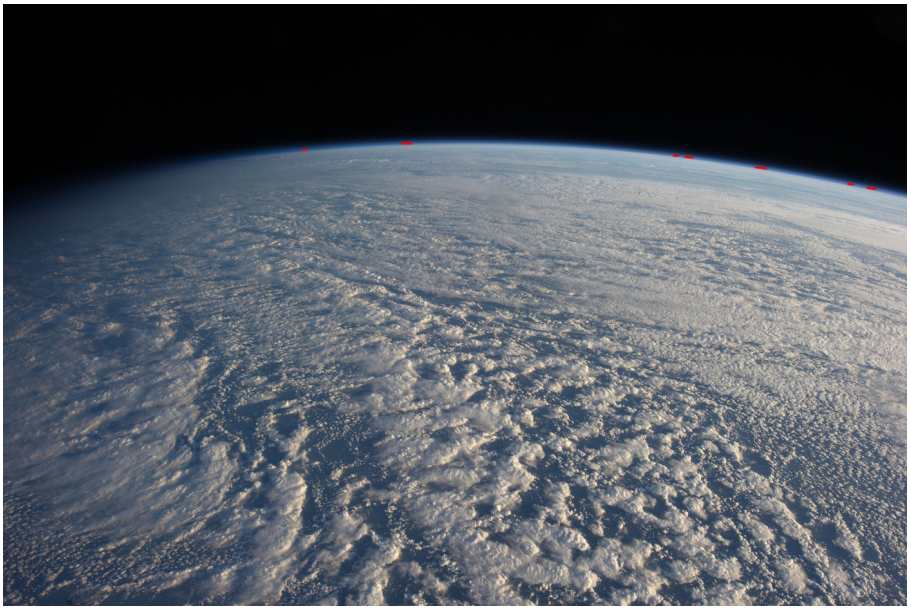
Horizon 1 Hough Line



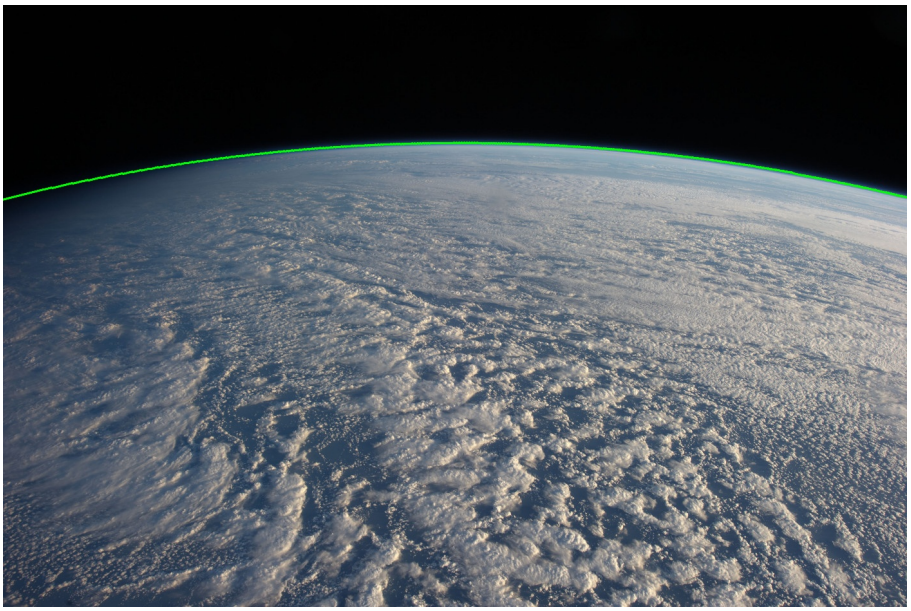
Horizon 1 Short Lines Removed



Horizon 1 Vertical Lines Removed



Horizon 1 Horizon Line



Horizon 1 Parameters

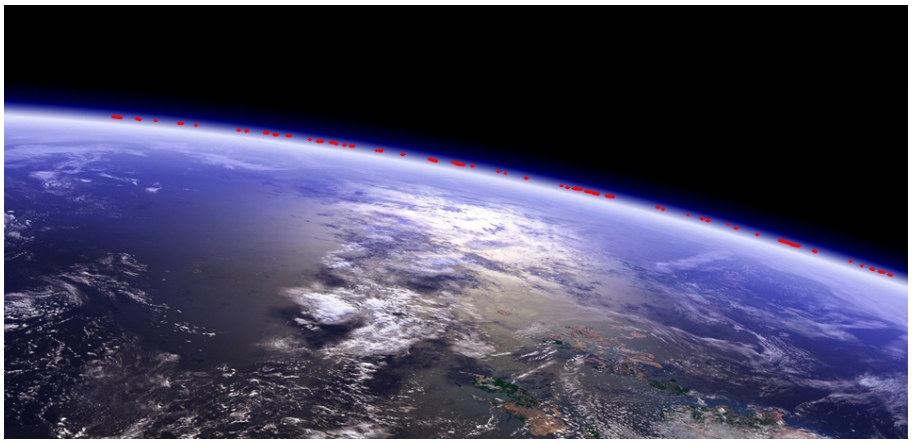
Canny Image		**Other Filters**	
Lower Threshold	30	Min Line Length	5
Upper Threshold	150	Max Angle Threshold	15
Hough Lines Image		**Processing (Gaussian Blur)**	
Rho	1	Blur Size	7
Theta	170	Standard Deviation	4
Threshold	100		

Horizon 2

Horizon 2 Original Image

Horizon 2 Canny Edge

Horizon 2 Hough Line



Horizon 2 Short Lines Removed

Horizon 2 Vertical Lines Removed

Horizon 2 Horizon Line

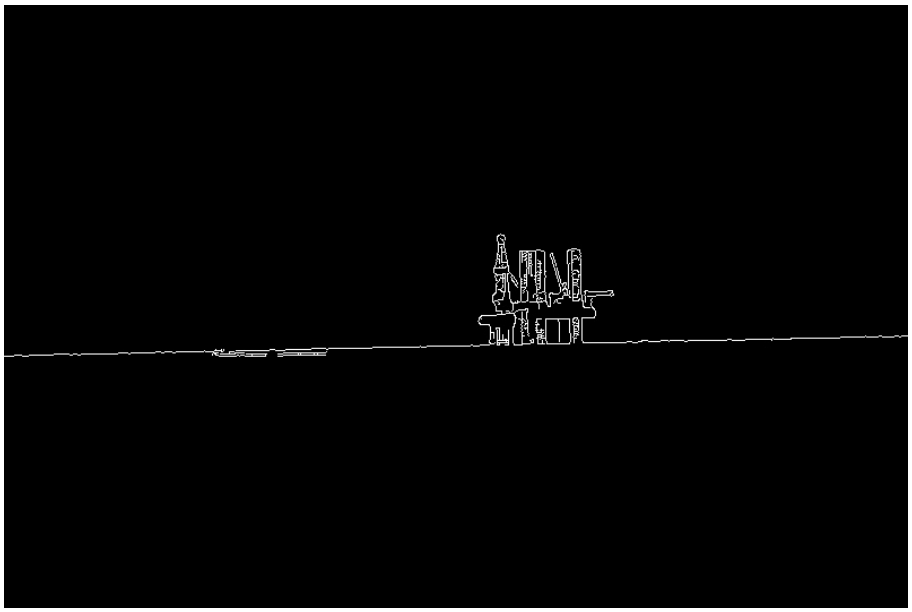
Horizon 2 Parameters

Canny Image		**Other Filters**	
Lower Threshold	135	Min Line Length	5
Upper Threshold	145	Max Angle Threshold	15
Hough Lines Image		**Processing (Gaussian Blur)**	
Rho	1	Blur Size	4
Theta	180	Standard Deviation	4
Threshold	75		

Horizon 3

Horizon 3 Original Image

Horizon 3 Canny Edge



Horizon 3 Hough Line

Horizon 3 Short Lines Removed

Horizon 3 Vertical Lines Removed

Horizon 3 Horizon Line

Horizon 3 Parameters

Canny Image		**Other Filters**	
Lower Threshold	30	Min Line Length	20
Upper Threshold	250	Max Angle Threshold	30
Hough Lines Image		**Processing (Gaussian Blur)**	
Rho	1	Blur Size	1 (No Blur)
Theta	180	Standard Deviation	4
Threshold	30		

Code 2: Source Code for Horizon.cpp

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/imgcodecs.hpp>
#include <cmath>

using namespace cv;

//Polynomial regression function
std::vector<double> fitPoly(std::vector<cv::Point> points, int n)
{
  //Number of points
  int nPoints = points.size();
```

```
15
16    //Vectors for all the points' xs and ys
17    std::vector<float> xValues = std::vector<float>();
18    std::vector<float> yValues = std::vector<float>();
19
20    //Split the points into two vectors for x and y values
21    for(int i = 0; i < nPoints; i++)
22    {
23      xValues.push_back(points[i].x);
24      yValues.push_back(points[i].y);
25    }
26
27    //Augmented matrix
28    double matrixSystem[n+1][n+2];
29    for(int row = 0; row < n+1; row++)
30    {
31      for(int col = 0; col < n+1; col++)
32      {
33        matrixSystem[row][col] = 0;
34        for(int i = 0; i < nPoints; i++)
35          matrixSystem[row][col] += pow(xValues[i], row + col);
36      }
37
38      matrixSystem[row][n+1] = 0;
39      for(int i = 0; i < nPoints; i++)
40        matrixSystem[row][n+1] += pow(xValues[i], row) * yValues[i
             ];
41
42    }
43
44    //Array that holds all the coefficients
45    double coeffVec[n+2]; // the "= {}" is needed in visual studio,
          but not in Linux
46
47    //Gauss reduction
48    for(int i = 0; i <= n-1; i++)
49      for (int k=i+1; k <= n; k++)
50      {
51        double t=matrixSystem[k][i]/matrixSystem[i][i];
52
53        for (int j=0;j<=n+1;j++)
54          matrixSystem[k][j]=matrixSystem[k][j]-t*matrixSystem[i][j
               ];
55
56      }
57
```

```
58    //Back-substitution
59    for (int i=n;i>=0;i--)
60    {
61      coeffVec[i]=matrixSystem[i][n+1];
62      for (int j=0;j<=n+1;j++)
63        if (j!=i)
64          coeffVec[i]=coeffVec[i]-matrixSystem[i][j]*coeffVec[j];
65
66      coeffVec[i]=coeffVec[i]/matrixSystem[i][i];
67    }
68
69    //Construct the vector and return it
70    std::vector<double> result = std::vector<double>();
71    for(int i = 0; i < n+1; i++)
72      result.push_back(coeffVec[i]);
73    return result;
74  }
75
76  //Returns the point for the equation determined
77  //by a vector of coefficents, at a certain x location
78  cv::Point pointAtX(std::vector<double> coeff, double x)
79  {
80    double y = 0;
81    for(int i = 0; i < coeff.size(); i++)
82      y += pow(x, i) * coeff[i];
83    return cv::Point(x, y);
84  }
85
86
87  //
       ----------------------------------------------------------------------
88
89
90
91  int main(int argc, char* argv[])
92  {
93      Mat img;
94      img = imread(argv[1], IMREAD_COLOR);
95
96      // Check for failure
97      if (img.empty()) {
98          printf("Failed to load image '%s'\n", argv[1]);
99          return -1;
100     }
101
```

```
102     Mat grayImg;
103     if (img.channels() > 1) {
104         cvtColor(img, grayImg, COLOR_BGR2GRAY);
105     } else {
106         grayImg = img.clone();
107     }
108 
109     // Blur size
110     int blurSize = 1;
111 
112 
113     if (blurSize % 2 == 0) {
114         blurSize++;
115     }
116 
117     // Apply Gaussian blur to the grayscale image
118     Mat blurredImg;
119     GaussianBlur(grayImg, blurredImg, Size(blurSize, blurSize),
            4);
120 
121     // Applys Canny edge detection
122     Mat edges;
123     double lowerThreshold = 30;
124     double upperThreshold = 250;
125     Canny(blurredImg, edges, lowerThreshold, upperThreshold);
126 
127 
128     imshow("Canny Edges", edges);
129     imwrite("edges.jpg", edges);
130 
131 
132     // Apply probabilistic Hough transformation
133     std::vector<Vec4i> lines;
134     double rho = 1;
135     double theta = CV_PI / 180;
136     int threshold = 30;
137     HoughLinesP(edges, lines, rho, theta, threshold);
138 
139     // Draw the detected lines on the original image
140     Mat houghLinesImg = img.clone();
141     for (size_t i = 0; i < lines.size(); i++) {
142         Vec4i l = lines[i];
143         line(houghLinesImg, Point(l[0], l[1]), Point(l[2], l[3]),
                Scalar(0, 0, 255), 2, LINE_AA);
144     }
145 
```

```
146     imshow("Hough Lines", houghLinesImg);
147     imwrite("houghLinesImg.jpg", houghLinesImg);
148 
149 
150     // Remove short lines
151     std::vector<Vec4i> filteredShortLines;
152     double minLengthThreshold = 20;
153     for (size_t i = 0; i < lines.size(); i++) {
154         Vec4i l = lines[i];
155         double length = sqrt(pow(l[2] - l[0], 2) + pow(l[3] - l
                [1], 2));
156         if (length >= minLengthThreshold) {
157             filteredShortLines.push_back(l);
158     }
159 
160     // Draw the original image with short lines removed
161     Mat imgWithoutShortLines = img.clone();
162     for (size_t i = 0; i <filteredShortLines.size(); i++) {
163         Vec4i l = filteredShortLines[i];
164         line(imgWithoutShortLines, Point(l[0], l[1]), Point(l[2],
                l[3]), Scalar(0, 0, 255), 2, LINE_AA);
165     }
166 
167     imshow("Original Image with Short Lines Removed",
            imgWithoutShortLines);
168     imwrite("imgWithoutShortLines.jpg", imgWithoutShortLines);
169 
170 
171 
172 
173     // Remove vertical lines
174     std::vector<Vec4i> filteredVerticalLines;
175     double maxAngleThreshold = 30.0;
176     for (size_t i = 0; i < filteredShortLines.size(); i++) {
177         Vec4i l = filteredShortLines[i];
178         double angle = atan2(l[3] - l[1], l[2] - l[0]) * 180.0 /
                CV_PI;
179         if (std::abs(angle) < maxAngleThreshold || std::abs(angle
                ) > 180  - maxAngleThreshold) {
180             filteredVerticalLines.push_back(l);
181         }
182     }
183 
184     // Draw the original image with vertical lines removed
185     Mat imgWithoutVerticalLines = img.clone();
186     for (size_t i = 0; i < filteredVerticalLines.size(); i++) {
```

```
187         Vec4i l = filteredVerticalLines[i];
188         line(imgWithoutVerticalLines, Point(l[0], l[1]), Point(l
                [2], l[3]), Scalar(0, 0, 255), 2, LINE_AA);
189     }
190
191     imshow("Original Image with Vertical Lines Removed",
            imgWithoutVerticalLines);
192     imwrite("imgWithoutVerticalLines.jpg",
            imgWithoutVerticalLines);
193
194
195     // Find nearly horizontal lines' points
196     std::vector<Point> horizontalLinePoints;
197     double angleThreshold = 30.0;
198     for (size_t i = 0; i < filteredShortLines.size(); i++) {
199         Vec4i l = filteredShortLines[i];
200         double dangle = atan2(l[3] - l[1], l[2] - l[0]) * 180.0 /
                CV_PI;
201         if (std::abs(dangle) < angleThreshold || std::abs(dangle)
                > 180 - angleThreshold) {
202             horizontalLinePoints.push_back(Point(l[0], l[1]));
203             horizontalLinePoints.push_back(Point(l[2], l[3]));
204         }
205     }
206
207     std::vector<double> curveCoefficients = fitPoly(
            horizontalLinePoints, 2);
208
209     // Draw the curve on the original image (Horizon Line)
210     for (int x = 0; x < img.cols; x++) {
211         int y = 0;
212         for (int i = 0; i < curveCoefficients.size(); i++) {
213             y += curveCoefficients[i] * pow(x, i);
214         }
215         if (y >= 0 && y < img.rows) {
216             line(img, Point(x, y), Point(x, y), Scalar(0, 255, 0)
                    , 2, LINE_AA);
217         }
218     }
219
220     imshow("Original Image with Curve", img);
221     imwrite("horizon.jpg", img);
222
223
224     waitKey(0);
225
```

```
226     return 0;
227 }
```