Visual Computing Lab 4

Ayush Gupta

April 26, 2024

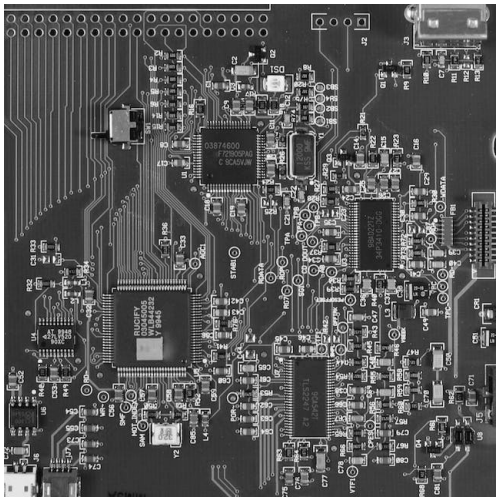
1 Histogram Images and Code

1.1 Code For Histogram Generation

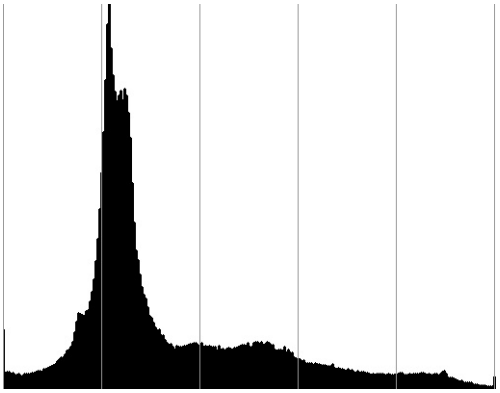
```
void createHistogram(Mat& img, Mat& hist)
{
        long counts[256] = {}; // Array for counts, zeroed
        long max = 0;
        hist = Mat(400, 512, CV_8UC1, 255); // empty white image for histogram

        for (int i = 0; i < img.rows; i++){
                for (int j = 0; j < img.cols; j++){
                        counts[img.at<uchar>(i,j)]++;
                }
        }

        for (int i = 0; i < 256; i++){
                if (max < counts[i]){
                        max = counts[i];

                }
                // std::cout << counts[i] << " ";
        }
        // std::cout << "max = "<< max;

        for (int i = 0; i < 256; i++){
                int len = static_cast<int>((HIST_IMG_HEIGHT*counts[i])/max);
                line(hist, Point(2*i, HIST_IMG_HEIGHT - len), Point(2*i, HIST_IMG_HEIGHT),
                Scalar(0), 2);
        }

        for (int i = 0; i < 256; i += 256 / 5){
                line(hist, Point(2*i, 0), Point(2*i, HIST_IMG_HEIGHT), Scalar(150), 1);
        }
}
```

1.2 Images and Their Histograms

1.2.1 Circuit Board Image



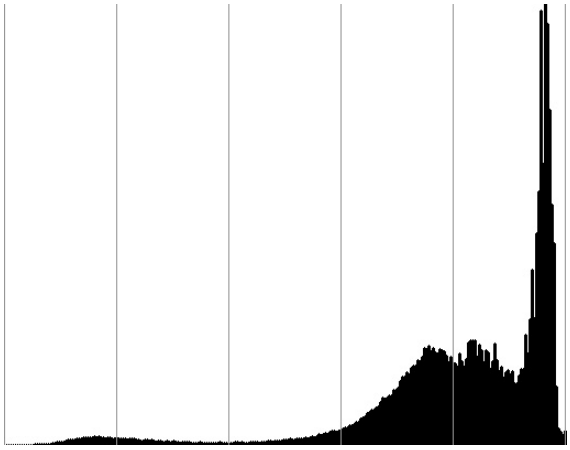
(a) Original Image



(b) Histogram

1.2.2 Science Person Image

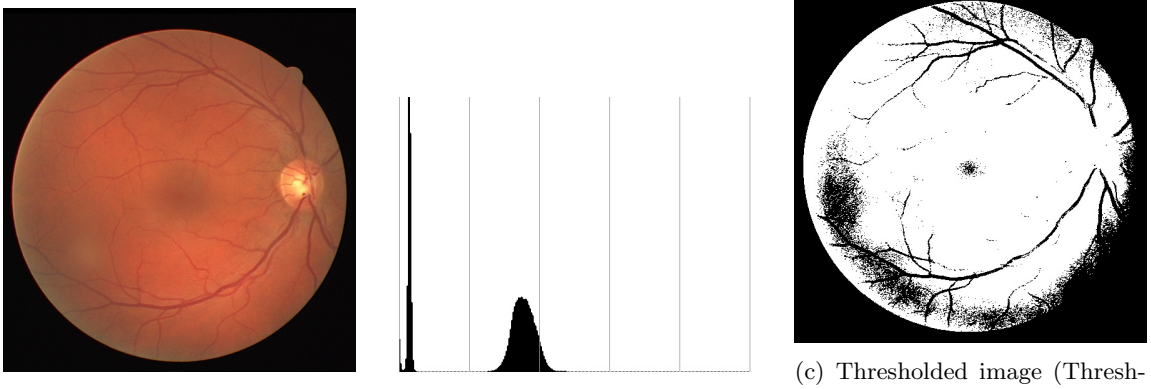
(a) Original Image



(b) Histogram

2 Image Thresholding

2.1 Thresholding Analysis of Fundus Image

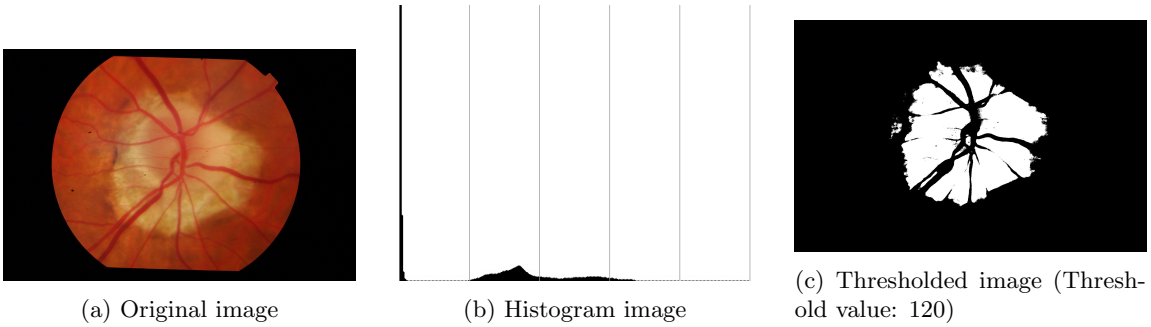


(a) Original image (b) Histogram (c) Thresholded image (Threshold value: 82)

Figure 3: Thresholding of fundus image

The threshold value was selected with the primary objective of achieving optimal visibility of the veins within the fundus image. Veins began to emerge distinctly after surpassing the threshold of 75, with further enhancement in clarity observed around the threshold value of 82. However, it's noteworthy that the intensity of grey levels within the veins closely resembled that of their surrounding regions. This is evidenced by the fact that upon exceeding the threshold of 82, the surrounding area adjacent to the lower veins progressively darkened, as depicted in the chosen thresholded image.

2.2 Thresholding Analysis of Glaucoma Image

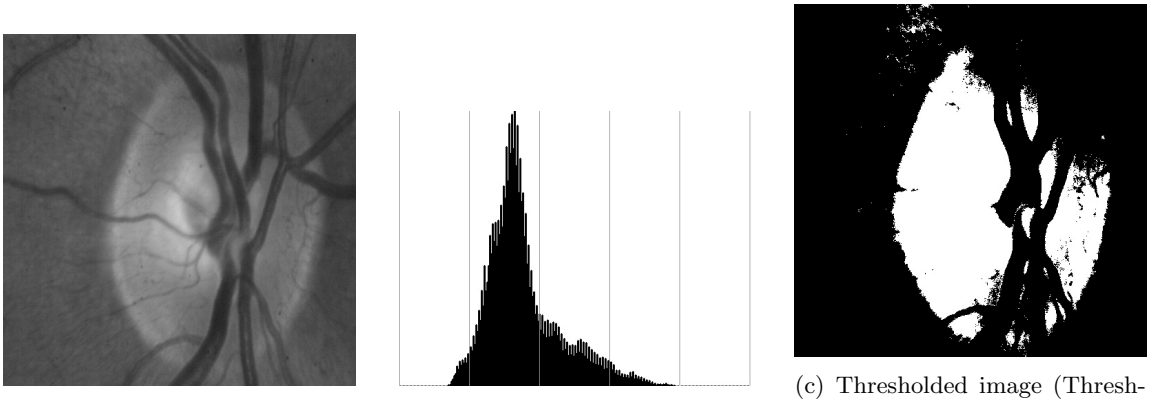


(a) Original image (b) Histogram image (c) Thresholded image (Threshold value: 120)

Figure 4: Thresholding of glaucoma image

The chosen threshold value effectively isolates the central brighter area in the glaucoma picture from the background. Determining the appropriate threshold value for this image was relatively straightforward, as the central brighter area is sufficiently distinct from the surrounding regions. Within the range of threshold values 105-120, the thresholded image remains relatively consistent. Opting for a threshold value of 120 helps sharpen the delineation of boundaries within the central bright area.

2.3 Thresholding Analysis of Optic Nerve Head Image

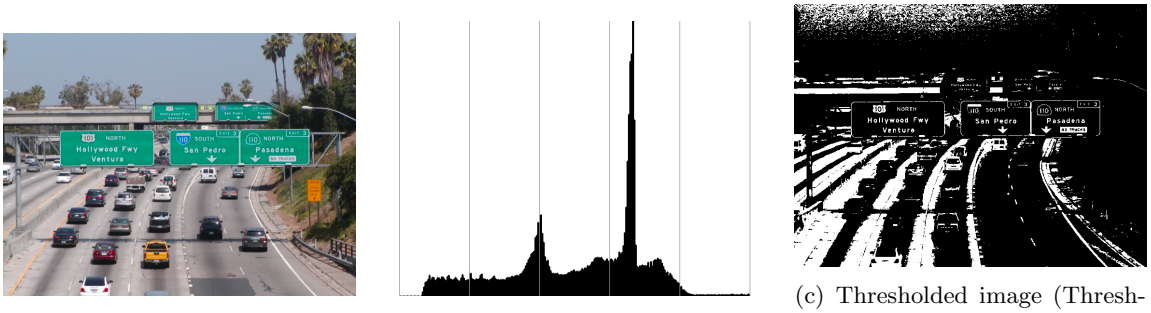


(a) Original image (b) Histogram image (c) Thresholded image (Threshold value: 102)

Figure 5: Thresholding of optic nerve image

The objective of the processing is to delineate the outlines of both the large, slightly bright area and the smaller, brighter area within it. The challenge encountered in determining the suitable threshold value arose from balancing the need for sharp boundaries around the bright area while avoiding the loss of detail in the top right corner region. As the threshold value was increased to refine the boundaries, the densely packed dark nerves in the top right corner began to darken. To prioritize obtaining well-defined boundaries, I opted for a threshold value of 102.

2.4 Thresholding Analysis of Motorway Image



(a) Original image (b) Histogram image (c) Thresholded image (Threshold value: 170)

Figure 6: Thresholding of motorway image

With the goal of isolating the white text for the car sensor to read speed limit signs, I approached the determination of the threshold value differently. As the threshold value increased, the text became increasingly visible, starting from around 140. However, this also retained other elements such as cars, roads, and other signboards in the image. Recognizing that this additional information could potentially act as noise for the sensor and distract it from focusing on the main target, I continued to increase the threshold value to eliminate this extra noise. This is evident in the resulting thresholded image, where the top and right regions of the image are nearly blacked out, leaving only the white text visible and legible, even for humans.

3 Horizon Detection

3.1 Code For Horizon Detection

3.1.1 Greying and Blurring Implementation

```
  img = cv::imread(argv[1]);
  // Point2f center(img.cols/2.0, img.rows/2.0);
  // Mat rotation_matrix = getRotationMatrix2D(center, -1.1472, 1.0);
  // warpAffine(img, img, rotation_matrix, img.size());
  dupimg = img.clone();
  dupimg1 = img.clone();
  dupimg2 = img.clone();

        // Check if the image was successfully loaded
        if (img.empty()) {
                        printf("Failed to load image '%s'\n", argv[1]);
                        return -1;
        }
  imshow("Original Image", img);
  cv::blur(img, blurimg, cv::Size(10, 10));
  imshow("blurimg",blurimg);
  cv::imwrite("blurimg.jpg", blurimg);
  cv::cvtColor(blurimg, grey_img, cv::COLOR_BGR2GRAY);
  imshow("greyimg",grey_img);
```

3.1.2 Canny Implementation

```
  namedWindow("Result", cv::WINDOW_NORMAL);
  resizeWindow("Result", 800, 500);
  createTrackbar("Threshold1", "Result", NULL, 255, canny1);
  createTrackbar("Threshold2", "Result", NULL, 255, canny2);


  void canny1(int val, void*)
{
        Canny(grey_img, out, val, th2);
  th1 = val;
        imshow("Result", out);
}

void canny2(int val, void*)
{
        Canny(grey_img, out, th1, val);
  th2 = val;
        imshow("Result", out);
}
```

3.1.3 HoughLine (Short and Vertical Line Filter) Implementation

```
 HoughLinesP(out, lines, 1, CV_PI / 180, 50);
  std::vector<Point> points;
  int angle;
  for(int i = 0; i < lines.size(); i++){
    Vec4i lin = lines[i];
    angle = abs((( atan2( (lin[1] - lin[3]) , (lin[0] - lin[2]) ) ) * 180 / CV_PI ));
```

```
    line(dupimg, Point(lin[0], lin[1]), Point(lin[2], lin[3]), Scalar(0,0,255), 2, LINE_AA);

    if(sqrt( pow(lin[0] - lin[2], 2) + pow(lin[1] - lin[3],2) ) > 5){
        line(dupimg1, Point(lin[0], lin[1]), Point(lin[2], lin[3]), Scalar(0,0,255),
        2, LINE_AA);

      if(angle < 40 || angle > 140){
        line(dupimg2, Point(lin[0], lin[1]), Point(lin[2], lin[3]), Scalar(0,0,255),
        2, LINE_AA);
        points.push_back(Point(lin[0], lin[1]));
        points.push_back(Point(lin[2], lin[3]));

      }

    }
  }
```

3.1.4 Drawing Horizon Implementation

```
  std::vector<double> coeffs = fitPoly(points, 1);
  for(int x = 0; x < img.cols; x++){
    Point point = pointAtX(coeffs, x);
    if(point.y >= 0 && point.y < img.rows ){
      circle(img, point, 2, Scalar(0,0,255), FILLED);
      // img.at<Vec3b>(point.y,point.x) = Vec3b(0,0,255);
    }
  }
```

3.1.5 Full Code

```
  #include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <cmath>
using namespace cv;
using namespace std;
Mat img;
Mat dupimg;
Mat dupimg1;
Mat dupimg2;
Mat blurimg;
Mat grey_img;
Mat out;
int th1 = 43;
int th2 = 31;
//Polynomial regression function
std::vector<double> fitPoly(std::vector<cv::Point> points, int n)
{
  //Number of points
  int nPoints = points.size();

  //Vectors for all the points' xs and ys
  std::vector<float> xValues = std::vector<float>();
```

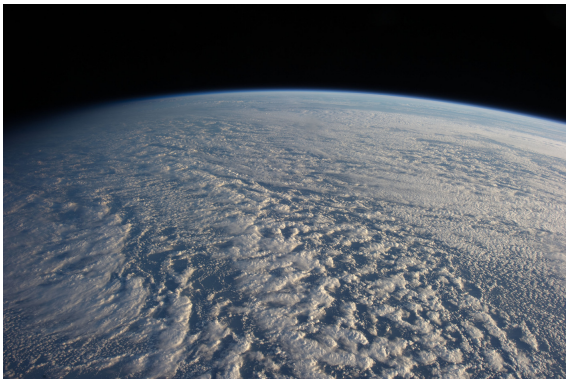
```
std::vector<float> yValues = std::vector<float>();

//Split the points into two vectors for x and y values
for(int i = 0; i < nPoints; i++)
{
  xValues.push_back(points[i].x);
  yValues.push_back(points[i].y);
}

//Augmented matrix
double matrixSystem[n+1][n+2];
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;
```

```
}

//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);
}


void canny1(int val, void*)
{
        Canny(grey_img, out, val, th2);
  th1 = val;
        cv::imshow("Result", out);
}

void canny2(int val, void*)
{
        Canny(grey_img, out, th1, val);
  th2 = val;
        cv::imshow("Result", out);
}



int main(int argc, char *argv[])
{
        img = cv::imread(argv[1]);
  // Point2f center(img.cols/2.0, img.rows/2.0);
  // Mat rotation_matrix = getRotationMatrix2D(center, -1.1472, 1.0);
  // warpAffine(img, img, rotation_matrix, img.size());
  dupimg = img.clone();
  dupimg1 = img.clone();
  dupimg2 = img.clone();

        // Check if the image was successfully loaded
        if (img.empty()) {
                        printf("Failed to load image '%s'\n", argv[1]);
                        return -1;
        }
  imshow("Original Image", img);
  cv::blur(img, blurimg, cv::Size(10, 10));
  imshow("blurimg",blurimg);
  cv::imwrite("blurimg.jpg", blurimg);
  cv::cvtColor(blurimg, grey_img, cv::COLOR_BGR2GRAY);
  imshow("greyimg",grey_img);
        // cv::namedWindow("Result", cv::WINDOW_NORMAL);
        // cv::resizeWindow("Result", 800, 500);
        // cv::createTrackbar("Threshold1", "Result", NULL, 255, canny1);
        // cv::createTrackbar("Threshold2", "Result", NULL, 255, canny2);
  Canny(grey_img, out, th1, th2);
  // imwrite("grey_img.jpg", grey_img);
```

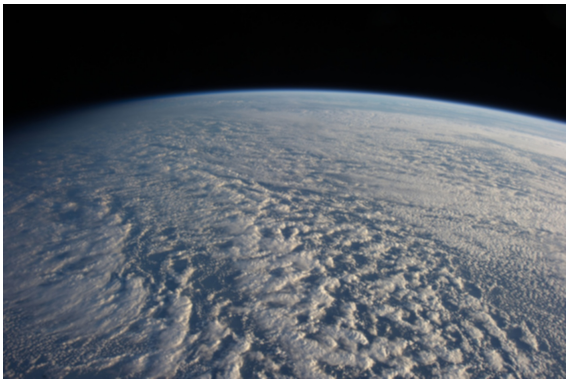
```
  imshow("Canny Edge Image", out);
  imwrite("CannyEdge.jpg", out);
  std::vector<Vec4i> lines;
  HoughLinesP(out, lines, 1, CV_PI / 180, 50);
  std::vector<Point> points;
  int angle;
  for(int i = 0; i < lines.size(); i++){
    Vec4i lin = lines[i];
    angle = abs((( atan2( (lin[1] - lin[3]) , (lin[0] - lin[2]) ) ) * 180 / CV_PI ));
    // cout << angle << " ";
    line(dupimg, Point(lin[0], lin[1]), Point(lin[2], lin[3]), Scalar(0,0,255), 2, LINE_AA);

    if(sqrt( pow(lin[0] - lin[2], 2) + pow(lin[1] - lin[3],2) ) > 5){
        line(dupimg1, Point(lin[0], lin[1]), Point(lin[2], lin[3]), Scalar(0,0,255), 2, LINE_AA

      if(angle < 40 || angle > 140){
        line(dupimg2, Point(lin[0], lin[1]), Point(lin[2], lin[3]), Scalar(0,0,255), 2, LINE_AA
        points.push_back(Point(lin[0], lin[1]));
        points.push_back(Point(lin[2], lin[3]));

      }

    }
  }
  imshow("Hough Line Image", dupimg);
  imshow("Hough Line Image - shortLines", dupimg1);
  imshow("Hough Line-(FilteredVertical) Image", dupimg2);
  imwrite("HoughLineImage.jpg",dupimg);
  imwrite("HoughLineImage_shortLines.jpg",dupimg1);
  imwrite("HoughLine_(FilteredVeritical)Image.jpg", dupimg2);
  std::vector<double> coeffs = fitPoly(points, 1);
  for(int x = 0; x < img.cols; x++){
    Point point = pointAtX(coeffs, x);
    if(point.y >= 0 && point.y < img.rows ){
      circle(img, point, 2, Scalar(0,0,255), FILLED);
      // img.at<Vec3b>(point.y,point.x) = Vec3b(0,0,255);
    }
  }

        // Call the callback function for the initial processing

  // imshow("Greyscale Image", out);
  // imshow("blurimg", grey_img);
  imshow("Final Image", img);
  imwrite("FinalImage.jpg",img);
        // Wait for a key press before quitting
        waitKey(0);

        return 0;
}
```

3.2 Images with Horizon Drawn

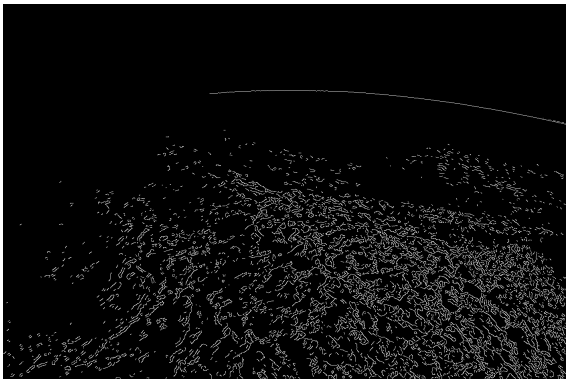
3.2.1 Horizon Detection on horizon1.jpg



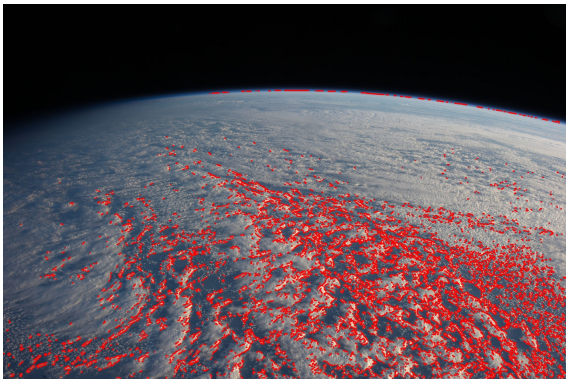
(a) Original Image



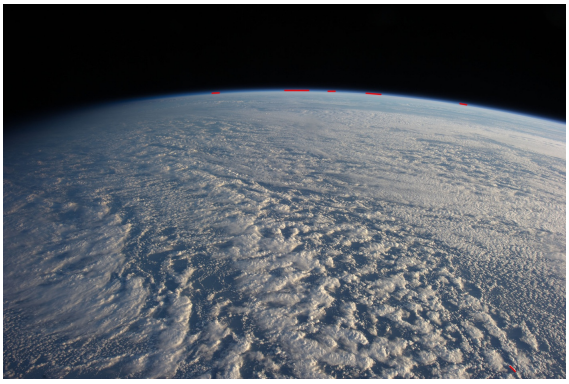
(b) Blurred Image



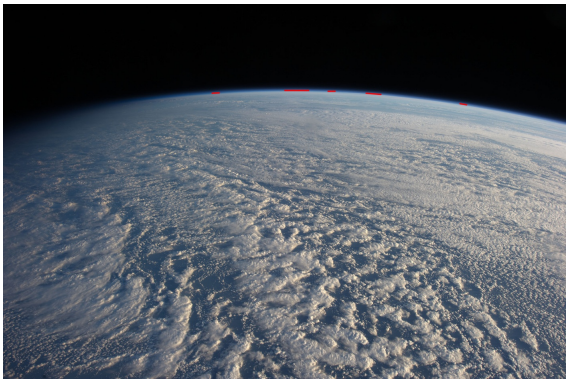
(c) Canny Edge Image



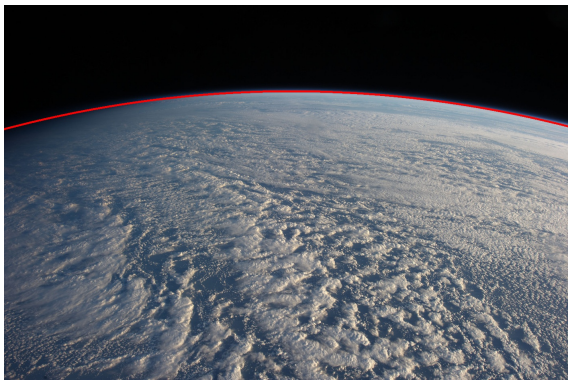
(d) Probabilistic Hough Lines



(e) Image with Short Lines Removed



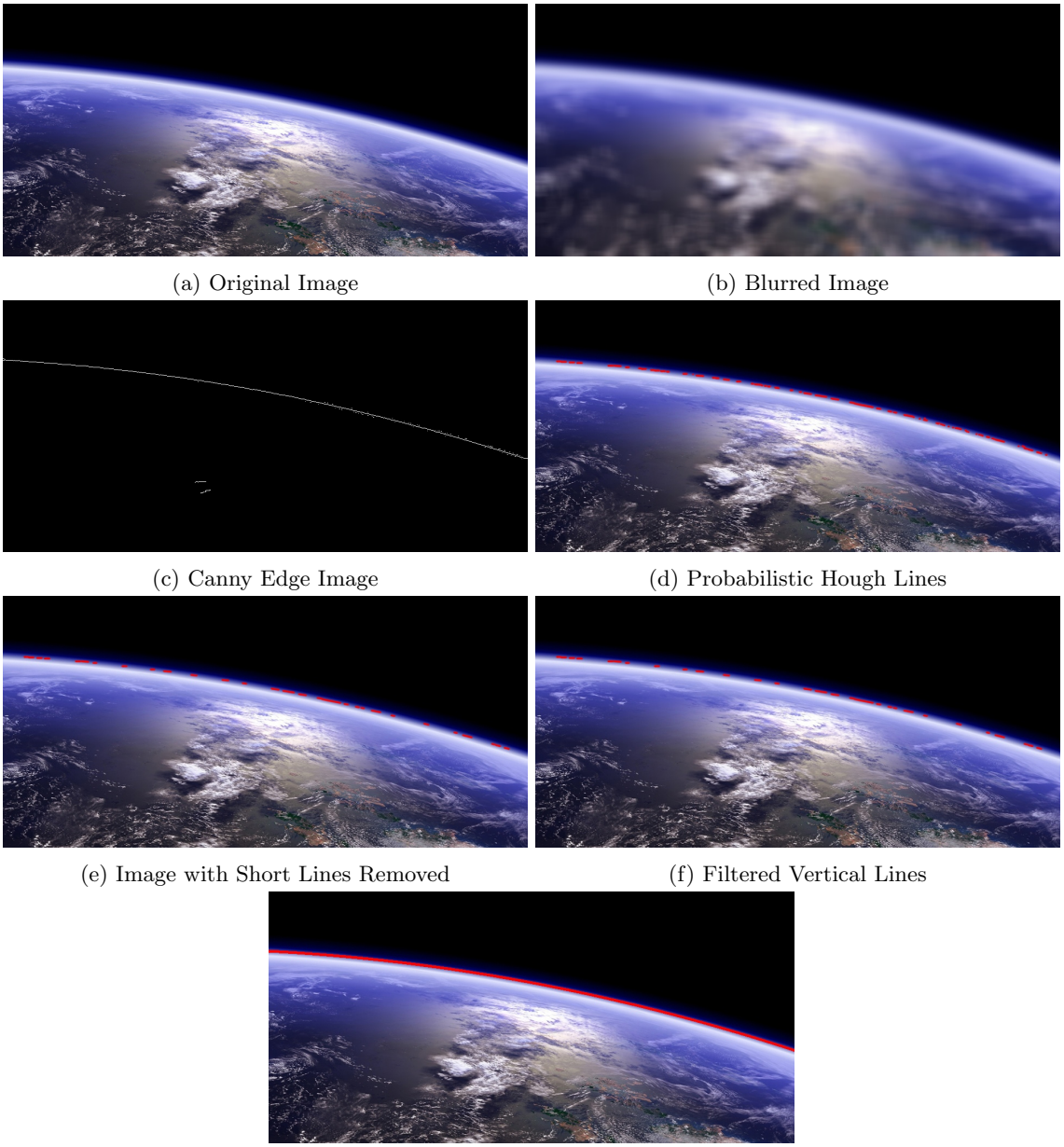
(f) Filtered Vertical Lines



(g) Horizon Drawn

Figure 7: Various processing steps applied to horizon 1 image.

3.2.2 Horizon Detection on horizon2.jpg



(a) Original Image

(b) Blurred Image

(c) Canny Edge Image

(d) Probabilistic Hough Lines

(e) Image with Short Lines Removed

(f) Filtered Vertical Lines

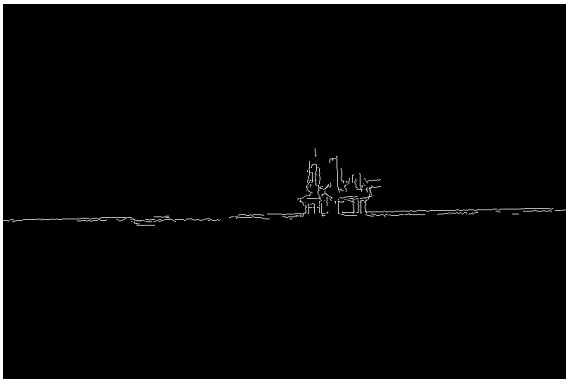
(g) Horizon Drawn

Figure 8: Various processing steps applied to the horizon 2 image.

3.2.3 Horizon Detection on horizon3.jpg

(a) Original Image

(b) Blurred Image



(c) Canny Edge Image

(d) Probabilistic Hough Lines

(e) Image with Short Lines Removed

(f) Filtered Vertical Lines

(g) Horizon Drawn

Figure 9: Various processing steps applied to the horizon 3 image.

3.3 Parameters Chosen For Above Images

Parameters for horizon1.jpg
Blurring - *Blur(5,5)*
Canny - *LowThreshohld = 43, HighThreshold = 31*
HoughLines - *rho = 1, theta = CV_PI, threshold = 50*
Shortest Line Size - *15*
Horizontal Angle Range - *0 - 40, 140 - 180 (in degrees)*
Degree of Polynomial - *2*

Parameters for horizon2.jpg
Blurring - *Blur(20,20)*
Canny - *LowThreshohld = 97, HighThreshold = 3*
HoughLines - *rho = 1, theta = CV_PI, threshold = 50*
Shortest Line Size - *5*
Horizontal Angle Range - *0 - 40, 140 - 180 (in degrees)*
Degree of Polynomial - *2*

Parameters for horizon3.jpg
Blurring - *Blur(10,10)*
Canny - *LowThreshohld = 43, HighThreshold = 31*
HoughLines - *rho = 1, theta = CV_PI, threshold = 50*
Shortest Line Size - *5*
Horizontal Angle Range - *0 - 40, 140 - 180 (in degrees)*
Degree of Polynomial - *1*

NOTE: *Blurring of Images was done as Extra Processing to make it convenient for Canny Edge Detection and finding the right parameters.*