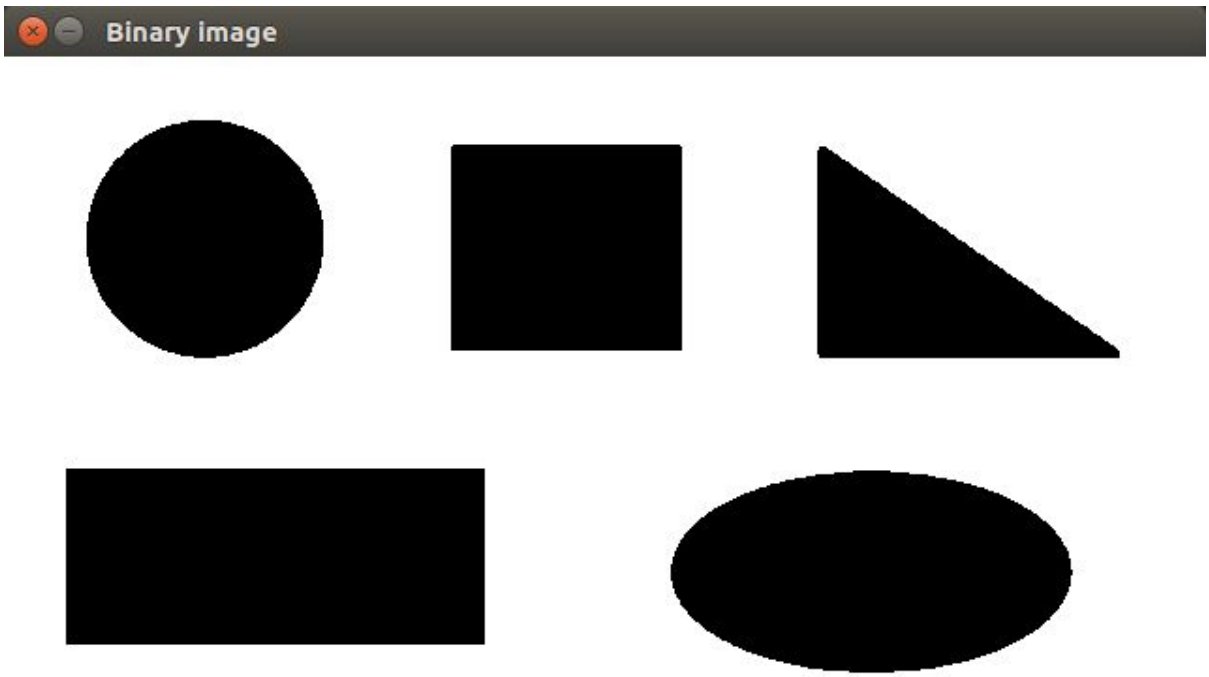
Nícolas Maduro

The codes are appended at the end.

1. In Question 1, the binary image was created with the *threshold* function. Option 1 just counts the black pixels and/or make the reverse binary image. Option 2 uses the *Canny* function to find the step edges, then uses the *findContours* function to find the contours. When a black object is clicked area and perimeter are calculated with OpenCV functions and the diameter is calculated Euclidean distance.

Results:



Obtained values:

Circle:
Perimeter: 420.475
Area: 12611
Diameter: 127.914

Square:
Perimeter: 463.657
Area: 13524
Diameter: 163.686

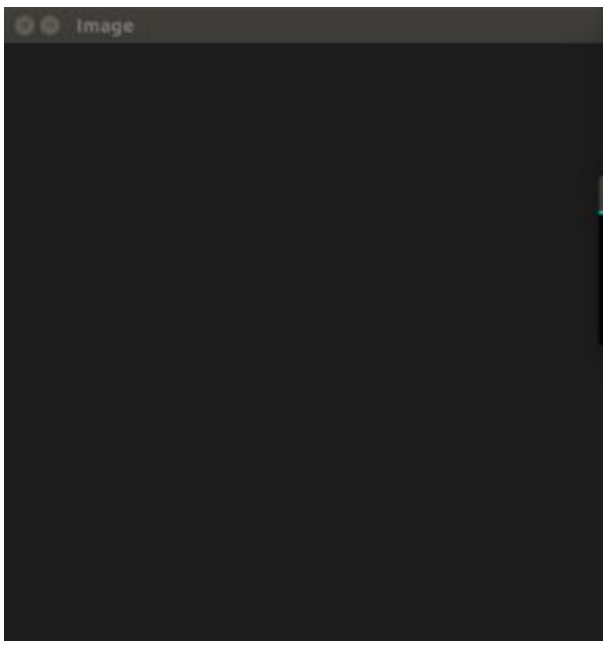
Triangle:
Perimeter: 481.22
Area: 9662
Diameter: 194.733

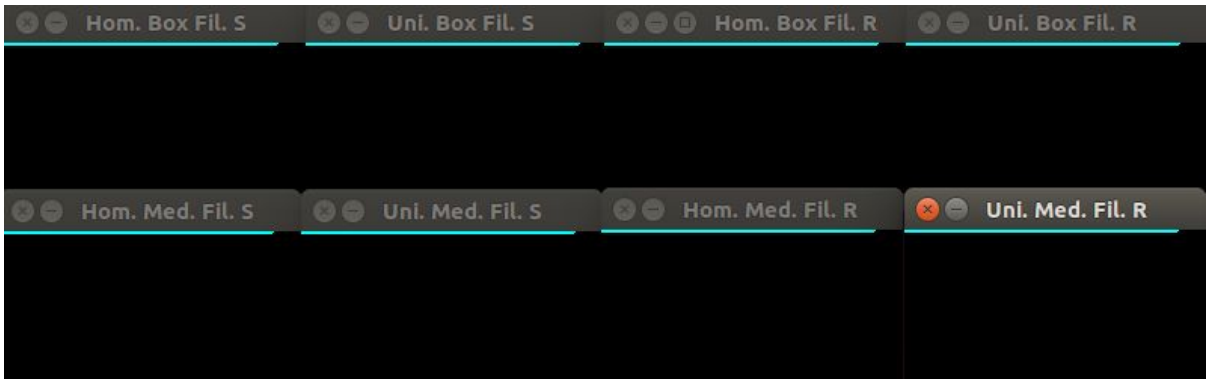
Rectangle:
Perimeter: 631.657
Area: 20959
Diameter: 241.232

Ellipse:
Perimeter: 548.517
Area: 18103.5
Diameter: 214.058

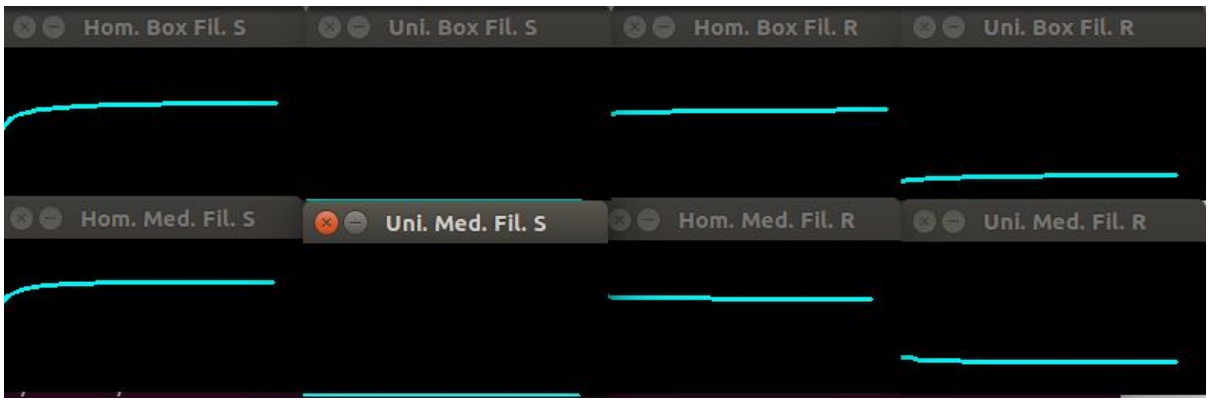
2. In question 2, the *medianBlur* was used to apply the local median operator and the *Blur* to apply the box filter. The smoothed S(n) and residual images R (n) for n = 0, ..., 30 with respect to the two noise-removal operations was produced according equation 2.33 in page 72 of the textbook. The co-occurrence matrix was created by checking the 4 adjacencies. The uniformity and homogeneity were calculated according to the formulas passed in class.

Results:





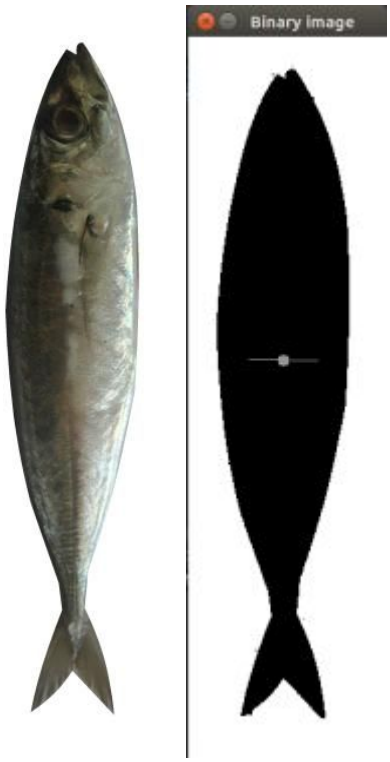
For a uniform image, all curves of the graphics are near 1. Indicating correctly the homogeneity and uniformity of the figure.



For a non-uniform image, results of two different noise-removal techniques are similar. They show that the image becomes more homogeneous when are produced new smoothed images to a certain point, then it stabilizes. However uniformity does not change.

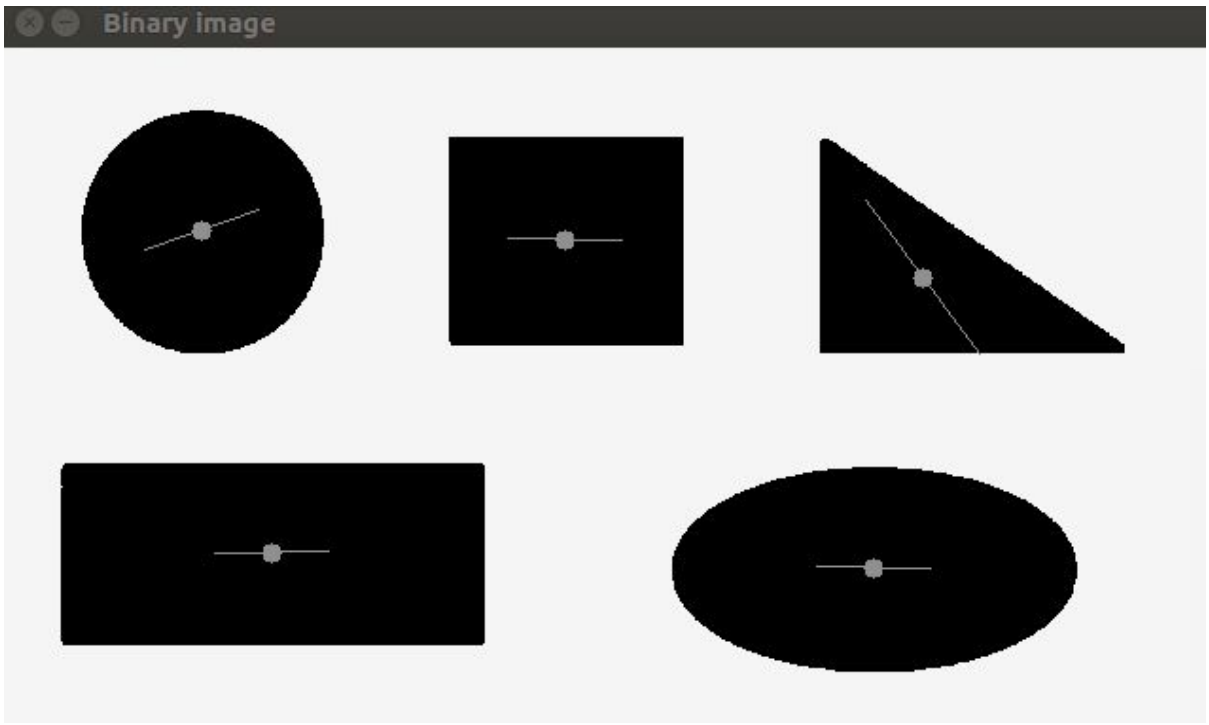
3. In Question 3, the binary image was created with the *threshold* function. Then uses the *Canny* function to find the step edges, then uses the *findContours* function to find the contours. When an object is clicked, it is recognized by the pointPolygonTest function that checks if a point is inside a contour. Then the centroid and eccentricity are calculated as the formulated shown in class, and it draws a circle in the centroid. The main axis was found using the orientation angle of the region and the reduced line equation (y = ax + b).

Results:



Eccentricity: 0.850384

It was found in this image an apparently correct centroid and a high eccentricity, as expected for this type of image. But one wrong main axis.



Circle:
Eccentricity: 9.67096e-07

Square:

Eccentricity: 0.0139741

Triangle:
Eccentricity: -0.0945561

Rectangle
Eccentricity: 0.471213

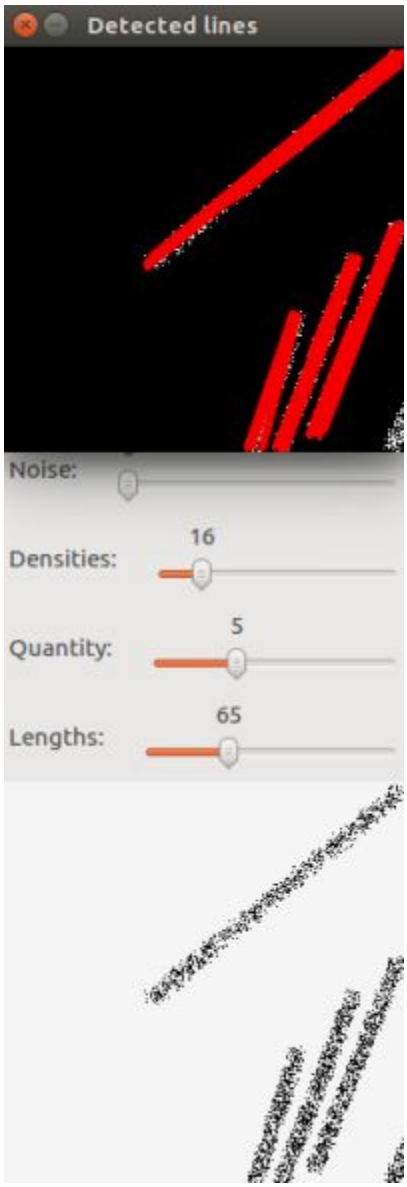
Ellipse:
Eccentricity: 0.349619

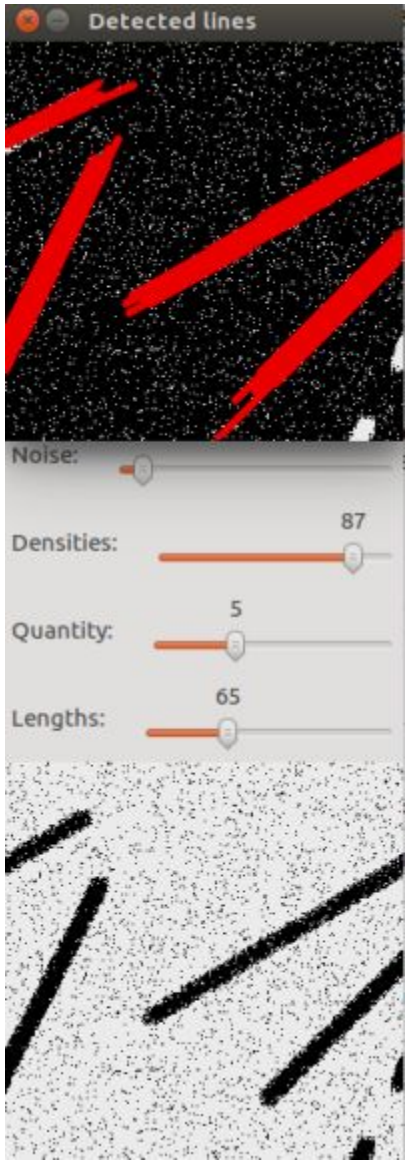
It was found in all this images an expected centroid, eccentricity, and main axis.

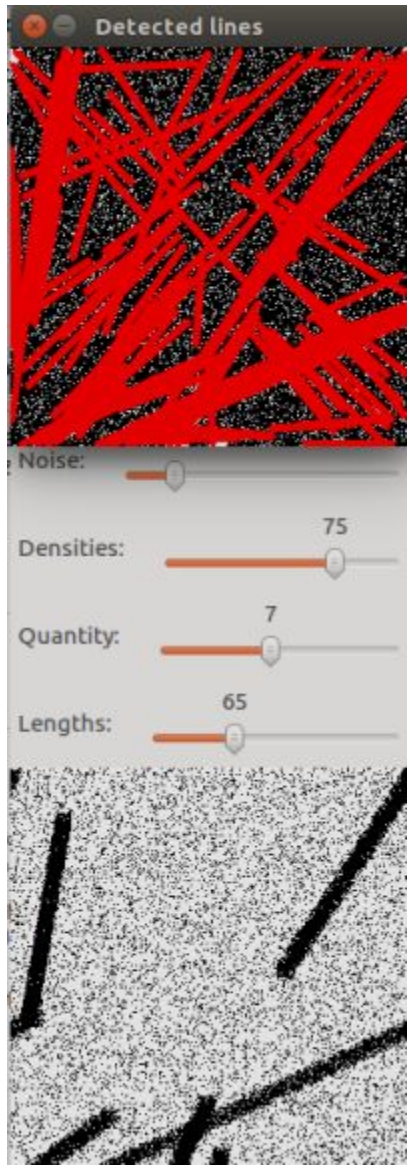
4. The lines were created using the Hessian Normal Form ($d = x + y \cos\alpha \sin\alpha$). They were detected using Probabilistic Hough Transform Line through *HoughLinesP* function.

Results:

For low noise detection was satisfactory, even at low densities. But the increase in noise creates several false positives. It was tried to reduce the noise with the box Filter but this created even more false positives.


Detected lines
Noise:
Densities:
16
Quantity:
5
Lengths:
65


Detected lines
Noise:
Densities:
87
Quantity:
5
Lengths:
65


Detected lines
Noise:
Densities:
75
Quantity:
7
Lengths:
65

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

using namespace cv;
using namespace std;

vector<vector<Point> > contours;
vector<Vec4i> hierarchy;

int const MAX_VALUE = 255;
enum Menus { menuInicial, menuOpcao1,menuOpcao2};
Menus menuAtual=menuInicial;

void putOptions(Mat img, string textOption1, float origScale) {
    int fontFace = FONT_HERSHEY_COMPLEX_SMALL;
    double fontScale = 0.7;
    int thickness = 1;
    int baseline = 0;
    Size textSize = getTextSize(textOption1, fontFace,
                                fontScale, thickness, &baseline);
    baseline += thickness;
    Point textOrig((img.cols - textSize.width) / 2,
                   (img.rows + img.rows / origScale + textSize.height) / 2);
    putText(img, textOption1, textOrig, fontFace, fontScale,
            Scalar::all(150), thickness, 8);
}

int coutBlackPixels(Mat img){
    int sum=0;
    for (int j =0;j<img.cols;j++)
        for (int i =0;i<img.rows;i++)
            if (img.at<uchar>(i,j)==0){
                sum++;
            }
    return sum;
}

float diameter(vector<Point> points){
    float diameter=0;
    for(int i=0; i<points.size();i++){
        for(int j=0; j<points.size();j++){

            if(diameter*diameter<pow(points[i].x-points[j].x,2)+pow(points[i].y-points[j].y,2)){

                diameter=sqrt(pow(points[i].x-points[j].x,2)+pow(points[i].y-points[j].y,2));
            }
        }
    }
    return diameter;
}


void onMouse(int event, int x, int y, int flags, void* param)
{
    if ( event == CV_EVENT_LBUTTONDOWN  && menuAtual==menuOpcao2)
    {
        double a;
        for (int i=0 ;i<contours.size();i++) {
            a = pointPolygonTest(contours[i], Point2f(x, y), true);
            if (a>0) {
                double perimeter = arcLength(contours[i], true);
                double area = contourArea(contours[i], false);
                double diamete =diameter(contours[i]);
                cout<<"Perimeter: "<<perimeter<<endl;
                cout<<"Area: "<<area<<endl;
                cout<<"Diameter: "<<diamete<<"\n"<<endl;
                break;
            }
        }
    }
}
```

```
 72
 73  void writeMenu(Mat img){
 74      putOptions(img,"1. Counting components",2);
 75      putOptions(img,"2. Geometric features of a selected component",1.7);
 76  }
 77
 78
 79  int main(int argc,char** argv){
 80      Mat img = imread(argv[1], CV LOAD IMAGE GRAYSCALE);
 81      if(!img.data)
 82          return -1;
 83
 84      Mat imgBin,imgBinInv;
 85      int thresholdValue = 240;
 86      int thresholdType = 0;
 87      blur(img, img, Size(3,3));
 88
 89      threshold( img, imgBin, thresholdValue, MAX VALUE,thresholdType );
 90      threshold( imgBin, imgBinInv, 40, MAX VALUE,1 );
 91
 92      Mat currentImg=imgBin.clone();
 93
 94      namedWindow("Binary image", CV WINDOW AUTOSIZE );
 95      Mat imgMenu = currentImg.clone();
 96      writeMenu(imgMenu);
 97      imshow("Binary image", imgMenu);
 98
 99      setMouseCallback( "Binary image", onMouse, 0 );
100      char a='a';
101      while(a!=27){
102          a=waitKey(0);
103          switch (menuAtual){
104              case menuInicial: {
105                  imgMenu = currentImg.clone();
106                  writeMenu(imgMenu);
107                  imshow("Binary image", imgMenu);
108                  if (a == '1') {
109                      menuAtual = menuOpcao1;
110                      Mat imgOpcao1 = currentImg.clone();
111                      putOptions(imgOpcao1, "1. White < Black", 2);
112                      putOptions(imgOpcao1, "2. Black < White", 1.7);
113                      imshow("Binary image", imgOpcao1);
114                  } else if (a == '2') {
115                      menuAtual = menuOpcao2;
116                      imshow("Binary image", currentImg);
117                  }
118                  break;
119              }case menuOpcao1: {
120                  if (a == '1') {
121                      cout << "Black pixels " << coutBlackPixels(imgBinInv) << endl;
122                      imshow("Binary image", imgBinInv);
123                      currentImg = imgBinInv.clone();
124                  } else if (a == '2') {
125                      cout << "Black pixels " << coutBlackPixels(imgBin) << endl;
126                      imshow("Binary image", imgBin);
127                      currentImg = imgBin.clone();
128                  }
129                  menuAtual = menuInicial;
130                  break;
131              }case menuOpcao2:{
132                  Mat canny output;
133                  Canny( currentImg, canny output, 100, 200, 3 );
134                  /// Find contours
135                  findContours( canny output, contours, hierarchy, CV RETR TREE,
                     CV CHAIN APPROX SIMPLE, Point(0, 0) );
136
137              }
138          }
139      }
140  }
```

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

using namespace cv;
using namespace std;

void produceSmoothedResidualToBoxFilter(Mat img,Mat R[],Mat S[],int n){
    blur(img, S[0], Size( 3, 3 ));
    R[0]=img-S[0];

    for (int i=1;i<n;i++){
        blur(S[i-1], S[i], Size( 3, 3 ));
        R[i]=img-S[i];
    }
}

void produceSmoothedResidualToMedianFilter(Mat img,Mat R[],Mat S[],int n){
    medianBlur(img, S[0],3);
    R[0]=img-S[0];

    for (int i=1;i<n;i++){
        medianBlur ( S[i-1],S[i], 3 );
        R[i]=img-S[i];
    }
}

Mat constructCoOccurrenceMatriceA4(Mat img,int Gmax){
    Mat coOccurrence(Gmax,Gmax,CV_32SC1,Scalar(0));
    for(int i = 0;i<img.rows;i++){
        for(int j=0;j<img.cols;j++){
            int x=img.at<uchar>(i,j);
            if (i+1<img.cols){//bottom
                int y=img.at<uchar>(i+1,j);
                coOccurrence.at<int>(y,x)++;
            }
            if (j+1<img.rows){//right
                int y=img.at<uchar>(i,j+1);
                coOccurrence.at<int>(y,x)++;
            }
            if (i>0){//top
                int y=img.at<uchar>(i-1,j);
                coOccurrence.at<int>(y,x)++;
            }
            if (j>0){//left
                int y=img.at<uchar>(i,j-1);
                coOccurrence.at<int>(y,x)++;
            }
        }
    }
    return coOccurrence;
}

int getSumCoOccurrenceMatrice(int rows,int adjacency){
    return rows*(rows-1)*adjacency;
}

float calculateHomogeneity(Mat img,int Gmax){
    Mat C=constructCoOccurrenceMatriceA4(img,Gmax);
    int n =getSumCoOccurrenceMatrice(img.rows,4);
    float hom=0;
    for(int u=0;u<Gmax;u++)
        for(int v=0;v<Gmax;v++) {
            float P = (float) C.at<int>(u, v) / (float) n;
            hom+=P/(1.0+(float)abs(u-v));
        }
    return hom;
}

float calculateUniformity(Mat img,int Gmax){
    Mat C=constructCoOccurrenceMatriceA4(img,Gmax);
    int n =getSumCoOccurrenceMatrice(img.rows,4);
```

```
76      float h=0;
77      float uni=0;
78      for(int u=0;u<Gmax;u++)
79          for(int v=0;v<Gmax;v++) {
80              float P = (float) C.at<int>(u, v) / (float) n;
81              uni+=P*P;
82          }
83      return uni;
84  }
85
86  Mat plotGraph(Mat values, int XRange[2])
87  {
88      int w = 200;
89      int h = 100;
90      Mat graph(h, w, CV 8UC3, Scalar( 0,0,0) );
91      int size = XRange[1]-XRange[0]+1;
92      int bin w = cvRound( (double) w/size );
93      for( int i = XRange[0]+1; i < XRange[1]; i++ )
94      {
95
96          line( graph, Point( bin w*(i-1), h - cvRound(values.at<float>(0,i-1)) ) ,
97                Point( bin w*(i), h - cvRound(values.at<float>(0,i)) ),
98                Scalar( 255, 255, 0), 2, 8, 0  );
99
100     }
101     return graph;
102 }
103
104 int main(int argc,char** argv){
105     int Gmax=255;
106     Mat img = imread(argv[1],CV LOAD IMAGE GRAYSCALE);
107     if(!img.data)
108         return -1;
109     namedWindow("Image", CV WINDOW AUTOSIZE );
110     imshow("Image",img);
111
112     int n=31;
113     Mat bfR[n],bfS[n];
114     produceSmoothedResidualToBoxFilter(img,bfR,bfS,n);
115
116     Mat mfR[n],mfS[n];
117     produceSmoothedResidualToMedianFilter(img,mfR,mfS,n);
118
119     Mat homBfR(1,n,CV 32F);
120     Mat homBfS(1,n,CV 32F);
121
122     Mat uniBfR(1,n,CV 32F);
123     Mat uniBfS(1,n,CV 32F);
124
125     Mat homMfR(1,n,CV 32F);
126     Mat homMfS(1,n,CV 32F);
127
128     Mat uniMfR(1,n,CV 32F);
129     Mat uniMfS(1,n,CV 32F);
130
131     for (int i=0;i<n;i++){
132         homBfR.at<float>(0,i)=100*calculateHomogeneity(bfR[i],Gmax);
133         homBfS.at<float>(0,i)=100*calculateHomogeneity(bfS[i],Gmax);
134
135         uniBfR.at<float>(0,i)=100*calculateUniformity(bfR[i],Gmax);
136         uniBfS.at<float>(0,i)=100*calculateUniformity(bfS[i],Gmax);
137
138         homMfR.at<float>(0,i)=100*calculateHomogeneity(mfR[i],Gmax);
139         homMfS.at<float>(0,i)=100*calculateHomogeneity(mfS[i],Gmax);
140
141         uniMfR.at<float>(0,i)=100*calculateUniformity(mfR[i],Gmax);
142         uniMfS.at<float>(0,i)=100*calculateUniformity(mfS[i],Gmax);
143     }
144
145     int Xrange[2]={0,n};
146
147     Mat homGraphBfR=plotGraph(homBfR,Xrange);
148     Mat homGraphBfS=plotGraph(homBfS,Xrange);
149
150     Mat uniGraphBfR=plotGraph(uniBfR,Xrange);
```

```
151       Mat uniGraphBfS=plotGraph(uniBfS,Xrange);
152
153       Mat homGraphMfR=plotGraph(homMfR,Xrange);
154       Mat homGraphMfS=plotGraph(homMfS,Xrange);
155
156       Mat uniGraphMfR=plotGraph(uniMfR,Xrange);
157       Mat uniGraphMfS=plotGraph(uniMfS,Xrange);
158
159       namedWindow("Hom. Box Fil. R", CV WINDOW AUTOSIZE );
160       imshow("Hom. Box Fil. R",homGraphBfR);
161
162       namedWindow("Hom. Box Fil. S", CV WINDOW AUTOSIZE );
163       namedWindow("Uni. Box Fil. R", CV WINDOW AUTOSIZE );
164       namedWindow("Uni. Box Fil. S", CV WINDOW AUTOSIZE );
165       namedWindow("Hom. Med. Fil. R", CV WINDOW AUTOSIZE );
166       namedWindow("Hom. Med. Fil. S", CV WINDOW AUTOSIZE );
167       namedWindow("Uni. Med. Fil. R", CV WINDOW AUTOSIZE );
168       namedWindow("Uni. Med. Fil. S", CV WINDOW AUTOSIZE );
169
170       imshow("Hom. Box Fil. S",homGraphBfS);
171       imshow("Uni. Box Fil. R",uniGraphBfR);
172       imshow("Uni. Box Fil. S",uniGraphBfS);
173       imshow("Hom. Med. Fil. R",homGraphMfR);
174       imshow("Hom. Med. Fil. S",homGraphMfS);
175       imshow("Uni. Med. Fil. R",uniGraphMfR);
176       imshow("Uni. Med. Fil. S",uniGraphMfS);
177
178
179       imshow("Image",img);
180
181       waitKey(0);
182       return 0;
183   }
```

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


using namespace cv;
using namespace std;

vector<vector<Point> > contours;
vector<Vec4i> hierarchy;

Mat binImg;

int const MAX_VALUE = 255;
int MAX_DIAMETER=100;

float diameter(vector<Point> points){
    float diameter=0;
    for(int i=0; i<points.size();i++){
        for(int j=0; j<points.size();j++){

            if(diameter*diameter<pow(points[i].x-points[j].x,2)+pow(points[i].y-points[j].y,2)){

                diameter=sqrt(pow(points[i].x-points[j].x,2)+pow(points[i].y-points[j].y,2));
            }
        }
    }
    return diameter;
}

float centralMoment(Mat S,Point2d centroid,int a,int b){
    float u=0;
    for(int i =0;i<S.rows;i++){
        for(int j =0;j<S.cols;j++) {
            if(S.at<uchar>(i,j)==0){
                u+=pow(j-centroid.x,a)*pow(i-centroid.y,b);
            }
        }
    }
    return u;
}

Point2d calculateCentroid(Mat S){
    float m00=0,m01=0,m10=0;
    for(int i =0;i<S.rows;i++){
        for(int j =0;j<S.cols;j++) {
            if(S.at<uchar>(i,j)==0){
                m00++;
                m10+=j;
                m01+=i;
            }
        }

    }

    float xs = m10 / m00;
    float ys = m01 / m00;
    Point centroid((int) xs, (int) ys);
    return centroid;
}

float eccentricity(Mat S){
    Point2d centroidS=calculateCentroid(S);
    float u20=centralMoment(S,centroidS,2,0);
    float u02=centralMoment(S,centroidS,0,2);
    float u11=centralMoment(S,centroidS,1,1);
    float e=(pow(u20-u02,2)-4*pow(u11,2))/pow(u20+u02,2);
    return e;
}

void mainAxis(Mat S,Mat img){
    Point2d centroidS=calculateCentroid(S);
```

```
72      float a=2.0*centralMoment(S,centroidS,1,1)/(centralMoment(S,centroidS,2,0)-centralMoment(S,centroidS,0,2));
73      float b=centroidS.y-a*centroidS.x;
74      Point pt1(centroidS.x-30,a*(centroidS.x-30)+b);
75      Point pt2(centroidS.x+30,a*(centroidS.x+30)+b);
76
77      line(img, pt1, pt2, Scalar(150), 1,8,0);
78  }
79
80
81
82  void onMouse(int event, int x, int y, int flags, void* param)
83  {
84      if ( event == CV_EVENT_LBUTTONDOWN  )
85      {
86          bool flag=true;
87          for (int k=0 ;k<contours.size()&&flag;k++) {
88              Mat raw_dist(binImg.size(),CV_8UC1,255);
89
90              double diamete =diameter(contours[k]);
91              if(diamete>MAX_DIAMETER){
92                  if(pointPolygonTest( contours[k], Point2f(x,y), true)>=0) {
93                      for( int j = 0; j < binImg.rows; j++ )
94                      { for( int i = 0; i < binImg.cols; i++ )
95                          {
96                              if (pointPolygonTest( contours[k], Point2f(i,j), false)>=0)
97                                 raw_dist.at<uchar>(j, i) = 0;
98                          }
99                      }
100                     flag=false;
101                     Point centroid=calculateCentroid(raw_dist);
102                     circle(binImg, centroid, 5,Scalar(150),-1, 8, 0);
103                     mainAxis(raw_dist,binImg);
104                     cout<<"Eccentricity: "<<eccentricity(raw_dist)<<endl;
105                 }
106             }
107         }
108         imshow("Binary image",binImg);
109     }
110 }
111
112 int main(int argc,char** argv){
113
114     Mat img = imread(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
115     if(!img.data)
116         return -1;
117
118     int thresholdValue = 250;
119     int thresholdType = 0;
120     blur(img, img, Size(3,3));
121     threshold( img, binImg, thresholdValue, MAX_VALUE,thresholdType );
122
123     Mat currentImg=binImg.clone();
124
125     namedWindow("Binary image", CV_WINDOW_AUTOSIZE );
126     imshow("Binary image", binImg);
127
128     setMouseCallback( "Binary image", onMouse, 0 );
129     Mat canny_output;
130     Canny( currentImg, canny_output, 100, 200, 3 );
131     findContours( canny_output, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
132     waitKey(0);
133 }
```

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


using namespace cv;
using namespace std;

int noise;
int quantity;
int lengths;
int densitie;

const int IMG SIZE=255;

Mat img (IMG SIZE,IMG SIZE,CV 8UC1,Scalar(255));

bool drawLine(){
    bool entrou=false;
    img = Mat(IMG SIZE,IMG SIZE, CV 8UC1, cvScalar(255));
    srand(time(0)); // use current time as seed for random generator
    for (int i=0;i<img.rows;i++)
        for (int j=0;j<img.cols;j++) {
            int random variable = rand()%100+1;
            if (random variable<noise)
                img.at<uchar>(i,j)=0;
        }
    if(quantity!=0&&lengths!=0&&densitie!=0)
        for (int k=0;k<quantity;k++) {
            float d= rand() % (int)sqrt(2*IMG SIZE*IMG SIZE);
            float alpha=rand()%90+1;
            alpha*=M PI/180;

            int t;
            for (int l = IMG SIZE / 2 - lengths, t = rand() % IMG SIZE; l < IMG SIZE / 2
            + lengths; l++, t++) {
                float x = t;
                float y=(d-x*cos(alpha))/sin(alpha);
                for (int d1 = ((int) x) - 5; d1 < ((int) x) + 5; d1++)
                    for (int d2 = ((int) y) - 5; d2 < ((int) y) + 5; d2++)
                        if ((rand() % 100 + 1) * 2 < densitie)
                            if (d1 < IMG SIZE && d2 < IMG SIZE && d1 > 0 && d1 > 0) {
                                img.at<uchar>((int) d1, (int) d2) = 0;
                                entrou = true;
                            }
            }
        }
    imshow("Image", img);
    return entrou;
}

void detectLines(Mat img){
    Mat colorImg;
    threshold(img,img,100,255,1);
    cvtColor(img, colorImg, CV GRAY2BGR);

    vector<Vec4i> lines;
    HoughLinesP(img, lines, 1, CV PI/180, 50, 50, 10 );
    for( size t i = 0; i < lines.size(); i++ )
    {
        Vec4i l = lines[i];
        line( colorImg, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, CV AA);
    }
    imshow("Detected lines", colorImg);
}


void on trackbar( int , void*)
{
    if(drawLine());
        detectLines(img);
}
```

```
75
76  int main(int argc,char** argv){
77      namedWindow("Image", CV WINDOW AUTOSIZE );
78      namedWindow("Detected lines", CV WINDOW AUTOSIZE );
79
80      noise=0;
81      lengths=0;
82      densitie=0;
83      quantity=0;
84
85      createTrackbar("Noise: ", "Image", &noise, 100, on trackbar);
86      createTrackbar("Densities: ", "Image", &densitie, 100, on trackbar);
87      createTrackbar("Quantity: ", "Image", &quantity, 15, on trackbar);
88      createTrackbar("Lengths: ", "Image", &lengths, 200, on trackbar);
89
90      waitKey(0);
91      return 0;
92  }
```