BABEŞ-BOLYAI UNIVERSITY CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Undergraduate thesis

Detecting straight lines using clustering algorithms

Abstract

Over the last four decades computer vision has been one of the most important research topics in computer science and due to rapid growth in technology algorithms that were developed have become more feasible since extracting useful information from images has proven to be a very complex task. The context of this thesis is the processing of sketches and 2D images, to extract straight lines that can be used later to extract certain features.

The process of extracting straight lines presented in this paper consists of three steps: preprocessing the image using an edge detector, since images can contain a large amount of information it is a good idea to remove any redundant data. A good quality edge detector will be used, namely the Canny edge detector which will output one pixel outlines. The second step is to build the set of line segments using the Hough transformation which is one of the most common ways to detect lines in an image. Because lines can also be formed by smaller miss-aligned line segments we will try to use a clustering algorithm in the final step to group them into larger ones. The implemented clustering algorithm will use as distance metric to determine the colinearity(similarity) of the line segments, the one defined by Koji Tsuda, Michihiko Minoh and Katsuo Ikeda at the Department of Information Science, Kyoto University.

In the second and third chapter a general overview of image processing will be presented. The fourth will contain the basic algorithms used to extract the lines. Continuing with the practical application containing specification and design and then in chapter 5 some experimental results are shown. This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

SEPTEMBER 2014 CRISTIAN FLORIN GHIŢĂ

ADVISOR:
ASSOC. PROF. DR. LEHEL CSATÓ

BABEŞ-BOLYAI UNIVERSITY CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Undergraduate thesis

Detecting straight lines using clustering algorithms

SCIENTIFIC SUPERVISOR:
ASSOC. PROF. DR. LEHEL CSATÓ

STUDENT:
CRISTIAN FLORIN GHIŢĂ

SEPTEMBER 2014

UNIVERSITATEA BABEŞ-BOLYAI, CLUJ–NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

Detectarea liniilor drepte folosind algoritmi de clustering

CONDUCĂTOR ŞTIINŢIFIC:
CONFERENŢIAR DR. LEHEL CSATÓ

ABSOLVENT:
CRISTIAN FLORIN GHIŢĂ

SEPTEMBRIE 2014

Contents

Chapter 1

Introduction

1.1 Research problem and objectives

The main objective of this paper is to provide a method to extract a set of 2D straight lines, mainly from sketches or even more complex digital images. These lines are to be used in a variety of applications to interpret visual data or to extract useful features. Another objective was to understand, and explain methods used for the detection of arbitrary shapes from images. Because making sketches does not always assume straight long lines but rather smaller more dense lines that form the final shape of the object, a grouping of the smaller line segments is necessary to determine the outline of the object.

1.2 Overview

Following chapters cover the basic techniques used to analyze digital images to detect line segments. In order to do so we must first understand how visual information from images is interpreted and represented within a computer, this is presented in the following chapter. The third chapter explains the fundamental algorithms used to extract line segments from images such as Hough Transform and mathematical convolution as well as using clustering algorithms to merge these segments into larger ones. Continuing in chapter four with the practical implementation, that contains the specification and overall design and architecture of the library. And finally in chapter five, some experimental results are shown.

1.3 Contribution

The purpose of this thesis is to construct a system that is able to form a working dataset of line segments from a sketch and merge these segments into larger continuous lines. Throughout the implementation of this library we make the use of the OpenCV library for basic image

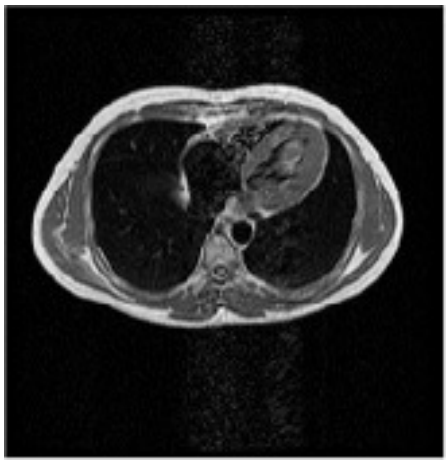
processing, and the Eigen library for matrix computations, also a hierarchical clustering algorithm is implemented and used in the final process. This algorithm will use distance measure between line segments defined in the method created by Koji Tsuda, Michihiko Minoh and Katsuo Ikeda, which is described in [rput chapter]. The library is implemented in C++11 to provide best performance and it is designed as a framework such that it can be extended easily with other clustering algorithms.

Chapter 2

Human and computer vision

Analogous to humans, computers acquire image from a sensible part called an image sensor that captures the visible light and converts it into an electronic signal, which is then processed by the $central - processing - unit(CPU)$, or in humans by the brain. The difference between how humans and computer process visual information is that humans do it in a semantic way, we extract meaningful features such as shapes, boundaries and shadows to understand an objects geometry whereas doing this with computers is still a very difficult task since most of the visual information is removed from the image. For example, removing information such as color and texture, proves to be useful in image processing, since it greatly simplifies the task of reading interesting features. There are many different image sources, as illustrated in Figure 2.1 an image can reflect information from devices that use ultrasonic audio waves to capture blood flow in the artery.[3]

a) Face from a camera[3] b) Artery from ultrasound[3]
Figure 2.1: Images from different sources.



c) Body by magnetic resonnance[3]
Figure 2.2: Images from different sources.

As we can see in the above images computer vision is an important research topic, because it allows us to observe places which otherwise without the use of computer systems would be hard or even impossible.

Chapter 3

Image processing

To know what image processing is we must first define what an image is and how it is interpreted by a computer. An image is an artifact that depicts or records visual perception, for example a two-dimensional picture, that has a similar appearance to some subject usually a physical object or a person, thus providing a depiction of it. Images may be two-dimensional, such as a photograph, screen display, and as well as a three-dimensional, such as a statue or hologram. They may be captured by optical devices such as cameras, mirrors, lenses, telescopes, microscopes, etc. and natural objects and phenomena, such as the human eye or water surfaces[1]. Captured images are typically represented in computer memory by a continuous stream of bytes as shown below, in turn these bytes form the pixels which contain the visual information to be processed.

Technically a pixel is generally thought of as the smallest single component of a digital image[1]. Having a higher pixel density in an image will yield a better quality image, but it becomes harder to process. In the context of computer imaging and color systems a pixel is a basic unit of programmable color and typically on modern computers is the size of four bytes, containing intensity values for the red, green and blue components and alpha which will be described further in this chapter. The mixture of these values is what makes the final color in the image. Mathematically an image can be represented as a function $f(x, y)$ where x, y are coordinates in the Cartesian coordinate system greater than (0,0) and less than the image width and height, respectively.

Since an image can contain large amounts of visual information, sometimes it is necessary to remove some features that we are not interested in. The purpose of the modification is not really relevant in general; the result must simply meet some user-defined criteria of "goodness".All this falls into the realm of image processing and is very common these days, especially when images are to be used by humans[4]. Typically Image processing is used to enhance an image for further use by various vision algorithms. In this paper several image enhancement techniques will be described, such as the Canny edge detector and color conversion and smoothing.

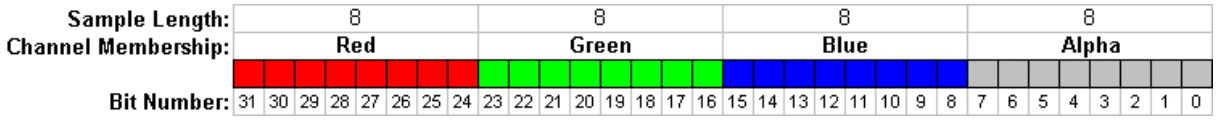
One of the most common pixel formats used is the RGBA format, often referred to as simply the color space and is actually the use of the RGB color model with an extra component, the

alpha channel. Technically a pixel representation using this format will usually contain three color intensity values and one alpha value which is used as a mask over the first three components, the lower the value of this component the less significant the values of the first three components will be. Mathematically it can be represented in the following way:

$$P = \begin{bmatrix} r & g & b \end{bmatrix} * a \tag{3.1}$$

where P is the pixel, r, g, b are color intensities with values typically between $[0, 255]$ and a can be viewed as a percentage of how much there values will matter.

In computer memory a pixel is represented as an array of bytes, as illustrated in Figure 3.1 and are usually accessed using bitwise operations for best performance since the values of a pixel can be stored in a single register on a 32 bit machine with x86 or ARM architecture. Basically image processing revolves around the alteration of pixel values.



c) In memory representation of a RGBA pixel

Figure 3.1: Pixel layout[1].

The OpenCV is a well known computer vision library that offers a vast variety of algorithms for image processing.

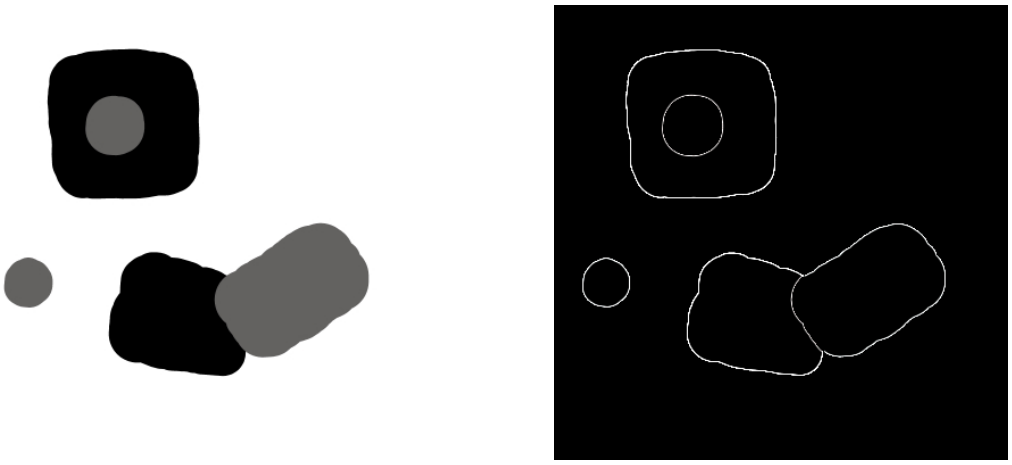
Chapter 4

Line Extraction

Sumar: *This chapter will enumerate all techniques used to form the final lines in an image.*

4.1 Edge detection

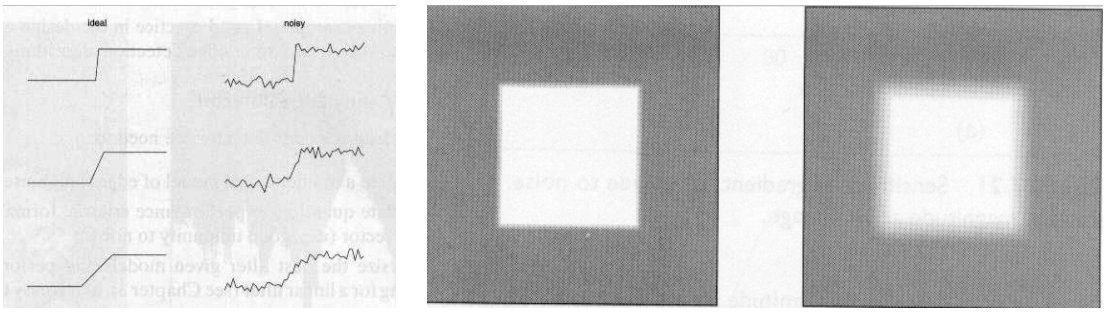
An edge is typically a boundary between two different regions in an image. These regions which may or may not belong to the same object have different brightness or color values. Using an edge detector one can retrieve the outline that separates these regions. This means that if the edge in an image can be identified accurately, all the objects can be located, and basic properties such as area, perimeter, and shape can be measured. Since computer vision involves the identification and classification of objects in an image, edge detection is an essential tool. Edge detection is a part of a process called image segmentation - the identification of regions within an image[4]. Figure **??** illustrates an example. Important features can be extracted from the edges of an image, e.g. corners, lines, curves and shapes, which then can be used by higher-level computer-vision algorithms for recognition.



a) Original image showing different shapes b) Edge-enhanced image showing outlines
Figure 4.1: Edge detection example.

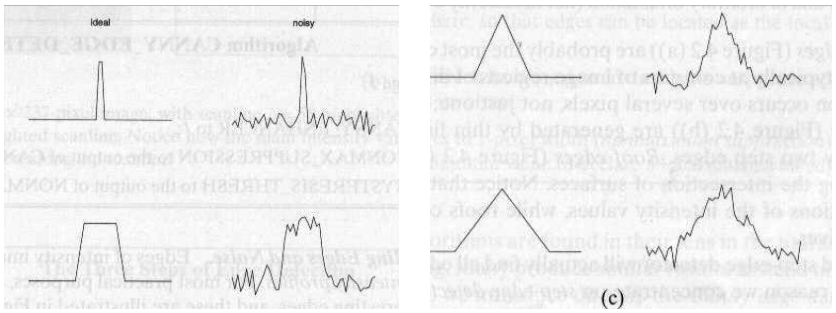
Because an edge can also be defined as a significant local change in intensity in an image it

can be modeled according to its intensity profile.



a) b)

Figure 4.2: a) Step edge, image intensity changes abruptly b) Ramp edge, intensity change is not instantaneous but occur over a finite distance[11].



a) b)

Figure 4.3: a) the image intensity abruptly changes value but then returns to the starting value within some short distance (generated usually by lines). b) a ridge edge where the intensity change is not instantaneous but occur over a finite distance (generated usually by the intersection of surfaces)[11].

These changes in intensity can be expressed using partial derivatives, points that are on the edge can be detected by finding the local maxima or minima of the first derivative and by finding the zero-crossing of the second derivative. Since images are two dimensional, it is important to consider level changes in many directions. For this reason, the partial derivatives of the image are used, with respect to the principal directions x and y[11]. The operators that are used are actually the gradient of the image, and for 2D its defined as a two dimensional vector :

$$\Delta A(x, y) = \left(\frac{\partial A}{\partial x}, \frac{\partial A}{\partial y} \right) \tag{4.1}$$

The direction and magnitude of the vector are defined as :

$$mag(\Delta A) = \sqrt{\left(\frac{\partial A}{\partial x}\right)^2 + \left(\frac{\partial A}{\partial y}\right)^2} \tag{4.2}$$

$$dir(\Delta A) = \tan^{-1}\left[\frac{\frac{\partial A}{\partial y}}{\frac{\partial A}{\partial x}}\right] \tag{4.3}$$

The magnitude of the gradient provides information about the strength of the edge and the direction of the gradient is always perpendicular to the direction of the edge[11].

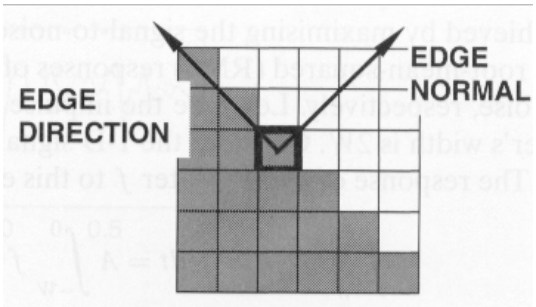


Figure 4.4: Edge magniture and direction[11].

Instead of using the derivative operator directly, a smaller, convolution mask or kernel is often used instead. Convolution is a mathematical operation on two functions f and g, producing a third function that is typically viewed as a modified version of one of the original functions[1]. The masks will attempt to model the intensity changes in the edge or approximate a derivative operator. The basic convolution algorithm on an image $f(x, y)$ with a kernel $k(x, y)$ to produce a new image $g(x, y)$:

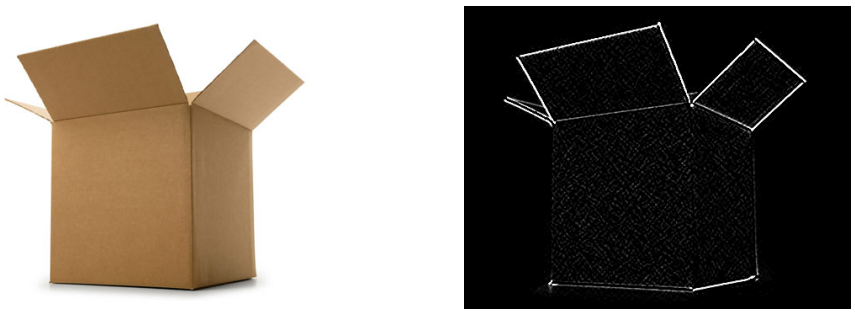
Data: $f(x,y)$, $k(x,y)$, $kernelsize, height, width$
Result: $g(x,y)$
for *y = 0 to height* **do**
  for *x = 0 to width* **do**
    $sum = 0$
    for *i = -kernelsize/2 to kernelsize/2* **do**
      for *j = -kernelsize/2 to kernelsize/2* **do**
        $sum = sum + k(j,i) * f(x-j, y-i)$
      end
    end
    $g(x,y) = sum$
  end
end

Algorithm 1: Image convolution

One of many, the Sobel edge detector for a 3 x 3 kernel on both directions[1] :

$$\begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} = Sy \tag{4.4}$$

$$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} = Sx \tag{4.5}$$



a) Original image of a box b) Edge-enhanced using the Sobel operator
Figure 4.5: The Sobel operator.

4.1.1 Canny edge detector

The Canny edge detector developed by John F. Canny in 1986 is probably one of the most widely used in computer vision. Canny specified three issues that an edge detector must address. The $Error\ rate$: the edge detector should respond only to edges, and should find all of them; no edges should be missed. $Localization$: the distance between the edge pixels as found by the edge detector and the actual edge should be as small as possible. $Response$: the edge detector should not identify multiple edge pixels where only a single edge exists[4].

For the purpose of line detection in this particular implementation, good thin edges are needed, to reduce the number of line segments detected by the $Hough\ Transform$ described in chapter... and overall complexity, that is the main reason why this operator is a good candidate. Because we will focus more on the implementation of clustering the line segments, we will use the OpenCV's function Canny. As described in http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html the operator uses a multi-stage process to detect edges :

Step 1 Smoothing the image to reduce the noise by applying a Gaussian filter. An example of what Gaussian kernel might be used:

$$K = \frac{1}{159}\begin{matrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{matrix} \tag{4.6}$$

Step 2 a. Applying the Sobel filter to find the intensity gradient in the image

$$Gx = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \tag{4.7}$$

$$Gy = \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} \tag{4.8}$$

b. Finding the gradient strength and magnitude with (**??**) and (**??**) The direction is rounded to one of four possible angles (namely 0, 45, 90 or 135)[9]

Step 3 Applying non-maxima suppression to keep only pixels that are on the edge, creating

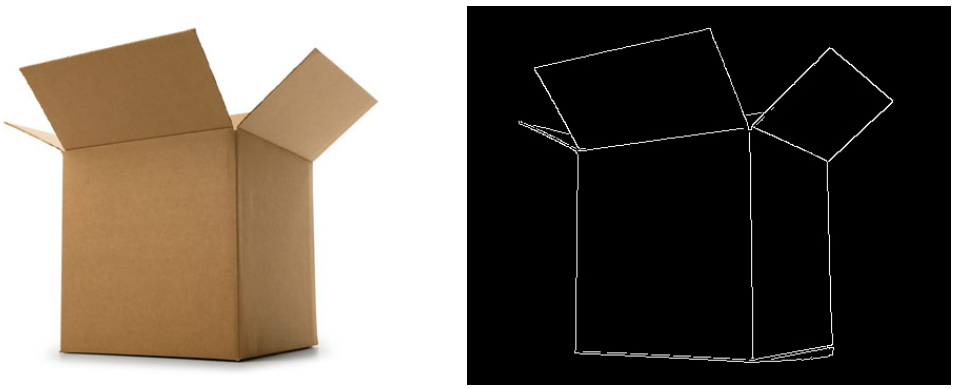
single pixel lines.

Step 4 Hysteresis thresholding :

a.If a pixel gradient is higher than the upper threshold, the pixel is accepted as an edge

b.If a pixel gradient value is below the lower threshold, then it is rejected.

c.If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the upper threshold.



a) Original image of a box b) Edge-enhanced using Canny

Figure 4.6: The Canny edge detector.

4.2 Hough Transform

The Hough Transform is a technique used in image processing to detect shapes in images, if the shape can be mathematically interpreted it can e detected. In particular, it has been used mostly to extract primitive shapes such as, lines circles and elipses. It was first introduced by Paul Hough in 1962, and then used to find bubble tracks rather than shapes in images[1]. The advantage of using this technique is that it is tolerant of imperfections in shapes and relatively unaffected by noise. It uses an $accumulator\ array$ (the Hough parameter space) and for every matching shape it finds a vote is cast into the array. The resulting peaks in the array represent strong evidence that a corresponding shape exists in the image[4].

4.2.1 Line segment detection

There are two basic implementations of Hough transform for detecting lines, one which uses the Cartesian parametrization, where collinear points in the image with coordinates (x,y) are related by their slope m and intercept b according to the following equation :

$$y = mx + b \tag{4.9}$$

In this parametrization m can take an infinite range of values, since lines can vary from horizontal to vertical. Because of this we will use the other, most used Hough transform implementation which uses Polar coordinates to define the lines, proposed by Richard Duda and Peter Hart in 1972, and overcomes the problem of having an unbounded transform space[1]. A corresponding line using this parametrization is given by :

$$y = \left(-\frac{\cos\theta}{\sin\theta} \right) x + \left(\frac{r}{\sin\theta} \right) \tag{4.10}$$

which can be arranged as

$$r = x\cos\theta + y\sin\theta, r \in R, \theta \in [0, \pi) \tag{4.11}$$

where r is the distance from the line to the origin, and θ is the angle between that line and x axis, as illustrated in Figure **??**. For a certain interval these two parameter can be represented as a sinusoid in the $(\ r, \theta\ )$ plane, shown in Figure **??**, thus a set of points will produce a set of sinusoids that will cross at coordinates :

$$x = r\cos\theta, y = r\sin\theta \tag{4.12}$$

The number of sinusoids that intersect is the actual number of votes in the two dimensional accumulator array. The accumulator array is a set of 180 bins for values of $\theta \in [0, 180)$ and $r \in [0, \sqrt{N^2 + M^2})$ where $N,\ M$ are the width and height of the image[4].

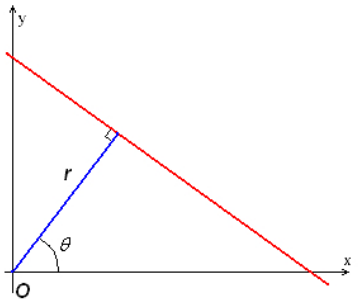


Figure 4.7: Line using Polar coordinates[1].

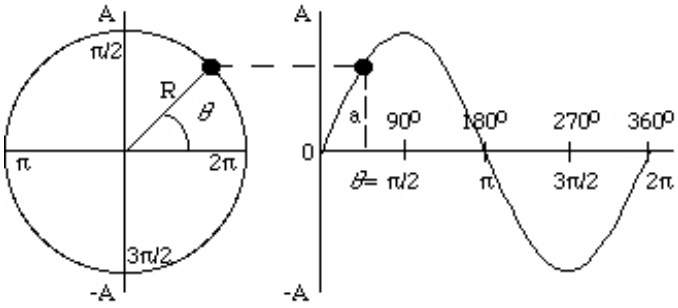


Figure 4.8: $(r,\ \theta)$ plane sinusoid[1].

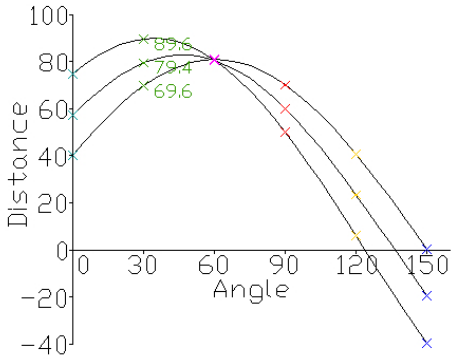


Figure 4.9: The Hough space graph[1].

In the practical application we used the OpenCV library to detect lines with the use of the HoughLine function defined as follows :

C++ : $void\ HoughLines($
$InputArray\ image, OutputArray\ lines, double\ rho,$

$double\ theta, int\ threshold, double\ srn = 0,$
$double\ stn = 0);$

The $image$ parameter is the image input of type $cv :: Mat$ and $lines$ is the output and actual accumulator two dimensional array. The rho and $theta$ parameters are used in the voting process, rho can be viewed as the line width, and theta as the angle threshold.

4.3 Cluster analysis

Cluster analysis was originated in anthropology by Driver and Kroeber in 1932 and introduced to psychology by Zubin in 1938 and Robert Tryon in 1939 and famously used by Cattell beginning in 1943 for trait theory classification in personality psychology.It is basically a way of grouping distinct objects, into individual groups, called $clusters$, such that objects that are in the same group are similar to some degree, to each other by a given criteria. Cluster analysis itself is not one specific algorithm, but the general task to be solved[1], and to better understand what constitutes a cluster, consider Figure **??** which shows several points each having a particular shape and two ways of grouping them. The points in Figure **??** a can be grouped by dividing them into two sets based on their distance between them, while the points from Figure **??** can be clustered also by their shapes also into two groups[8]. There are various clustering algorithms each of them defining the notion of a cluster in a different way, depending on the requirements of the application. Although the end result will be groups of similar objects, cluster analysis should not be confused with classification, since classification requires the classes that the objects will be associated with, known and defined beforehand, while clustering does not, but it can be regarded as a form of classification in which the grouped objects can be later labeled.

A collection of clusters is often referred to as a "clustering", usually containing all objects in the data set. Roughly, clusterings can be placed into three main categories:

Partitional A partitional clustering all the objects are placed into non-overlapping clusters, such that each object is assigned to a single cluster.

Hierarchical Hierarchical clustering basically forms a tree of data objects, in which a new branch is created when a dissimilarity is found between the compared clusters. A node in the tree represents a cluster, the union of all its subclusters and the root is the cluster that will contain all the objects.A hierarchical clustering can be viewed as a sequence of partitional clusterings and a partitional clustering can be obtained by taking any member of that sequence; i.e., by cutting the hierarchical tree at a particular level[8].

Both $partitional$ and $hierarchical$ fall into the category of $hard\ clustering$ in which the assignment of the objects to the clusters is done based on the idea that an object belongs to only one clusters.

Fuzzy In fuzzy clustering also known as $soft\ clustering$ each object can belong to more than one cluster. The assignment is based on a membership value, which can be viewed as a probability, $w \in [0, 1]$, typically between 0 and 1, 0 meaning that the object doesn't belong to the cluster and 1 absolutely belongs[8].

Depending on the nature of the data and the requirements of how the data should be grouped to be useful, different cluster types can be rendered :

Conectivity-Based or Well-Separated Clusters in this category contain objects that are closer or more similar to other objects in the same cluster than in the others. This idealistic definition of a cluster is satisfied only when the data contains natural clusters that are quite far from each other[8]. For example in Figure **??** The distance between any two points in different groups is larger than the distance between any two points within a group. Sometimes a threshold is used to specify that all the objects in a cluster must be sufficiently close (or similar) to one another. Most common algorithm to group such clusters is hierarchical clustering described that will be described in the next section.

Prototype-Based or Centroid-Based One or more prototypes are given, and each object will be closer or more similar to the prototype that forms the cluster than any other prototype.

Graph-Based If the data is represented as a graph, where the nodes are objects and the links represent connections among objects, then a cluster can be defined as a connected component; i.e., a group of objects that are connected to one another, but that have no connection to objects outside the group.

Density-Based A cluster is a dense region of objects that is surrounded by a region of low density.[4]

For the purpose of grouping line segments together, we will use a hierarchical clustering algorithm that will be described in the next section.

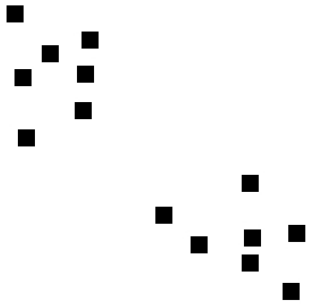


Figure 4.10: Two groups of points.

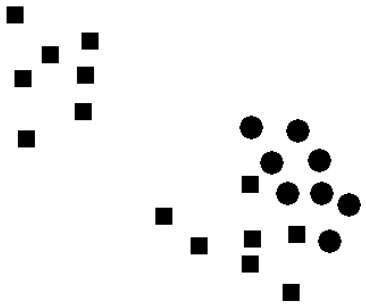


Figure 4.11: Points with different shapes.

4.3.1 Similarity and dissimilarity measures

The relationship,or similarity, between the objects that will be grouped together is very important as almost all clustering algorithms rely on some distance measure. Depending on what data is to be clustered, choosing the correct distance measure is essential to output well bounded clusters. A distance function D on a dataset X is a binary function that satisfies the following conditions[4] :

1. $D(x, y) >= 0$
2. $D(x, y) = D(y, x)$
3. $D(x, y) = 0 \Leftrightarrow x = y$

4. $D(x, y) <= D(x, z) + D(z + y)$

where x, y, and z are arbitrary data points in X. A distance function is also called a metric, which satisfies the above four conditions.[6]

For the purpose of grouping line segments the following distance measure defined in [2] can be used.:

$$d = \frac{l(Td - d/l)(Ts - s)(Tc - c)}{TdTsTc} \tag{4.13}$$

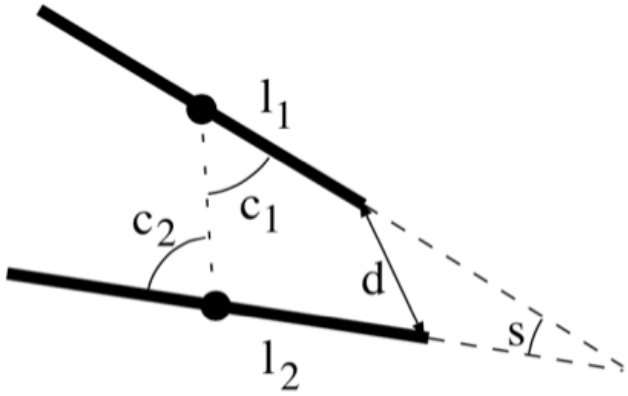


Figure 4.12: "Colinearity" of two lines[2].

where l is the average length between the line segments l1 and l2 d is the distance between two nearest end-points, s is the angle between two segments and c is the average between c1 and c2 [2]. The purpose of l and d is to ensure that the line segments to be merged are not too far apart on the horizontal and vertical axis, while the c and s values are used to ensure "collinearity" of two line segments. By "collinearity" we mean that a segment has the same orientation with respect to the other and the centers of each segment that form a third line which will be collinear with both l1 and l2.

4.3.2 Hierarchical clustering

A clustering algorithm of this type aims to divide a collection of objects into a sequence of nested subsets, that form a hierarchy of clusters. Each level in the hierarchy contains a collection of objects that are similar to each other with the value returned by the distance measure used. The actual output can be represented by a dendrogram, shown in Figure **??**. A dendrogram (from Greek dendron "tree" and gramma "drawing") is a tree diagram frequently used to illustrate the arrangement of the clusters produced by hierarchical clustering[1]. There are two types of hierarchical clustering algorithms :

Agglomerative In this category, at the beginning a cluster is created for each objects in the dataset. As the closest clusters are grouped together a new cluster emerges, this is repeated until all objects are grouped into one cluster. The hierarchy within the final cluster has the following properties: Clusters generated in early stages are nested in those generated in later stages. Clusters with different sizes in the tree can be valuable for discovery[7].

Divisive In contrast, divisive hierarchical clustering will start off using a single cluster which contains all the objects in the dataset. Two new clusters are formed by splitting larger cluster until each object is in its own cluster.

When using hierarcihcal clustering algorithms we also have to think about the linkage criteria of the clusters, which defines when the clusters should be merged and at what distance measure. In this sense we can categorize hierarchical clustering algorithms:

single linkage algorithms where clusters are merged together using this criteria $min\{d(a,b) : a \in A,\ b \in B\}$

complete linkage algorithms $max\{d(a,b) : a \in A, b \in B\}$,

average linkage algorithms $\frac{1}{|A||B|}\sum_{a \in A}\sum_{b \in B} d(a,b)$

Of course depending on the data, like we can create our own distance measure, we can create a custom linkage criteria, that fits best. In this chapter we will focus more on the Agglomerative hierarchical clustering which will be used to group the hough lines into larger ones, using the similarity measure described in Section **??**. Because the end result is a tree like structure, this clustering method is a good choice when viewing the data is necessary. Although this is not ideal in cases where data must be partitioned into a certain number of clusters, the advantage is that it can order the objects based on the distance measure value. There are however, solutions to overcome this problem, having the data separated into smaller clusters we can use a threshold

to cut the tree at the appropriate level, another common solution is to use statistical methods to match the data and assign the smaller clusters to larger, final clusters. The basic process of a hierarchical algorithm is written bellow:

Data: X : DataSet of objects, M : Map[distance,array of objects]
Result: M
Initialize initial clusters in C;
for *each* $X_i \in X$ **do**
| $C_i = X_i$
end
for *each Xi in X* **do**
| **for** *each Xj in X* **do**
| | D[i,j] = evaluated distance between Xi and Xj
| **end**
end
while *sizeof C < 1* **do**
| P = Pair of clusters to merge from C using linkage criteria;
| L = Linkage distance used to merge;
| M[L] = M[L] U P;
| remove P from C;
end

Algorithm 2: Basic agglomerative hierarchical clustering

Consider the following example in which 5 points are given, (1,1),(1,2),(2,2),(5,6),(6,7):

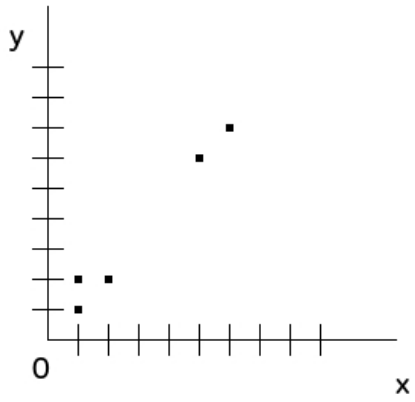


Figure 4.13: Set of points.

Applying the library implementation of the agglomerative hierarchical algorithm with the Euclidean distance as a similarity measure and single linkage, we get the following output:

42	1	1.41421	6.40312	7.81025
1	42	1	5.65685	7.07107
1.41421	1	42	5	6.40312
6.40312	5.65685	5	42	1.41421
7.81025	7.07107	6.40312	1.41421	42

1 (2,2);(1,1)

1 (2,2);(1,2);(1,1)

5 (2,2);(1,2);(1,1);(5,6)

5.65685 (2,2);(1,2);(1,1);(5,6);(6,7)

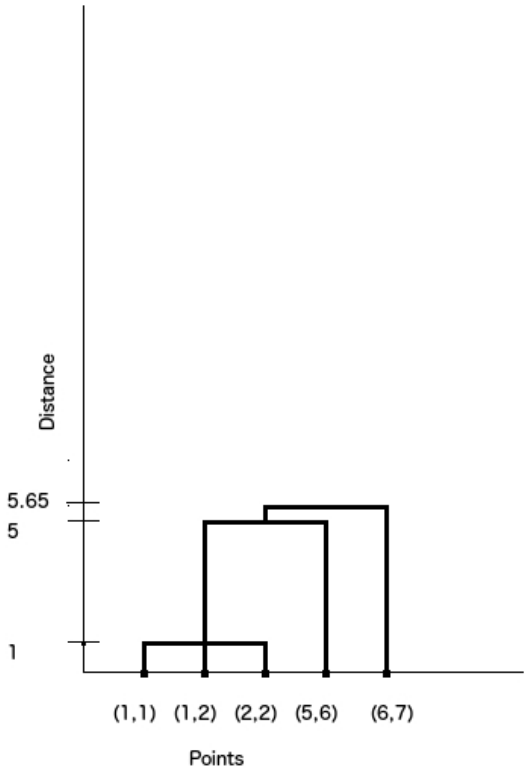


Figure 4.14: Dendrogram.

Chapter 5

Software

Sumar:

5.1 Specification

Because this library was designed as a framework, others can extend its functionality, for example new clustering algorithm can be included. The library uses template meta-programming specific to C++ and OOP principles so it is rather easy to add more classes.

To increase performance, two classes can be used to make use of multiple cores when processing the image. Images can be split into multiple smaller samples using the matrix_parallel_process class. The number of threads that run in parallel until the image(or matrix) is processed will operate on a row of the matrix. The basic processing unit is a matrix of an arbitrary size greater than 3. hough_line_dataset class can be used to return the line segments, as well as write them to the initial or a new image.

For testing purposes, we also created the random_lines class to generate line segments with random attributes(length,orientation), this can be used mainly to test the algorithms from a quantitive point of view and performance.

5.2 Design

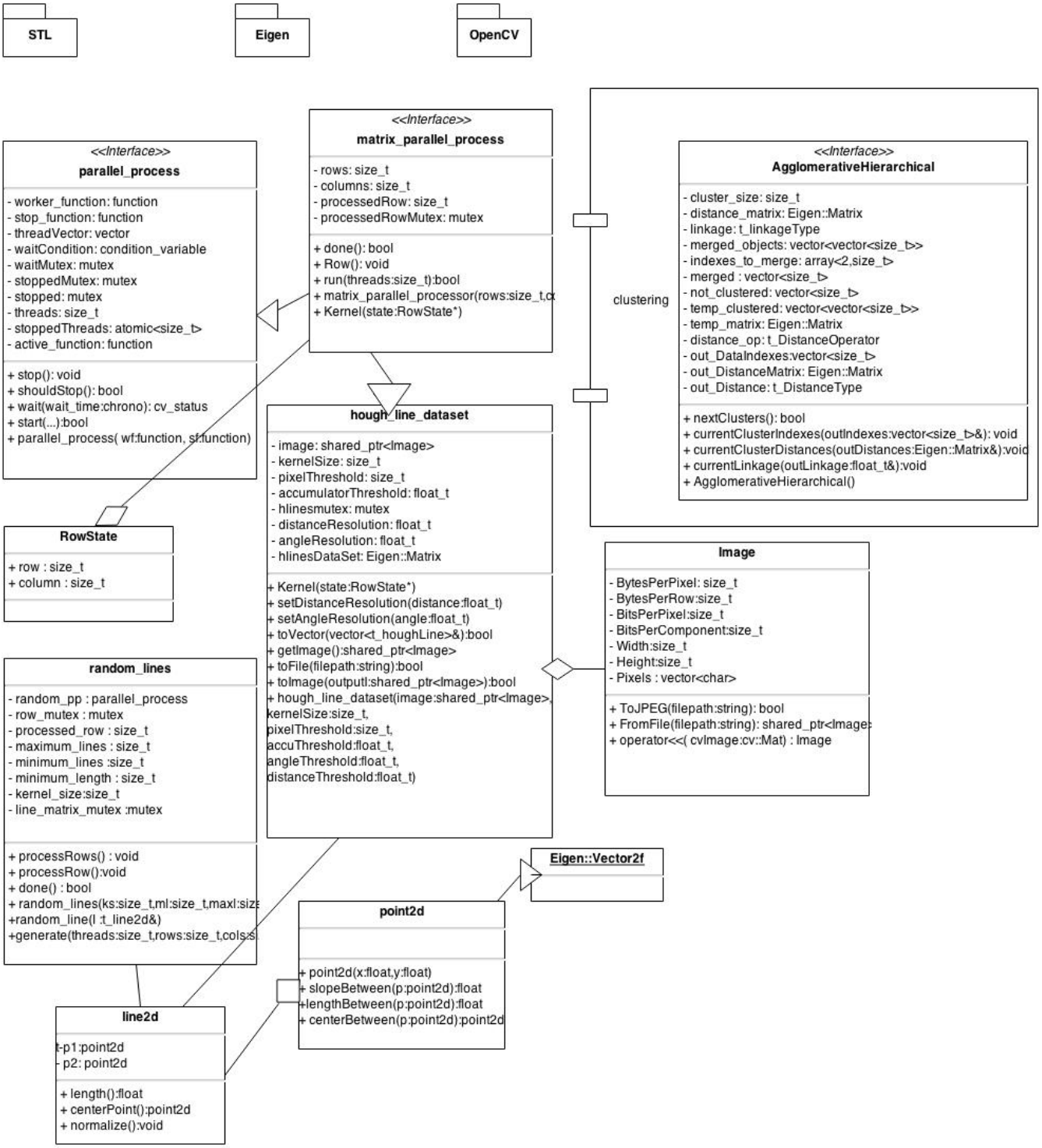


Figure 5.1: Class diagram.

Chapter 6

Experimental results

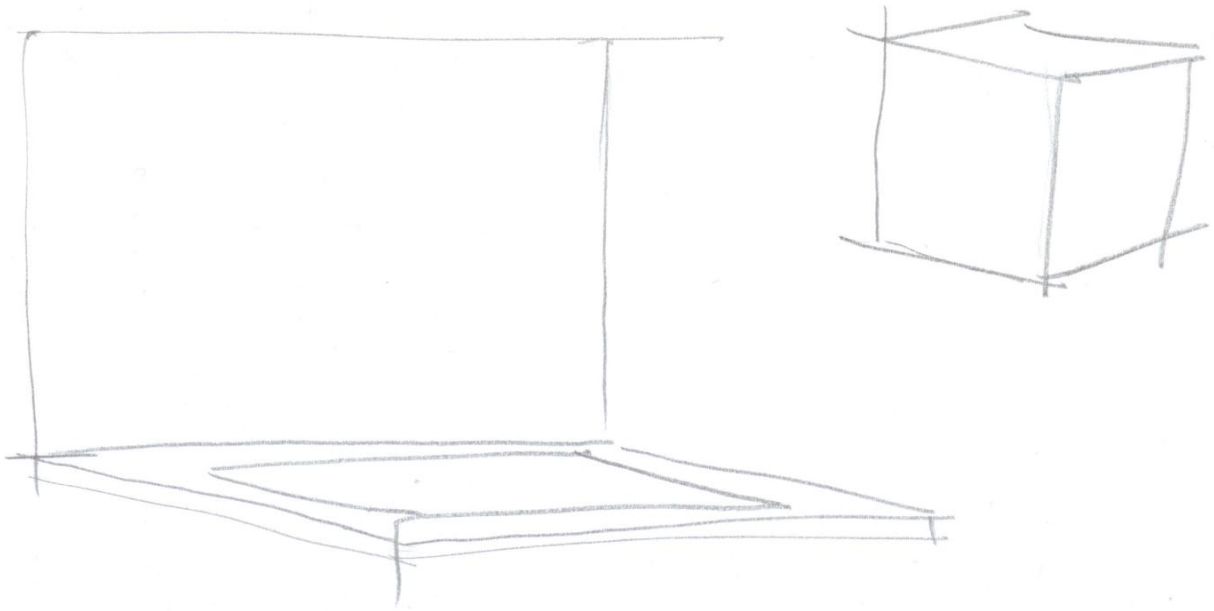


Figure 6.1: Original sketch.

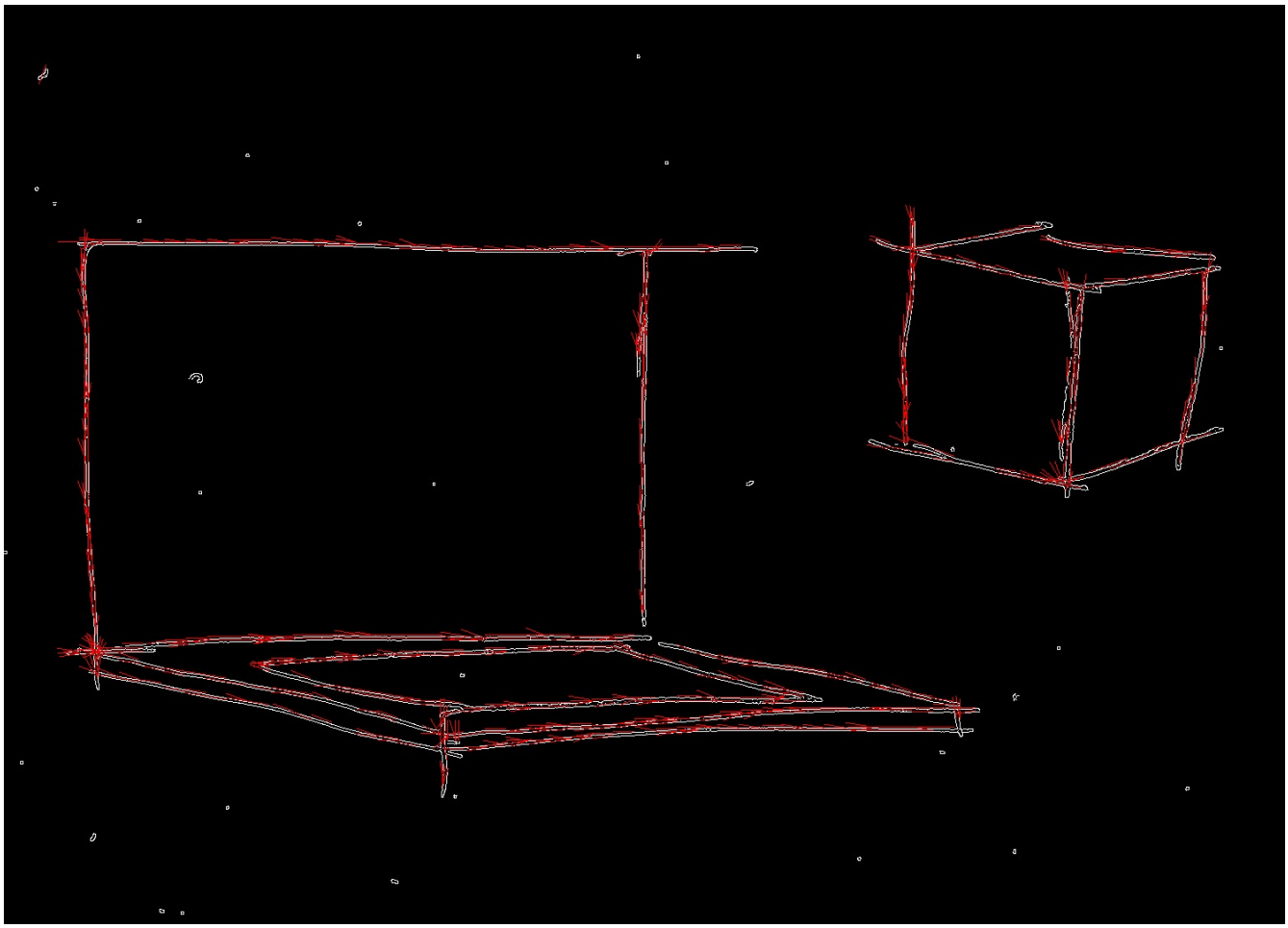


Figure 6.2: Hough lines are shown in red. White lines are the edges.

The above image shows the Canny edge detector and the line segment detection.

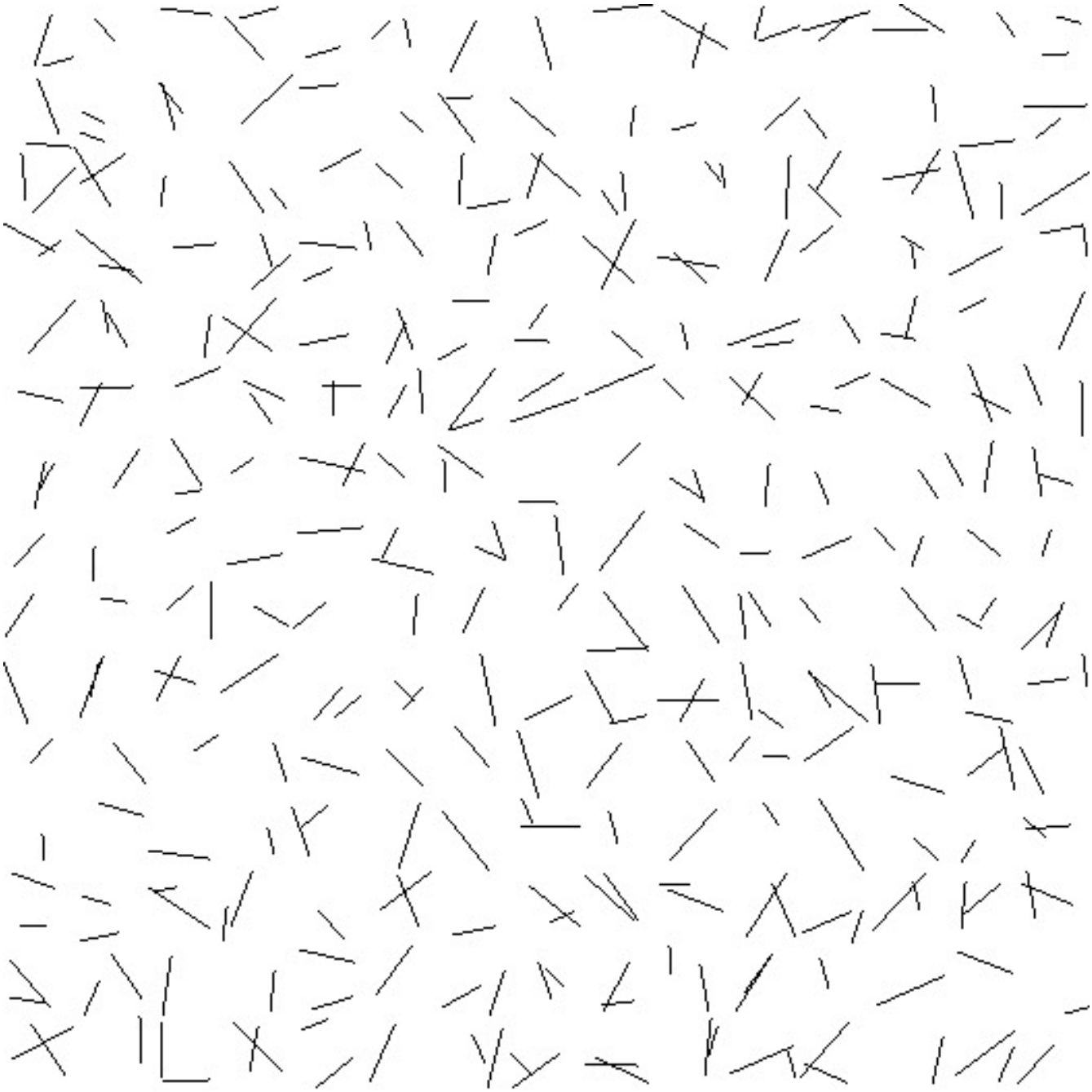


Figure 6.3: Random generated.

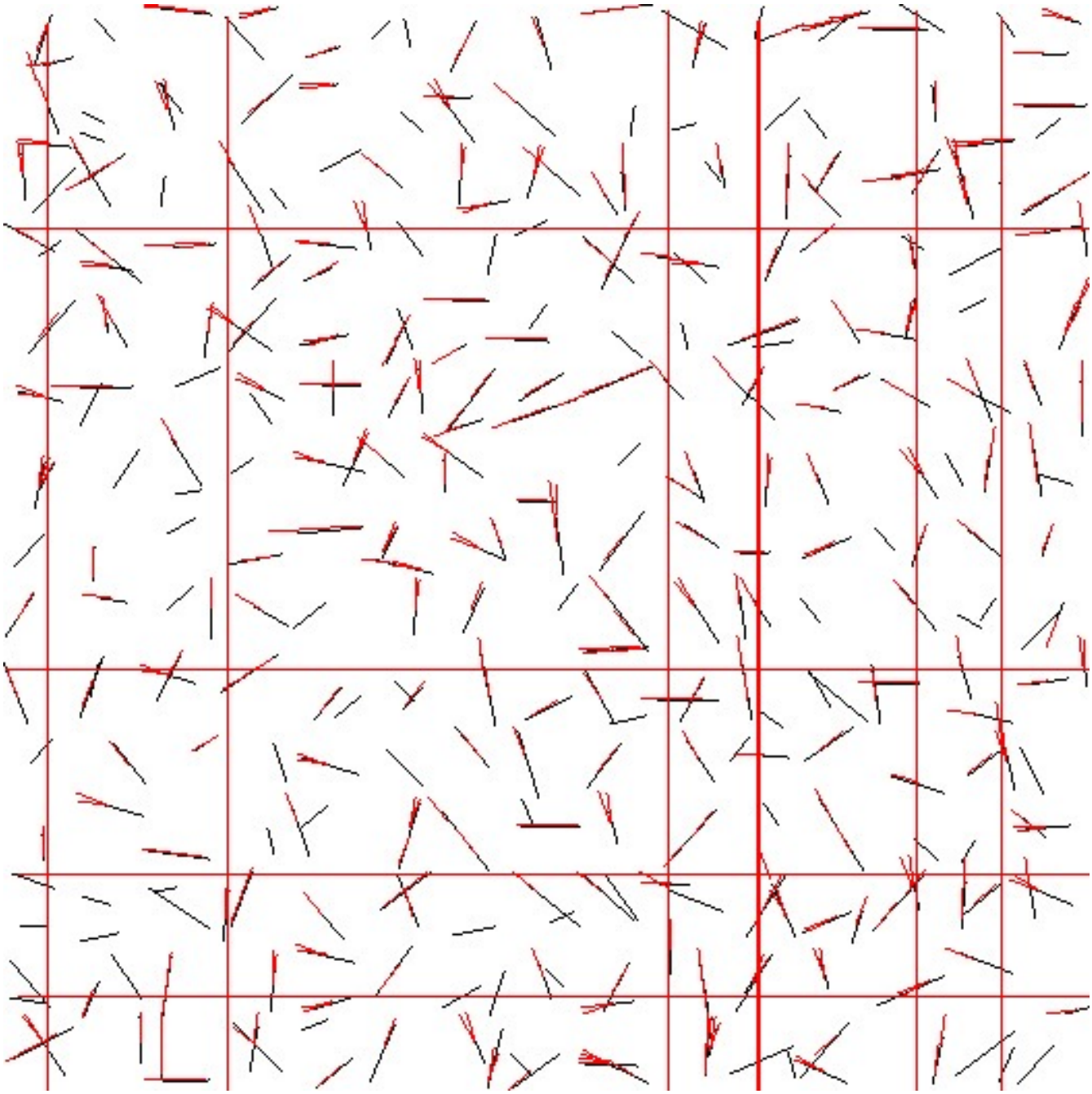


Figure 6.4: Output to random generated.

6.1 Conclusion

Bibliography

1. http://www.wikipedia.org/
2. Koji Tsuda, Michihiko Minoh and Katsuo Ikeda, Department of Information Science, Kyoto University, Kyoto 606-01, Japan, "Extracting Straight Lines by Sequential Fuzzy Clustering"
3. Mark Nixon & Alberto Aguardo, Feature Extraction & Image Processing, second edition"
4. J.R.Parker, "Algorithms for Image Processing & Computer Vision"
5. Rafael C. Gonzalez & Richard E. Woods, "Digital Image Processing", third edition
6. Guojun Gan, "Data Clustering in C++ : An Object Oriented Approach"
7. http://www.improvedoutcomes.com/docs/WebSiteDocs/Clustering/Agglomerative_Hierarchical_Clustering_Overview.htm
8. Pang-Ning Tan, Michigan State University,Michael Steinbach, University of Minnesota,Vipin Kumar, University of Minnesota, "Introduction to Data Mining"
9. http://docs.opencv.org
10. http://eigen.tuxfamily.org/dox/
11. http://www.cse.unr.edu/~bebis/

Appendix A

Using the library

```
void testEuclideanDistance(){

  vector<t_point2d> points;
  points.push_back(t_point2d{1,1});
  points.push_back(t_point2d{1,2});
  points.push_back(t_point2d{2,2});
  points.push_back(t_point2d{5,6});
  points.push_back(t_point2d{6,7});

  AgglomerativeHierarchical<vector<t_point2d>, EuclideanDistance, SingleLinkage<
      EuclideanDistance>> aggh(points.begin(),points.end());

  t_indexVector indexes;
  float_t distance = 0.0f;
  while (aggh.nextClusters()) {

    aggh.currentClusterIndexes(move(indexes));
    aggh.currentClusterDistance(distance);

    cout << distance << endl;

    for (t_indexVector::iterator i = indexes.begin(); i!=indexes.end(); i++) {
      cout << points[*i] << ";";
    }

    cout << endl;
  }
}
```

```
void testFilters(){
  Image image;

  if (!image.applySobel("/Volumes/External/Box1.jpg")){
    cout << "Failed to apply filter";
  }

  image.ToJPEGFile("/Volumes/External/Box1Sobel2.jpg", PixelFormat::RGBA);

  if (!image.applyCanny("/Volumes/External/Box1.jpg",true,5)){
    cout << "Failed to apply Canny" << endl;
  }

  image.ToJPEGFile("/Volumes/External/Box1Canny1.jpg", PixelFormat::RGBA);
}
```

```
void testRandomLines(){
  random_lines rl(25, 11, 1, 2);

  rl.generate(1, 15 , 15 );

  shared_ptr<Image> imageptr(new Image);

  rl.toFile("/Volumes/External/random_lines.txt");

  rl.toImage(imageptr);

  imageptr->ToJPEGFile("/Volumes/External/toimg.jpg", PixelFormat::RGBA);
}
```

```
void testHoughLines(){
  hough_line_dataset<RGBA> hlds(Image::FromFile("/Users/cristi/Desktop/asd.jpg"),
                                KERNEL_SIZE,
                                760,
                                KERNEL_SIZE*0.5,
                                2,
                                CV_PI/18);
  START_CLOCK
  hlds.run(1);

  hlds.wait(chrono::seconds(1000));
  STOP_CLOCK(milliseconds,"HL")
  //shared_ptr<Image> output_image(new Image);

  if (!hlds.toImage(nullptr)){
    cout << "Failed to output to image" << endl;
    return;
  }
  /*
   if (!output_image->ToJPEGFile("/Volumes/External/output_houghlines.jpg", PixelFormat::RGBA)
       ){
   cout << "FAILED TO WRITE" << endl;
   }
   */

  hlds.getImage()->ToJPEGFile("/Volumes/External/output_hlines.jpg", PixelFormat::RGBA);

  if (!hlds.toFile("/Volumes/External/line.txt") ){
    cout << " FAiled to write lines to file " << endl;
  }
}
```

```
typedef struct ColinearDistance{

  typedef float_t distance_type;

  float_t   td;
  float_t   ts;
  float_t   tc;

  ColinearDistance(float_t nd,
                   float_t ad,
                   float_t maa):
  td(nd),
  ts(ad),
  tc(maa){}

  ColinearDistance() = default;

  distance_type operator()(t_line2d& l1,
                           t_line2d& l2){

    float_t avgLength = (l1.length()+l2.length())*0.5f;
    l1.normalize();
    l2.normalize();

    float_t length_l1l2p1 = l1.p1.lengthBetween(l2.p1);
    float_t length_l1l2p2 = l1.p2.lengthBetween(l2.p2);
    float_t d = length_l1l2p1;
    float_t s = 0.0f;
    float_t c = 0.0f;

    t_line2d cline{ l1.centerPoint(), l2.centerPoint()};
    cline.normalize();

    /// calculate s
    if (length_l1l2p1 >= length_l1l2p2){
      /// from p2 to p1
      t_point2d v1{l1.p1.x()-l1.p2.x(),l1.p1.y() - l1.p2.y()},
                v2{l2.p1.x()-l2.p2.x(),l2.p1.y() - l2.p2.y()};

      s = fabs( v1.x()* v2.x() + v1.y()*v2.y() );
      s = s/(sqrtf(powf(v1.x(), 2)+powf(v1.y(), 2))*sqrtf(powf( v2.x(), 2)+powf(v2.y(), 2)));
      d = length_l1l2p2;
      s = acos(s)*DEG2RAD;
      /// create the vector c1 to c2
      t_point2d v{cline.p2.x() - cline.p1.x(), cline.p2.y() - cline.p1.y()};

      float_t c1 = fabs(v1.x()*v.x() + v1.y()*v.y());
      c1 = c1/(sqrtf(powf(v1.x(), 2)+powf(v1.y(), 2)) * sqrtf(powf(v.x(), 2)+powf(v.y(), 2)));
      c1 = acos(c1)*DEG2RAD;
      v.x() = cline.p1.x() - cline.p2.x();
      v.y() = cline.p1.y() - cline.p2.y();

      float_t c2 = fabs(v1.x()*v.x() + v1.y()*v.y());
```

```
      c2 = c2/(sqrtf(powf(v1.x(), 2)+powf(v1.y(), 2)) * sqrtf(powf(v.x(), 2)+powf(v.y(), 2)));
      c2 = acos(c2)*DEG2RAD;
      c = (c1+c2)*0.5f;

    }else{
      t_point2d v1{l1.p2.x()-l1.p1.x(), l1.p2.y() - l1.p1.y()},
                v2{l2.p2.x()-l2.p1.x(), l2.p2.y() - l2.p2.y()};
      s = fabs( v1.x()* v2.x() + v1.y()*v2.y() );
      s = s/(sqrtf(powf(v1.x(), 2)+powf(v1.y(), 2))*sqrtf(powf( v2.x(), 2)+powf(v2.y(), 2)));
      s = acos(s)*DEG2RAD;
      /// create the vector c1 to c2
      t_point2d v{cline.p2.x() - cline.p1.x(), cline.p2.y() - cline.p1.y()};

      float_t c1 = fabs(v1.x()*v.x() + v1.y()*v.y());
      c1 = c1/(sqrtf(powf(v1.x(), 2)+powf(v1.y(), 2)) * sqrtf(powf(v.x(), 2)+powf(v.y(), 2)));
      c1 = acos(c1)*DEG2RAD;
      v.x() = cline.p1.x() - cline.p2.x();
      v.y() = cline.p1.y() - cline.p2.y();

      float_t c2 = fabs(v1.x()*v.x() + v1.y()*v.y());
      c2 = c2/(sqrtf(powf(v1.x(), 2)+powf(v1.y(), 2)) * sqrtf(powf(v.x(), 2)+powf(v.y(), 2)));
      c2 = acos(c2)*DEG2RAD;
      c = (c1+c2)*0.5f;
    }

    float_t dist = avgLength*( (td-(d/avgLength)) * (ts-s) * (tc-c) );
    return dist;
  }

}ColinearDistance;

template<typename O>
class SingleLinkage{
  typename O::distance_type minimum;
public:
  explicit SingleLinkage(typename O::distance_type value) : minimum(value){}
  bool operator()(typename O::distance_type d){
    if ( d <= minimum ){
      minimum = d;
      return true;
    }
    return false;
  }
  typename O::distance_type Value(){ return minimum; }
  void Value(typename O::distance_type d) { minimum = d; }
};


void getLines(t_lineVector& hlines){

}

typedef struct EuclideanDistance{

  typedef float_t distance_type;

  distance_type operator()(const t_point2d& p1,
                           const t_point2d& p2){
    return p1.lengthBetween(p2);
  }

  distance_type infinity(){
    return 42.0f;
  }

  EuclideanDistance& operator=(const EuclideanDistance&) = default;
  EuclideanDistance() = default;

}EuclideanDistance;
```