1-1-2013

# Implementation of Edge & Shape Detection Techniques and their Performance Evaluation

Mohammad Shahnoor Islam Khan
m329khan@ryerson.ca

**IMPLEMENTATION OF EDGE & SHAPE DETECTION TECHNIQUES AND THEIR PERFORMANCE EVALUATION**

by

Mohammad Shahnoor Islam Khan
BSc, BUET, Dhaka, Bangladesh, 2001

A thesis
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Master of Science
in the Program of
Computer Science

Toronto, Ontario, Canada, 2013

**AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS**

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

MOHAMMAD SHAHNOOR ISLAM KHAN

**IMPLEMENTATION OF EDGE & SHAPE DETECTION TECHNIQUES AND THEIR PERFORMANCE EVALUATION**

Mohammad Shahnoor Islam Khan

MSc, Computer Science, Ryerson University, 2013

## ABSTRACT

In this thesis, we develop an industrial image processing application for baked goods. Images of baked objects on a conveyor belt are taken by high resolution cameras batch wise throughout the baking period. The network is designed with high performance equipments and the application is fast enough to complete all the steps within the allowed time window. The application uses Canny edge detection method [6] which optimizes the performance compared to other applications used in the industry. We have analyzed the performance of different key properties; such as processing time of an image, dimensions of an object shape, average color value to detect if an object is properly burned or damaged. Performance in detecting shapes shows higher accuracy for the developed application against other applications used in the baking industry.

# Acknowledgments

My utmost gratitude goes to my supervisor, Dr. Jelena Mišić for accepting me as her Masters student and guiding me with patience and encouragement. She has been abundantly helpful and has assisted me in numerous ways. Without her continual support and thoughtful mentoring, this thesis would not be possible.

Moreover, my appreciation goes to my friends in our lab who were there when I needed help.

I dedicate this thesis to my family: my wife, my son and my parents who supported me unconditionally and have been the source of motivation and inspiration for me and made my time more enjoyable.

*To my family for their kindness and encouragement.*

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background

The term baked or bakery products is applied to a wide range of food products, including breads, bagels, English muffins, and donuts and many other products. The antiquity of baked products goes long back in time when humans first learnt how to bake with cereal grains to improve their palatability and digestibility. All kinds of bread production undergoes heat processing normally referred as baking which causes changes in both form and structure. This is certainly true for the many different base products manufactured in bakeries. There are many different characteristics of baked products used for quality control and to determine the freshness of the baked product. Some of important characteristics (mostly concerning to this project) are listed below:

- Size: The simplest way is to measure the dimension (normally fixed) of the pan in which the product is baked. Even then many baked products have a

variable size due to domed shape; the numbers are highest at the beginning and rapidly decrease as production goes on. The variation normally occurs upward due to gas retention and batter (i.e. hot air) expansion, which is directly proportional to product volume, can be assessed quickly in terms of height. It is usually desirable that the overall shape of the final baked product should be uniform, as it may have been packed and distributed at many different locations. One way is to measure at random points or time, which is costly in terms of time and resources (man-power) and provides at best a partial assessment of the total production.

- Color: This tells if the product is baked well or not. To check if the product has been backed for a predefined time, we take the average color of the pixels of the center part of the object. Our main focus is on the average brightness of the pixels, if it is too high then the product is under cooked or if it too low that means it is over cooked or burnt.

- Structure or Shape: The shape of the finished product is very important, as it can become distorted or completely spoiled due to oven malfunctioning or improper use of ingredients. Such items are considered as damaged and will not be included in the total production of the day.

- Temperature: The temperature of the final baked product should be in the desired range depending on the actual product. The temperature is measured and recorded using industrial infrared camera(s).

Products of different kinds are formed by mixing necessary ingredients and kneading them to form the dough which is then shaped as required and baked at a predefined temperature for a predefined time to obtain the final bakery products. However, due to inherent variability of the parameters of these activities, individual items may not conform to specifications in terms of shape, size, or temperature. For example, a bread loaf may lose its shape before or during the baking process, and may not be subject to the predefined temperature or time combination. At present, conformance with specification is verified by manual inspection of product samples. This is not only costly and time-consuming, but is also incomplete, as not all items produced are actually inspected. Furthermore, manual inspection of samples can't provide immediate feedback information to allow for real-time adjustment of process parameters, even though the automated baking equipment is capable of accepting and making such adjustments. As a result, values of different process parameters are often set with ample margins just to be on the safe side, resulting in excessive use of ingredients and unnecessary energy consumption, whilst still not being able to guarantee that all items produced actually conform to the specifications.

The food industry, like other manufacturing sectors, has been trying to automate the quality control processes in order to decrease production costs and increase the quality and uniformity of the production. To increase the quality of their products while reducing waste and manual labor costs, manufacturers in food industries are leveraging machine vision systems. Most of the external quality attributes of a product can be inspected visually before the packaging line and items that do not

satisfy the set standards are automatically rejected. Machine vision based systems are of particular interest when it comes to measuring the superficial characteristics of a product for classification purposes. Such machine vision systems have been used over a wide variety of inspection applications in the food manufacturing industry including fruits and vegetables, meat, bakery products and prepared consumer foods [26].

When compared with manual inspection techniques, the benefits of using such systems provide cost-effective means to evaluate the quality of food products as they ensure a high level of consistency. Today, different types of machine vision systems perform these inspection tasks. For high speed inspection, baked products can be evaluated by inspecting images taken by good quality cameras [23]. To ensure items are properly baked, image processing systems can validate sizes, detect defects and evaluate color to ensure package consistency. Although the hardware on which the systems are based may be different, developers often use custom–built software to process and analyze captured images. In order to process these images, the use of proper edge and shape detection technique is the key to get quality output.

## 1.2 Problem Statement

The bakery vision industry is currently using different applications for controlling the baked food quality (physical damage, surface color etc.). The existing applications can not address all the problems as new challenges and requirements are being added everyday. Besides, commercial cost of these software is not afford-

able to all companies; especially small bakeries. To develop higher performance solution and to improve quality of industrial bakery within an affordable commercial cost is essential.

An industrial partner has approached Ryerson University with the problem of developing a prototype of integrated image capture and processing application called 'Bakery Vision Analyzer'.

The objective of the project 'Bakery Vision Analyzer' is to design and build a high performance solution to aid in quality control automation in industrial bakeries. The main task of the proposed project is to implement a highly modular vision analyzer that will

a) Obtain the image of each object taken immediately after baking, while the items are still on the conveyor belt, and then

b) Analyze the image in order to verify that the item has the desired properties i.e. shape, size and temperature.

c) Additional measurements such as laser height measurement will also be obtained and incorporated in the conformance assessment algorithm.

This application should satisfy the following requirements.

1 Stitching of two images 1.3MP each.

a. Image size is 1288x964 pixels

b. Color image is acquired once every second based on the external trigger.

c. Images are extracted from cameras after each trigger.

d. Cameras are capable of delivering 32 frames per second (FPS).

e. Image extraction will be done using FlyCapture [8] SDK that provides Windows compatible Application Programming Interface (API).

2 Develop pattern search algorithm to find predefined number of patterns in the acquired image

a. Patterns are square, oval or circular.

b. Position of objects is within predefined area.

c. Angle of object can be anywhere between -90° and +90° w.r.t. direction of the conveyor.

d. Images can be transferred in any of the three different ways mentioned below.

i. Ethernet IP.

ii. MODBUS TCP.

iii. SQL/DB structure.

3. Develop image enhancement algorithm for most effective edge detection.

a. Sobel [31] operator proves to be quite effective using Keyance [12] vision system. Any other operator that improves the quality can be considered for the the development.

b. Image averaging after Sobel filtration improves pattern search results according to Keyance vision system experiment.

   c. Effective edge detection method should be searched and applied for developing the application.

4. Develop algorithm for width and length measurement based on extracted and filtered image from previous task.

   a. Required measurement accuracy is one millimeter.

   b. Object(s) measurements should be reported in SQL/DB structure.

The application should satisfy the following optional requirements as well.

5. Develop algorithm for height measurement based on laser line distortion.

   a. Laser line analysis is done in predetermined window within acquired image

   b. Height extraction using green or red laser line analysis for given conveyor color.

   c. Required measurement accuracy is one millimeter.

6. Develop algorithm for color measurement in RGB or HSB color space.

   a. Calculation should be done using averaging the color value in predefined area.

   b. Possible conversion between RSB and HSB of a pixel color value parameter.

7. The Image manipulation routine should provide

   a. Storage of raw and filtered images.

b. Storage of good (physically not damaged) and bad (physically damaged) products.

8. Develop rejection mechanism based on all acquire data.

## 1.3 Solution Approach

The main task of the proposed project is to implement a highly modular vision analyzer that

- Obtains an image (or several images simultaneously using different imaging techniques) of each individual product item immediately after baking, while the items are still on the conveyor belt. We extract the image from camera by Flycapture [8] SDK or Windows compatible API.
- Analyzes the image (or images) in order to verify that the item has the desired properties.
- Measures the dimensions of an object including the height (by inspecting laser ray). incorporated in the conformance assessment algorithm.
- Provides real time adjustment of the parameters of the baking process by providing immediate feedback from the vision analyzer.
- Completes all the processing and calculations within an allocated time. For this project, the maximum allowed duration would be one second.

As industrial part of the project, we have developed a application which satisfies all the key requirements. The application has been developed, tested and handed

over to our industry partner for further test and commercial use bundled with appropriate documentations.

As part of the academic requirements, performance in detecting object shapes of the application was tested against that of other similar applications. For the purpose of performance evaluation, we have measured processing time, pattern of the probability of processing time, object missing rate, accuracy to calculate physical properties (damaged or perfect object , color value etc.) of our application and compared the results with other techniques used in the industry. Our main goal was to monitor if the developed application can satisfy processing time requirement and to compare the accuracy with other existing techniques used in bakery systems.

## 1.4 Thesis Contribution

We have developed an application for automated visual inspection of baked products. It can calculate length and width of a rectangular object, radius of a circular object, height of an object by inspecting laser ray and physical condition of an object. The application can trigger on alarm if an object is not properly burned or any plain of the object is damaged. The application saves all the information in a database for later analysis.

We have carried out a performance evaluation in which we have compared our technique to similar ones used in industry. The result shows that the processing time is not only well within the acceptance range but the technology is highly accurate; the processing time of the technique is within the acceptance range. In fact, our application, being based on Canny technique [6], is shown to have the

highest accuracy among a number of other existing applications in commercial use.

## 1.5 Thesis Organization

The remainder of this thesis is organized as follows. In chapter 2, we briefly describe edge detection and shape detection techniques used in image processing. Chapter 3 presents the implementation of new application by using Canny edge detection method and Aforge.net shape detection method. In chapter 4, we have compared the performance of our application with other similar applications. We will investigate the properties of some parameter value as well. Finally, Chapter 5 concludes the thesis.

# Chapter 2

# Related Work

Image processing is defined as the manipulation of image representation stored on a computer. Operations on images that are considered a form of image processing include zooming, converting to gray scale, increasing or decreasing image brightness, red-eye reduction in photographs, edge and shape detection of an object and analysis of object properties such as size and color. These operations typically involve iteration over all individual pixels in an image.

In bakery vision system, an image is usually processed in the following phases.

1. Edge Detection
2. Shape Detection
3. Object (shape) analysis by calculating different physical properties represented by the shape.

## 2.1 Algorithms used for Edge Detection

Edge detection refers to the process of identifying and locating sharp discontinuities in an image [10]. Edge detection technique is usually applied on gray–scale image. The discontinuities are abrupt changes in pixel intensity which characterize boundaries of objects in a scene. Classical methods of edge detection involve convolving [22] the image with an operator (a 2-D filter [30]), which is constructed to be sensitive to large gradients in the image while returning values of zero in uniform regions. There is an extremely large number of edge detection operators available, each designed to be sensitive to certain types of edges. Variables involved in the selection of an edge detection operator include:

- Edge orientation: The geometry of the operator determines a characteristic direction in which it is most sensitive to edges. Operators can be optimized to look for horizontal, vertical, or diagonal edges.

- Noise environment: Edge detection is difficult in noisy images, since both the noise and the edges contain high-frequency content. Attempts to reduce the noise result in blurred and distorted edges. This results in less accurate localization of the detected edges. Operators used on noisy images are typically larger in scope, so they can average enough data to discount localized noisy pixels.

- Edge structure: Not all edges involve a step change in intensity. Effects such as refraction or poor focus can result in objects with boundaries defined by a gradual change in intensity. The operator needs to be chosen to be responsive

to such a gradual change in those cases. Newer wavelet-based techniques [3] actually characterize the nature of the transition for each edge in order to distinguish, for example, edges associated with hair from edges associated with a face.

In the discussions that follow, the word derivative will refer to a spatial derivative of image pixel color value, unless otherwise specialized. There are many ways to perform edge detection. However, the majority of different methods may be grouped into two categories:

- Gradient: The gradient method detects the edges by looking for the maxima and minima in the first spatial derivative of the image. Mathematically, the gradient of a two-variable function (here the image intensity function) at each image point is a 2D vector with the components given by the derivatives in the horizontal and vertical directions. At each image point, the gradient vector points in the direction of largest possible intensity increase, and the length of the gradient vector corresponds to the rate of change in that direction.

  Spatial derivative for a $7x7$ matrix of pixel intensity values is shown in the Table 2.1. Partial derivative towards $x$ and $y$ direction of the pixels (intensity values)are calculated first by applying an operator. Then gradient value $df(x, y)$ is calculated from these partial derivatives. Clearly, the derivative shows a maximum located at the center of the edge in the original signal. This method of locating an edge is characteristic of the 'gradient filter' family of edge detection filters and includes the Sobel method. A pixel location is declared an edge location if the value of the gradient exceeds some threshold.

(a) Intensity values for pixels of an image

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 100 | 100 | 100 | 100 | 100 | 0 |
| 0 | 100 | 100 | 100 | 100 | 100 | 0 |
| 0 | 100 | 100 | 100 | 100 | 100 | 0 |
| 0 | 100 | 100 | 100 | 100 | 100 | 0 |
| 0 | 100 | 100 | 100 | 100 | 100 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(b) Partial derivative $|\frac{\partial f(x,y)}{\partial x}|$ to $x$ direction

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 300 | 0 | 0 | 0 | 300 | 0 |
| 0 | 400 | 0 | 0 | 0 | 400 | 0 |
| 0 | 400 | 0 | 0 | 0 | 400 | 0 |
| 0 | 400 | 0 | 0 | 0 | 400 | 0 |
| 0 | 300 | 0 | 0 | 0 | 300 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(c) Partial derivative $|\frac{\partial f(x,y)}{\partial y}|$ to $y$ direction

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 300 | 400 | 400 | 400 | 300 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 300 | 400 | 400 | 400 | 300 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(d) Spatial derivative $|\frac{\partial^2 f(x,y)}{\partial x \partial y}|$ of pixels

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 424 | 400 | 400 | 400 | 424 | 0 |
| 0 | 400 | 0 | 0 | 0 | 400 | 0 |
| 0 | 400 | 0 | 0 | 0 | 400 | 0 |
| 0 | 400 | 0 | 0 | 0 | 400 | 0 |
| 0 | 424 | 400 | 400 | 400 | 424 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 2.1: Spatial derivative of 7x7 pixels (border pixels are excluded for derivative calculation)

As mentioned before, edges will have higher pixel intensity values than those surrounding it. So once a threshold is set, the gradient value can be compared to the threshold value and an edge is detected whenever the threshold is

(a) Original image (b) Partial derivative $|\frac{\partial f(x,y)}{\partial x}|$ to $x$ direction

(c) Partial derivative $|\frac{\partial f(x,y)}{\partial y}|$ to $y$ direction (d) Spatial derivative $|\frac{\partial^2 f(x,y)}{\partial x \partial y}|$ of image

Figure 2.1: Spatial derivative of an image

exceeded.

Figure 2.1 shows images after applying different steps in spatial derivative.

- Second order derivative: Second order derivative, also known as Laplacian method [20], searches for zero crossings in the second derivative of the image to find edges. An edge has the one-dimensional shape of a ramp and calculating the derivative of the image can highlight its location. Furthermore,

when the first derivative is at a maximum, the second order derivative is zero. As a result, another alternative to finding the location of an edge is to locate the zeros in the second order derivative. Second order derivative of an image can be obtained by applying a suitable operator.

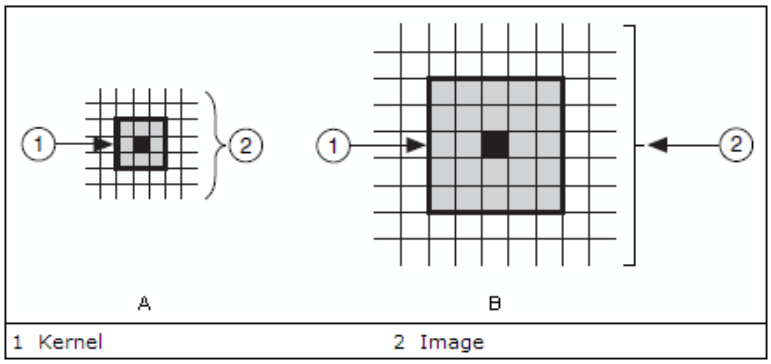
Most of the edge detection methods use convolution kernels for processing images. The pixel values in a gray-scale image is changed by a filter as per the definition of convolution kernel [22]. The convolution kernel is a 2–D structure and its coefficients define the computation of filtered value of each pixel. A convolution kernel of $n \times n$ mask is multiplied for each pixel location $(i, j)$ of image $I$ by following formula.

$$f(n,n) \star I(i,j) = \sum_{k=-n/2}^{+n/2} \sum_{l=-n/2}^{+n/2} f(k,l) \times I(i-k, j-l) \tag{2.1}$$

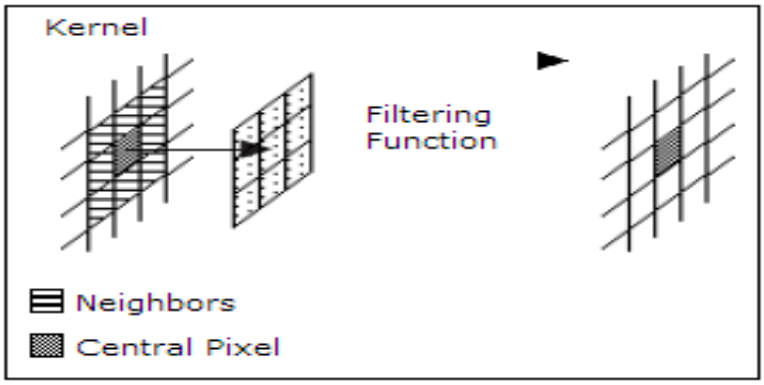
The filtered value of a pixel is a weighted combination of its original value and the values of its neighboring pixels. The convolution kernel coefficients define the weight of contribution of each neighboring pixel to the pixel being updated. The convolution kernel size determines the number of neighboring pixels whose values are considered during the filtering process.

In the case of a 3 x 3 kernel, illustrated in the left (A) of Figure 2.2(a), the value of the central pixel (shown in black) is derived from the values of its eight surrounding neighbors (shown in gray). A 5 x 5 kernel, shown in the right (B) of Figure 2.2(a), specifies 24 neighbors, a 7 x 7 kernel specifies 48 neighbors, and so forth.

In filtering operation the kernel moves from upper leftmost pixel to the lower rightmost pixel. At each pixel in the image, the new value is computed using the




(a) Kernel and Image




(b) Filtering Function

Figure 2.2: Filtering Function by applying convolution kernel

values that lie under the kernel, as shown in the Figure 2.2(b).

### 2.1.1 Robert's Cross operator

The Robert's Cross operator [5] performs a simple, quick to compute, 2-D spatial gradient measurement on an image. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point. The operator consists of a pair of 2x2 convolution kernels, as

$$G_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}; \quad G_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

Figure 2.3: Robert's Cross Operator

shown in Figure 2.3. One kernel can be obtained from the other by rotating 90°.

These kernels are designed to respond maximally to edges running at 45° to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (i.e. $G_x$ and $G_y$). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by,

$$|G| = \sqrt{{G_x}^2 + {G_y}^2} \tag{2.2}$$

The angle of orientation of the edge giving rise to the spatial gradient (relative to the pixel grid orientation) is given by, 2.3.

$$\theta = tan^{-1}\frac{G_x}{G_y} - 3\pi/4 \tag{2.3}$$

The results of this operation will highlight changes in intensity in a diagonal direction. One of the most appealing aspects of this operation is its simplicity; the kernel is small and contains only integers. However with the speed of computers today this advantage is negligible and the Roberts cross suffers greatly from sensitivity to noise.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}; \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figure 2.4: Sobel's operator

### 2.1.2 Sobel Operator

Sobel operator is a pair of 3x3 convolution kernels as shown in Figure 2.4 [31]. The second kernel is obtained by rotating the first by 90°; the two kernels are orthogonal to each other.

These kernel values are designed for maximum response to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. One can apply kernels separately to the input image, in order to produce separate measurements of the gradient component in each orientation (known as $G_x$ and $G_y$). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by Equation 2.2.

The angle of orientation of the edge (relative to the pixel grid) giving rise to the spatial gradient is given by,

$$\theta = tan^{-1}\frac{G_X}{G_Y} \tag{2.4}$$

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}; \quad G_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Figure 2.5: Prewitt operator

### 2.1.3 Prewitt Operator

Prewitt operator is similar to the Sobel operator and is used for detecting vertical and horizontal edges in images. At each point in the image, the result of the Prewitt operator is the corresponding gradient vector.

As can be seen, the result of all three (Robert's Cross, Sobel and Prewitt) operators at an image point which is in a region of constant image intensity is a zero vector and at a point on an edge is a vector which points across the edge, from darker to brighter values.

### 2.1.4 Laplacian of Gaussian Filter

The Laplace operator, also known as Laplacian, is a 2-D isotropic (i.e. does not depend on the direction) measure of the second spatial derivative of an image [13]. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection. The Laplacian is often applied to an image that has first been smoothed in order to reduce noise.

The operator normally takes a single gray level image as input and produces another gray level image as output.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}; \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}; \quad \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

Figure 2.6: Three commonly used discrete approximations to the Laplacian filter.

The Laplacian $L(x, y)$ of an image with pixel intensity values $I(x, y)$ is given by:

$$L(x, y) = \frac{\delta^2 I}{\delta x^2} + \frac{\delta^2 I}{\delta y^2} \tag{2.5}$$

Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Three commonly used small kernels are shown in Figure 2.6

Because these kernels are approximating a second derivative measurement on the image, they are very sensitive to noise. To counter this, the image is often Gaussian smoothed before applying the Laplacian filter. This pre-processing step reduces the high frequency noise components prior to the differentiation step.

In fact, since the convolution operation is associative, we can convolve the Gaussian smoothing filter with the Laplacian filter first and then convolve this hybrid filter with the image to achieve the required result. This approach has two advantages:

- Since both the Gaussian and the Laplacian kernels are usually much smaller than the image, this method usually requires far fewer arithmetic operations.

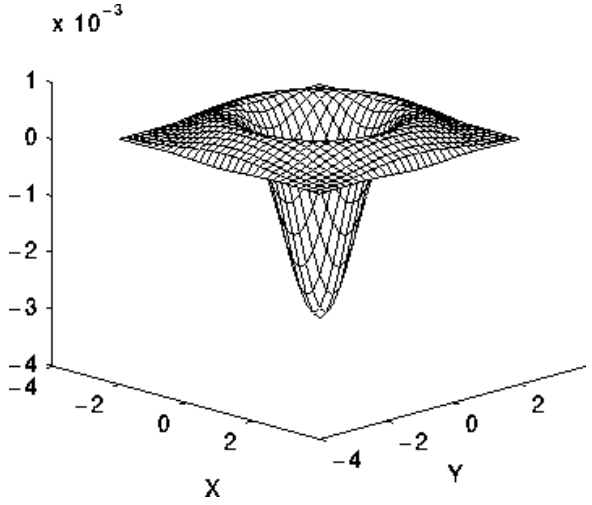



Figure 2.7: LoG centered on $(0, 0)$

- The LoG ('Laplacian of Gaussian') kernel can be precalculated in advance so only one convolution needs to be performed at run-time on the image.

The 2-D LoG function centered at $(0, 0)$ and with Gaussian standard deviation $\sigma$ has the form:

$$L(x, y) = \frac{1}{\pi\sigma^4}\left[1 - \frac{x^2 + y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.6}$$

whic is shown in Figure 2.7.

Note that as the Gaussian is made increasingly narrow by reducing $\sigma$, the LoG kernel becomes the same as the simple Laplacian kernels shown in Figure 2.6. This is because smoothing with a very narrow Gaussian ( $\sigma < 0.5$ pixels) on a discrete grid has no effect. Hence on a discrete grid, the simple Laplacian can be seen as a limiting case of the LoG for narrow Gaussians.

### 2.1.5 Canny's Edge Detection Algorithm

The Canny edge detection [6] algorithm was developed to improve the existing method of edge detection.The first and most obvious is low error rate: it is important that edges occurring in images should not be missed and that there be no response to non-edges. The second criterion is that the edge points be well localized. In other words, the distance between the edge pixels as found by the detector and the actual edge is to be at a minimum. A third criterion is to have only one response to a single edge. This was implemented because the first two were not substantial enough to completely eliminate the possibility of multiple responses to an edge.

Based on these criteria, the Canny edge detector first smoothes the image to eliminate noise. It then finds the image gradient to highlight regions with high spatial derivatives. The algorithm then tracks along these regions and suppresses any pixel that is not at the maximum (non–maximum suppression). The gradient array is now further reduced by hysteresis [19]. Hysteresis, shown in the Figure 2.8, is used to track along the remaining pixels that have not been suppressed. Canny edge follows two threshold values. If the magnitude is below the first threshold, it is set to zero (made a non–edge). If the magnitude is above the high threshold, it is made an edge. And if the magnitude is between the two thresholds, then it is set to zero unless there is a path from this pixel to a pixel with a gradient above the lower threshold value.

Figure 2.9 shows the original image and image after applied different operators to detect edges.

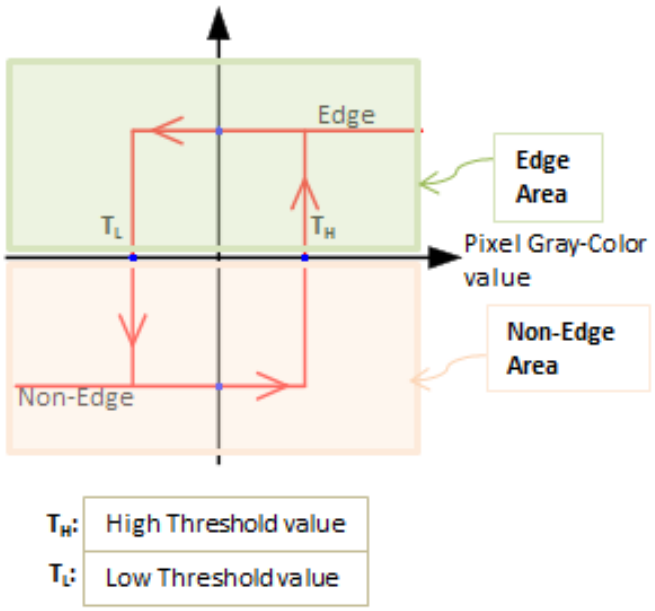



Figure 2.8: Hysteresis representing two threshold values

Canny edge detection has been improved by using type–2 fuzzy sets [4]. The two significant features of this method are improvement of NMS (non–maximum suppression) and double thresholding of the gradient image. Under poor illumination, the region boundaries in an image may become vague, creating uncertainties in the gradient image. Type–2 fuzzy based Canny edge detection technique handles uncertainties and automatically selects the threshold values needed to segment the gradient image. The results show that the algorithm works significantly well on different benchmark images as well as medical images (hand radiography images).

In chapter 3 we will explain the steps associated with Canny edge detection as we have used this technique in developing the application.


(a) Original image

(b) After applying Robert's Cross operator

(c) After applying Sobel operator

(d) After applying Prewitt operator

(e) After applying LoG operator

(f) After applying Canny method

Figure 2.9: Different edge detection methods applied on an image

## 2.2 Shape Detection Methods

Shape detection generally searches for effective and important shape features based on either shape boundary or boundary plus interior content. Various features have been developed for detecting shapes. Shape signature, signature histogram, shape invariants, moments, curvature, shape context, shape matrix, spectral features etc. are widely used techniques in detecting shapes. Accuracy of retrieving a similar shape from the designated database, is the key evaluating factor [24]. However, it is not sufficient to evaluate a representation technique only by the effectiveness of the features employed. This is because the evaluation might ignore other important characteristics of a shape representation technique such as efficiency in online retrieval.

We can generally classify all the shape representation techniques into two groups– one is contours based and the other is region based. In below, we have explained contour based shape detection techniques. In developing our application, we have considered contour based Hough Transform to detect straight lines and circular shapes.

### 2.2.1 Hough Transform

The Hough Transform (HT) method was implemented for particle physics to detect lines and arcs in the photographs obtained in cloud chambers [11]. Many elaborations and refinements of this method have been investigated since. It is applied to images which have already been freed from irrelevant detail (at some given size scale) by some combination of filtering, thresholding and edge detection.

The method attributes a logical label (set of parameter values defining a line or quadric) to an object that, until then, existed only as a collection of pixels; therefore it can be viewed as a segmentation procedure.

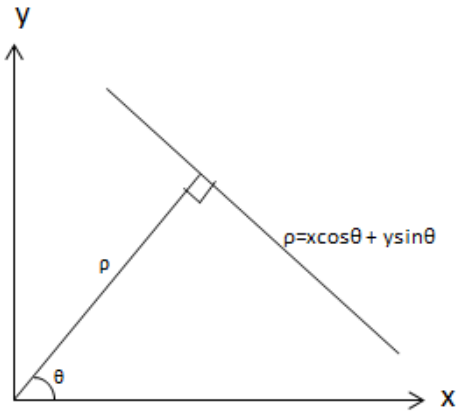
HT can detect edges consisting of straight lines, or parametric curves like circle, ellipse e.t.c. Here is a description to find straight line in a image by applying HT. All the lines passing through the point $(x', y')$ satisfy the Equation 2.7.

$$x'cos\theta + y'sin\theta = \rho \tag{2.7}$$

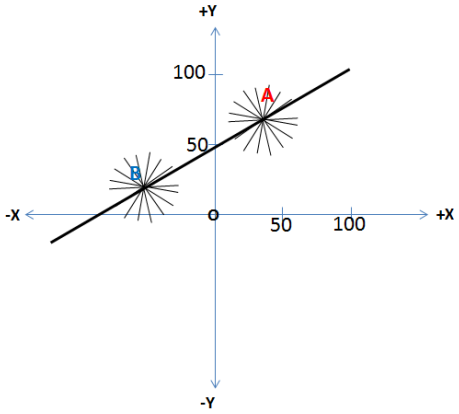
Here, $\rho$ is the perpendicular distance of the line from the origin shown in the Figure 2.10(a) and $\theta$ is the angle of $\rho$ with $X$ axis. For Equation 2.7, the values $\rho$ and $\theta$ change for different line passing through the point $(x', y')$. The lines in Figure 2.10(b) are mapped from $x - y$ space to $\rho - \theta$ space shown in Figure 2.10(c).

For detecting a straight line, HT goes for a voting system. A two dimensional array, also known as $R$–table, is required to save values $\rho$ and $\theta$ for all possible straight lines passing through a point in the edge. For each point in the edge, it adds one vote for all the edges passing through the point. The Figure 2.10(d) shows the voting matrix of HT. The cell in the Figure 2.10(d) is known as accumulator cell and can be adjusted to large or small by the users if required. All the points in a straight line will provide more votes for a particular cell that satisfy the $\rho$ and $\theta$ value for the edge. Though other cells will get some votes for noise and other non–existent edges passed through the points, maximum vote will be counted for the real edges.

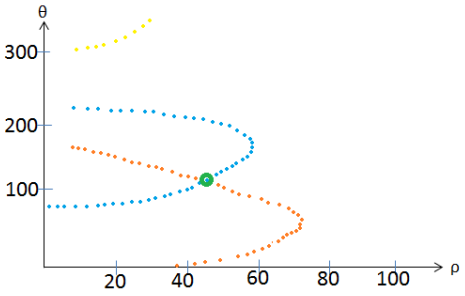
Circular, elliptical, parabolic shapes can also be detected by using HT. For circular shape, three variables are $x$ and $y$ coordinates of the centre and the radius.




(a) Straight line in polar–coordinate system




(b) Lines passing through point $A$ and $B$ in $x - y$ space




(c) HT transform of point $A$ and $B$ in Figure 2.10(b) in $\rho - \theta$ space

HT: Accumulator array for straight line (votes)

| θ \ ρ | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|---|---|---|---|
| 180 | 5 | 4 | 4 | 3 | 2 | 1 | 1 | 0 | 0 | 0 |
| 160 | 6 | 9 | 9 | 8 | 7 | 5 | 3 | 2 | 1 | 0 |
| 140 | 7 | 14 | 12 | 10 | 9 | 8 | 5 | 2 | 1 | 0 |
| 120 | 8 | 16 | 25 | 32 | 38 | 37 | 31 | 6 | 2 | 1 |
| 100 | 7 | 18 | 31 | 40 | **80** | 33 | 32 | 9 | 3 | 1 |
| 80 | 5 | 17 | 30 | 38 | 35 | 34 | 30 | 17 | 4 | 2 |
| 60 | 9 | 14 | 26 | 31 | 30 | 31 | 28 | 20 | 6 | 3 |
| 40 | 6 | 12 | 28 | 32 | 30 | 33 | 26 | 18 | 9 | 2 |
| 20 | 4 | 10 | 11 | 12 | 14 | 10 | 12 | 16 | 8 | 4 |
| 0 | 10 | 11 | 12 | 8 | 7 | 6 | 5 | 1 | 4 | 4 |

(d) Voting system for an arbitrary straight line

Figure 2.10: HT and voting for straight lines

We need three dimensional articulation points for saving these three variables. Like straight line, circular shape detection based on HT goes for the voting. Elliptical shape is having five variables: $x$ and $y$ coordinates of the centre, semi–major–axis ($a$), semi–minor–axis ($b$)and and angle $\theta$ between semi–major–axis and $x$ axis .

In summary, if a particular shape is present in the image then the mapping of all of its points into the parameter space must cluster around the parameter values which correspond to that shape. This approach maps distributed and disjoint elements of the image into a localized accumulation point, which is both a benefit and a drawback. Partially occluded shapes are still detected, on the evidence of their visible parts for example, all segments of the same circle contribute to the detection of that circle, regardless of the gaps between them. On the other hand, local information inherent in the points of the shape, such as adjacency, is lost - endpoints of circle arcs and line segments must be determined in a subsequent step.

Computational load of the method increases rapidly with the number of parameters which define the detected shape.

### 2.2.2 Generalized Hough Transform

The Hough Transform (HT) was initially developed to detect analytically defined shapes like straight lines, circular, elliptical, parabolic shapes. The Generalized Hough Transform (GHT) is used to find any arbitrary shape [2]. Many foods in baking industry do not fall into a regular shape such as heart–shaped valentine brownies. GHT can be applied in finding such irregular shapes.

Let $(X_c, Y_c)$ be a reference point in the Figure 2.11. We can generate following

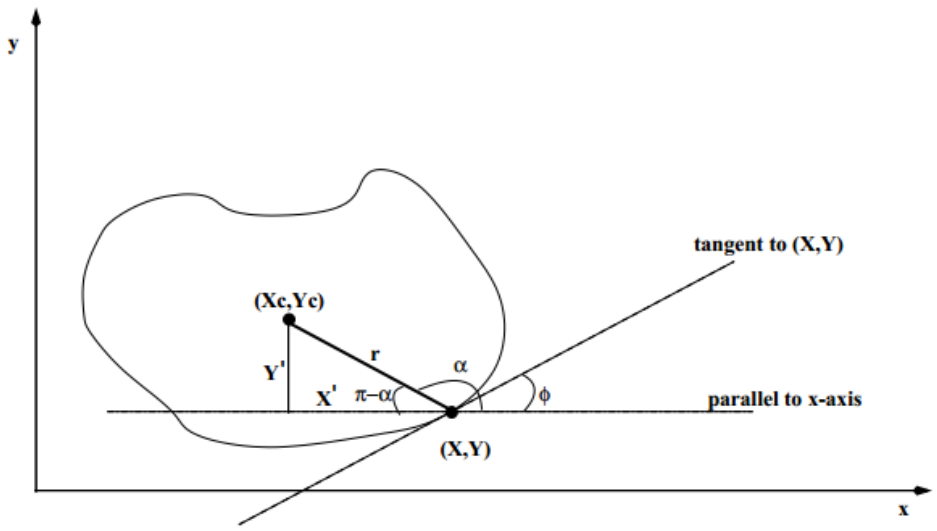



Figure 2.11: GHT technique

equation from the Figure 2.11,

$$\begin{aligned} x &= x_c + x' \\ y &= y_c + y' \\ cos(\pi - \alpha) &= \frac{x'}{r} \\ sin(\pi - \alpha) &= \frac{y'}{r} \end{aligned} \tag{2.8}$$

By considering the magnitude values of Equation 2.8, we can formulate Equation 2.9,

$$\begin{aligned} x_c &= x + cos(\alpha) \\ y_c &= x + sin(\alpha) \end{aligned} \tag{2.9}$$

GHT involves following processing steps,

1 A reference point $(X_c, Y_c)$ is taken arbitrary.

2 A line is drawn from a boundary (edge point) to the point $(X_c, Y_c)$.

3 Angle $\phi$ (perpendicular to the gradient's direction) is calculated.

4 The reference point $(X_c, Y_c)$ is stored as a function of $\phi$ in a special table named *R*-table shown in the Table 2.2 .

| | | | |
|---|---|---|---|
| $\phi_1$ : | $(r_1{}^1, \alpha_1{}^1)$, | $(r_2{}^1, \alpha_2{}^1)$, | ..... |
| $\phi_2$ : | $(r_1{}^2, \alpha_1{}^2)$, | $(r_2{}^2, \alpha_2{}^2)$, | ..... |
| ... | ..... | ..... | ..... |
| $\phi_n$ : | $(r_1{}^n, \alpha_1{}^n)$, | $(r_2{}^n, \alpha_2{}^n)$, | ..... |

Table 2.2: R-table for the reference point.

The *R*–table allows to use the contour edge points and gradient angle to re-compute the location of the reference point. Separate *R*–table is used for separate object. We need to quantize the reference point space as $P[x_{c_{min}}...x_{c_{max}}][y_{c_{min}}...y_{c_{max}}]$. For each edge point, $(r, \alpha)$ can be retrieved for each reference point indexed under angle $\phi$. For each $(r, \alpha)$ value, reference point can be calculated and one vote can be assigned for that reference point. The contour of the object is defined for maximum value of $P[x_c][y_c]$ reference point.

# Chapter 3

# Implementation of Edge Detection and Shape Detection Techniques

## 3.1 Introduction

In this chapter we will describe all the steps and their sequence involved in developing the software. A fully functional solution will make it possible to do independent testing of each baked product and thus assuring each product is according to specifications. Immediate feedback is also made possible through the software thus real time adjustment of the parameters of the baking process can be done easily. The goal is to develop a solution which is generic enough to be deployed in any industrial environment that can provide a bitmap image of the backed product.

## 3.2 Technology

By using integrated hardware and software components, porting of process control to other industrial environments will become easier. It will also speed up system configuration and installation, and decrease cost.

### 3.2.1 Hardware used

Figure 3.1 below shows the conceptual architecture of the proposed 'Bakery Vision Analyzer' solution. The software either will run on an industrial computer or on a computer housed in an industrial panel. The system is enclosed in hut–shaped container as shown in the Figure 3.1, the color cameras are used for visual inspection whereas infrared thermal camera is used for measuring product temperature. A laser line fixed at an angle of 45° approximately from the axis of the camera for product height calculation. Since the system is enclosed in a hut (as shown in Figure 3.1); white LEDs are used for product illumination to avoid external light interference. The cameras will be connected to the computer either by a high-speed USB 2.0/3.0 link, for shorter distances, or through an Ethernet link, possibly via a network switch or router, at longer distances that are not supported by the USB standard.

### 3.2.2 Software Tools

The 'Bakery Vision Analyzer' software is required to run on the computer that controls the operation of the cameras, the laser height metering, LED lighting,

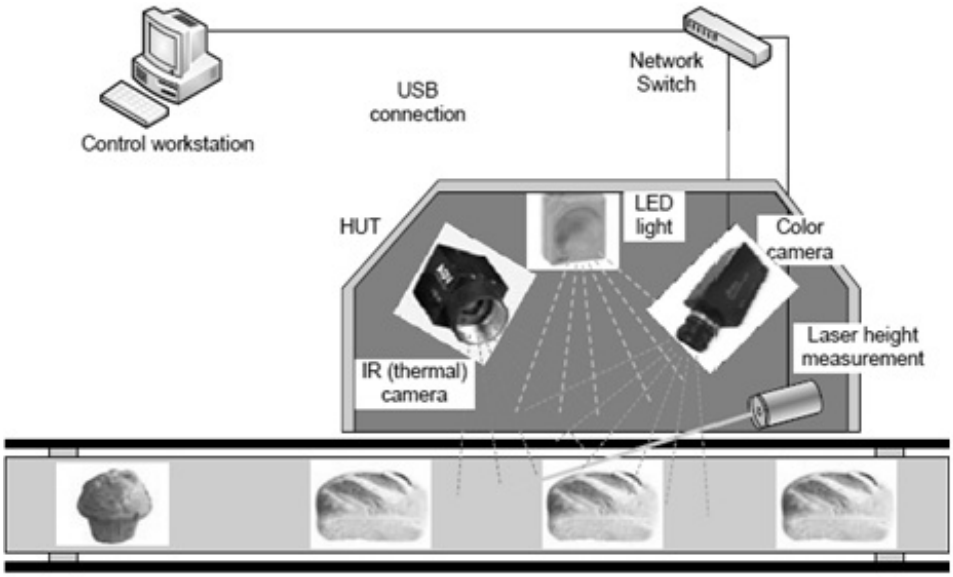



Figure 3.1: Conceptual architecture of Bakery Vision Analyzer.

and other necessary peripherals. The goal is to achieve high modularity, so that different items produced may be analyzed using the correct processing options and with little customization (or choosing appropriate option from the interface). The front end of the 'Bakery Vision Analyzer' software is implemented in C# language in a .NET environment (using Visual Studio 2010). The domain logic is implemented using AForge.NET [14] library classes and writing our own logic. In the back end, data storage and reporting has been done in Microsoft SQL Server 2008R2. The reason of choosing C# (VS 2010) over other programming language is: it provides many well defined and supporting DLLs for image analysis and processing. Moreover, it provides enriched set of controls, which help developers to build highly customizable and user-friendly interfaces. Among other advantages of using .NET includes: inter–operability among multiple systems (i.e. legacy

and newly developed system) and it is easy to maintain and deploy application features. Such features allow creating personalized and controlled components which help speeding up application development. Furthermore, keeping in mind the restriction of modularity, we have separated the data storage and reporting from the business logic. Initially we used SQL server (2008 R2) (ADO.NET) as back-end data storage and reporting tool as it is fast, easy to manage and further query the reporting table. In addition, VS 2010 provides support for many other back-end data storage tools such as MS Access, MySQL, and DB2 etc. These tools can easily be replaced by just adding a few lines of code.

## 3.3 Software Application

The application consists of three parts:

1. Canny edge detection
2. Shape detection
3. Calculation of the attribute values by analyzing object shape.

Figure 3.2 shows the high level diagram of the developed application.

### 3.3.1 Canny Edge Detection Method

By applying edge detection technique the data volume in the image can significantly be reduced without affecting the structural properties which will be later

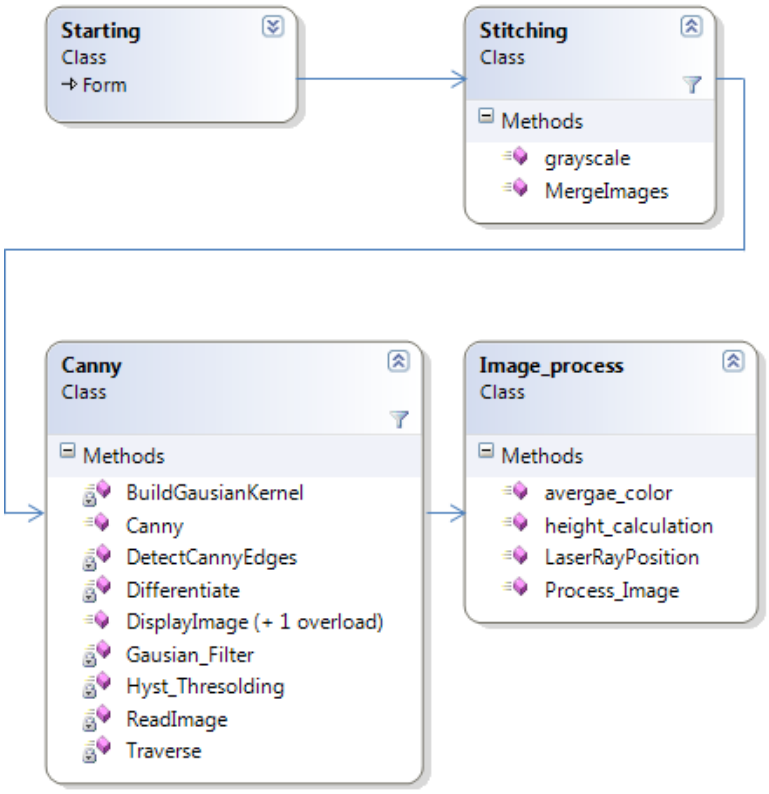



Figure 3.2: High–level structure of the developed application

used for image processing. There are several procedures available for edge detection as discussed in chapter 2.

We have considered Canny edge detection technique with regards to following criteria:

- Detection: The probability of detecting real edge points should be maximized while the probability of falsely detecting non-edge points should be minimized. This corresponds to maximizing the signal-to-noise ratio.

- Localization: The detected edges should be very close to real edges. There

will be minimum gap between real edge and detected Edge.

- Number of responses: One real edge should not result in more than one detected edge.

We have chosen Canny edge detector which is optimal for step edges [6]. Canny edge detection algorithm runs mainly in five sequential steps:

1. Smoothing: By applying Gaussian filter, smoothing of and image is done to reduce noise.

2. Finding gradients: The edges have been marked where the gradients of the image is having large magnitudes.

3. Non-maximum suppression: Only local maxima have been marked as edges.

4. Double thresholding: Potential and actual edges are determined by thresholding.

5. Edge tracking by hysteresis: Edges that are not connected with strong edges have been suppressed.

**Smoothing**

Images taken from a camera will contain some amount of noise. As noise can mislead the result in finding edges, we have to reduce the noise. Therefore the image is first smoothed [16] by applying a Gaussian filter. The kernel of a $5x5$

Gaussian filter with a standard deviation of $\sigma$= 1.4 is shown in Equation 3.1.

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{3.1}$$

We have implemented Gaussian filtering with specific kernel size (N) and Gaussian envelope parameter $\sigma$. The Gaussian filter mask of $NxN$ size is generated by invoking following function.

*private void GenerateGaussianKernel(int N, float S, out int Weight)*

The function 'GenerateGaussianKernel' takes kernel size ( parameter $N$) and the envelope parameter $\sigma$ (parameter $S$) as input to generate the $NxN$ kernel values.

Following subroutine removes noise by Gaussian filtering.

*private int[,] GaussianFilter(int[,] Data)*

The function 'GaussianFilter' multiplies each pixel in the image by the kernel generated. It returns the smoothed image in a two dimensional array.

**Finding Gradients**

The principle of Canny algorithm is based on finding edges where the intensity of the grey image changes to maximum value. These points are marked by determining the gradient of the image. Gradient for each pixel is calculated by applying Sobel operator. For each pixel, partial gradient towards $x$ and $y$ direction is determined respectively by applying the kernels mentioned in Equation 2.2.

By applying Pythagoras law shown in Equation 2.2, we have calculated the magnitude of gradient (strength of the edge) for each point. The directions of the edges have been calculated as shown in Equation 2.4.

The edges have been marked where the gradients of the image have large magnitudes. Sobel $G_x$ and $G_y$ masks are used to generate partial derivatives $\partial x$ and $\partial y$ (partial spatial derivatives) of the image. Spatial gradient value ($df(x, y)$) is obtained from this partial derivatives. The following function implements differentiation using Sobel filter mask.

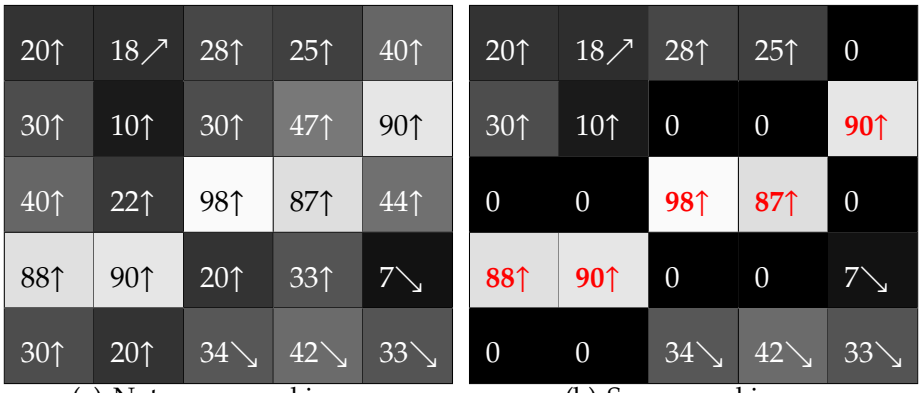
*private float[,] Differentiate(int[,] Data, int[,] Filter)*

The function takes the 'Gray' image as input. Color value of 'Gray' image is stored in a two dimensional array. The image array is smoothed first. Later, a derivative of the image is performed by calling the function 'Differentiate'. This function returns gradient image as a two dimensional array.

**Non-Maximum Suppression**

This is necessary to convert the blurred edges in the image of the gradient magnitudes to sharp edges. Actually this is performed by considering only all local maxima in the gradient image and deleting everything rest. The algorithm is applied for each pixel in the gradient image:

- Gradient direction is rounded to the nearest 45°. Thus, it can choose one of 8-connected neighbourhood.
- The edge strength of the current pixel is compared with the edge strength of the pixel in the positive and negative gradient direction. I.e. if the gradient




(a) Not suppressed image (b) Suppressed image

Figure 3.3: Example of non-maximum suppression. The edge strengths are indicated both as colors and numbers, while the gradient directions are shown as arrows. The resulting edge pixel values are marked with red colors.

direction is north ($\theta$= 90°), compare with the pixels to the north and south.

- If the edge strength of the current pixel is largest; preserve the value of the edge strength. Otherwise, suppress (i.e. remove) the value.

An example of non-maximum suppression is shown in Figure 3.3. As per the Figure, almost all pixels have gradient directions pointing towards north. That is why they are compared with the pixels above and below. The pixels with maximum values in this comparison are marked with white borders. All other pixels have been suppressed. Nom-maximum suppression technique is applied on Figure **??** to get the Figure **??**. Intensity of all non-maximum pixels are forced to value zero. For this reason, these non-maximum pixels are changed to black ( zero intensity) in color after applying the method.

Finally, only local maxima have been marked as edges. Thus, we can find gradient direction and using these directions we are able to perform non maxima

suppression.

**Double Thresholding**

After the non-maximum suppression step, the edge pixels are still marked with their strength pixel-by-pixel. Many of these may be true edges of the image, but some might be caused by noise or color variations for instance due to the rough surface. In order to get rid of such unpleasant situation, we can apply thresholding so that only edges stronger than a certain value would be preserved. In our implementation we have used double thresholding. Edge pixels which are stronger than the high threshold are marked as strong; edge pixels which are weaker than the low threshold are suppressed and edge pixels between the two thresholds are marked as weak. Potential edges are determined by thresholding. We have kept the provision to tune this high threshold and low threshold parameter value if required.

**Edge Tracking by Hysteresis**

Strong edges are referred as certain edges, and can immediately be included in the final edge image. Weak edges are considered if and only if they are connected to strong edges. The logic is of course that noise and other small variations are unlikely to result in a strong edge (with proper adjustment of the threshold levels). Thus strong edges will (almost) only be due to true edges in the original image. The weak edges can either be true edges or due to noise/color variations. The latter type will probably be distributed independently of edges on the entire image, and

thus only a small amount will be located adjacent to strong edges. Weak edges due to true edges are much more likely to be connected directly to strong edges.

Final edges are determined by omitting all edges that are not connected to a very certain (strong) edge. This is implemented by a recursive function which performs double thresholding by two thresholds 'High Threshold' ($T_h$) and 'Low Threshold' ($T_L$) and 8-connectivity analysis. The following function is called to perform this.

*private void HysteresisThresholding(int[,] Edges)*

The function 'HysteresisThresholding' defines pixels having color value higher then $T_h$ as strong edge . Pixels having color value with in the $T_h$and $T_L$ are known as weak edge. If both ends of a weak edge are connected to strong edges the function 'HysteresisThresholding' modifies the weak edge to a strong edge. Color value lower than $T_L$ is forced to set value zero.

### 3.3.2 Shape Detection

The shape detection algorithm performs checking/detection of some simple geometrical shapes for provided set of points (edge points of a blob). During the check the algorithms goes through the list of all provided points and checks how accurately they fit into assumed shape.

During image processing, especially for blob analysis, it is often required to check some objects' shape and depending on it might perform further processing for a particular object shape. For example, some applications may require finding only circles from all the detected objects, or quadrilaterals, rectangles, etc. Aforge.net [14] describes some basic techniques, which allows detecting such

simpler shapes like circles, triangles and quadrilaterals (plus their subtypes, like rectangle, rhombus, triangle, etc.).

The shape checking algorithm allows some deviation of points from the shape with given parameters. In a nutshell, it is permitted that specified set of points may form a little bit distorted shape, which might still be recognized. The allowed range of distortion can be controlled by two properties (Min–Acceptable–Distortion and Relative–Distortion–Limit), which allow higher distortion level for bigger shapes and smaller amount of distortion for smaller shapes. In order to check specified set of points, the algorithm calculates mean distance between specified set of points and edge of the given shape. If the mean distance is equal to or less than maximum permitted distance, then a shape is recognized. For simple shape detection, we have taken the assistance from AForge.net Framework. We have used simple shape Class under the framework during implementing the application. Simple shape class finds the blob points of the objects and preserves these points within an array named blob–array.

During implementation we have invoked separate methods for rectangular/square and circular objects.

**Rectangular or Square Shaped Objects**

The blob points are used to find four corner points for Rectangular or Square shaped object. The centre point of the object is calculated from these corner points.

As the expected dimension of the object is known and fixed for a batch, there is provision to provide expected length and width of the object as input values.

We have kept provision that calculated value (actual) may deteriorate +/-10% with respect to given input parameter values. That means if the actual object deteriorates maximum 10% with respect to given dimensions, the algorithm will be able to report the object as valid within the acceptable limit. The deterioration percentage can be tuned to more or less if anybody wants.

**Circular Shaped Objects**

The idea of circle detection algorithm is based on the assumption, that all circle's edge points have the same distance to its center, which equals to circle's radius. It may happen that circles may have some distortion of their shape, so some edge pixels may be closer or further to circle's center. The blob points have been considered to find the radius and centre for the circular object. The algorithm can easily trace a circular object. As mentioned, we can leverage the flexibility by varying the distortion percentage value parameter within a certain range. If the distortion is too big, than most probably the object has other than circle shape.

Finding circular object is easier compare to square or rectangular object due to homogeneity of circular objects. The algorithm is able to process doughnut (one type of baked product) image which is having common centre but different radius ( outer and inner circle). Thus it is able to calculate the thickness.

**Calculation of Attribute Values for Objects' Shape**

We have kept the provision of stitching of two images if required. During image processing, specially in case of batch of images, snapshots are taken repeatedly. For

a particular case, image of an object could be split with in two snaps. Stitching is simple but might be very effective method. It was an optional requirement for the development and we have implemented successfully. For stitching we have performed following steps.

- Get the width and height of all the images and sum them up and store in separate variables of type int.
- Draw a new bitmap with length and width equal to sum up values.
- Convert all the images to be stitched into bitmap or store them in a bitmap object.
- Using the nested loops get all the $x$ and and $y$ pixels and set the corresponding pixels in the newly drawn bitmap (stitched image).

One of the key requirement of this implementation is to find the dimension and relative position of the objects. These values are very crucial for rectagular/square shaped objects. We have calculated following physical attributes.

- Dimension calculation:(Only for Rectangular or Square shaped object)
  - Length and width are calculated by applying Simple shape class. First, a blob object consisting of points is identified from the image. The blob points of an object is preserved in a two dimensional array. From this array of blob points, four corners points are calculated. Then Edge length is calculated by finding the distance of two points. The higher length edges pair is considered as length of the object and the remaining pair is

considered the width of the object. We have marked corner points from 1 to 4 clockwise in an order to avoid the confusion between edges and diagonals.

  - Height of the object is calculated by inspecting the deflection of laser ray by an object in the image. Figure 3.4 explains height calculation of an item. The algorithm calculates the laser line. When the laser ray passes over an object it is deflected. The maximum value of perpendicular distances between the deflected line and the original line is used to calculate the height of the bread. In final step, we calculate the actual height by applying the angle of laser ray projection ($\tan\theta$) on the body of an object.

- Corner points: (Only for Rectangular or Square Shaped Object) As mentioned earlier, we can easily find the corner points from the blob array of of points for a particular object. In order to find corner points, we simply determine the intersection points for blob edges. Corners of the object have been marked with 1, 2, 3 & 4 in clockwise.

- Centre of the Object: The application is able to find the centre of the object from the blob array of points. This is accomplished by performing the simple average of all points lying within the border edges of the object. The centre is marked by a small and 'blue' color circle.

- Borders of the object: Borders of an object is marked with 'red' color. In case of rectangle or square, we have drawn lines simply joining the corner points

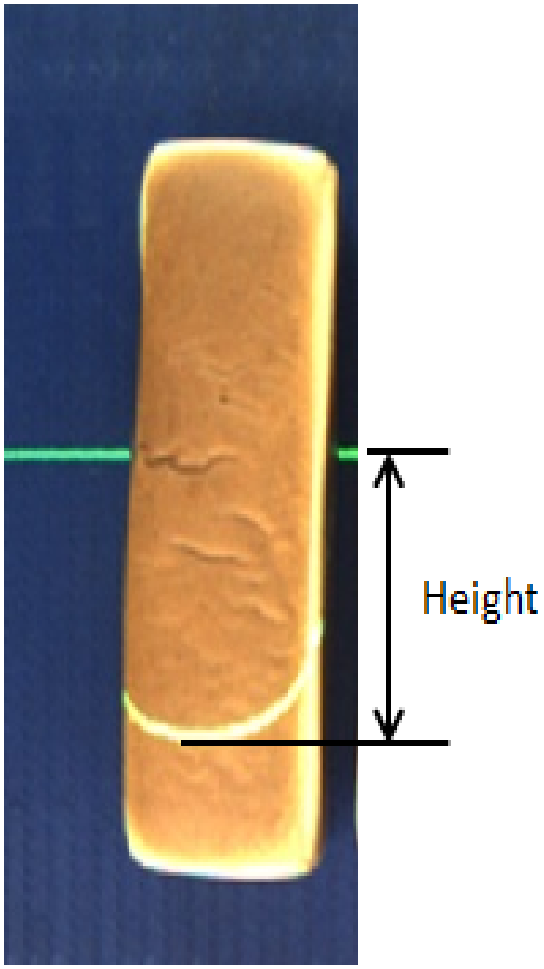



Figure 3.4: Height measurement of an object

in clockwise. For marking a circle, we have drawn a circle centred in the objects centre with a radius calculated from the blob points of the object.

- Angle of the object position: (Only for Rectangular or Square Shaped Object) Angle of the object position is calculated from the slope of the width edges of the object. The angle ranges from -90 degree to +90 degree. If the object tilts towards right direction then the angle is negative otherwise positive.

- Average color of the object: We have kept the provision to calculate the average color of the object to find if the object is over burned or not. For calculation, we have considered a square area whose centre and the objects centre is same. The length of the the square is less then the width ( in case of Rectangular) or radius (for Circular) of the object. This ensures that the sample area will never be outside of the object and will provide the color information of the central part of the object. We have simply summed up the pixels color from the original image before applying the Canny algorithm and then divided with the number of the pixels. Nevertheless, there is flexibility to change the area size if there is a need.

- Physical damage of the object: After applying Canny edge detection algorithm, a black pixel indicates that there is no change in pixel color with respect to neighbour pixels where white pixel indicates that there is change in color. For a physically damaged object, the pixel color in the damaged part differs abruptly. We have calculated the average color ( gradient image after applying Canny edge detection algorithm) of inside of the object. Pixels near the Edge border are usually white ( as color changes in this region). These pixels may lead higher average color value and may trigger false alarm regarding the damage of the object. To get rid of such unpleasant result we have considered 90% of the object's inside area leaving the border pixels out of the computation. Again, this parameter value can be tuned as per requirement. Average intensity of each object ( after applying Canny edge detection method) is calculated. If the average intensity is above a threshold

value we report as physically damaged object. In our experiment we have set the threshold value as 85. If the noise value is more than 85 the application reports the object as physically damaged; otherwise the object is undamaged. Of course, this threshold value can be changed if required.

- Object count: By using simple shape class we can detect any object having at least five pixel length, width or radius. Then after, we filter these blobs to find objects satisfying given requirements. Finally, we report total number of objects satisfying input criteria.

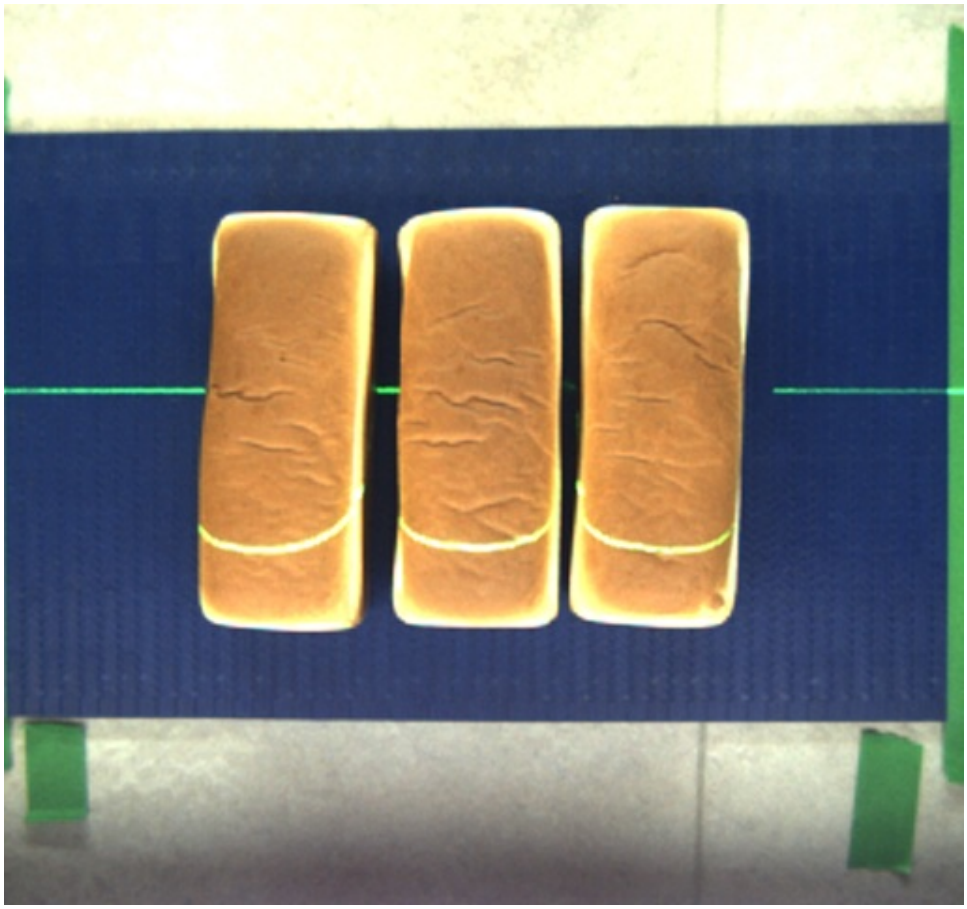
## 3.4 Observations

In this section we examine different physical properties of objects. All angles are measured with respect to the direction of motion which correspond to vertical axis in Figure 3.1. If an objet tilts towards the right of the vertical axis then the angle value will be negative, otherwise the value will be positive.

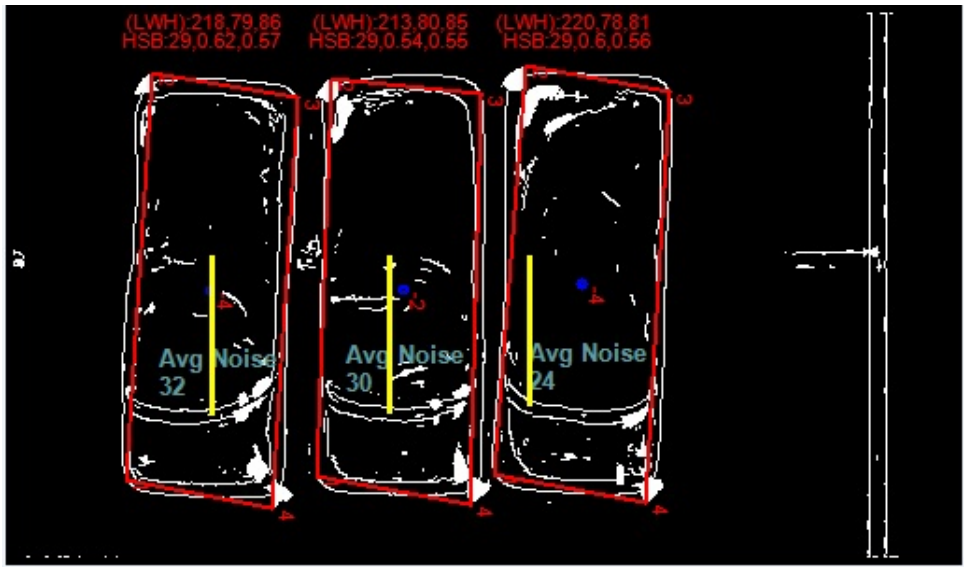
### 3.4.1 Case 1

In this subsection we discuss Figure 3.5. This is an image of three objects. All the objects have been detected. Neither damaged nor undetected objects has been found.

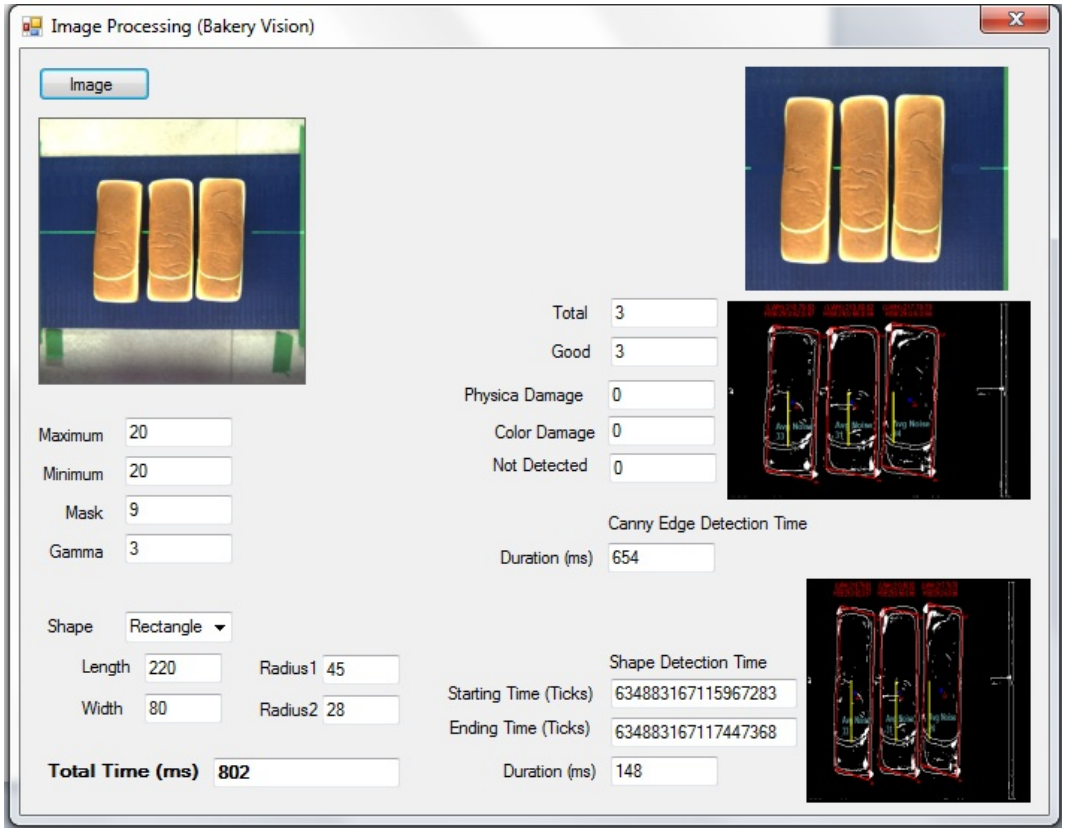
- In the upper part of the image, dimensions of all objects are shown. Dimensions of the three objects in Figure 3.5 are similar. Height for each of the objects is marked with 'Yellow' color line.



(a) Original Image




(b) Processed Image




(c) Application Output

Figure 3.5: Case 1 Result Summary

- Centre of the objects are marked with 'Blue' circle. The circle is centered at the center of gravity of the object [17] and the radius length is five pixels.
- Corner Points are labeled from 1 to 4 clockwise. The labeling sequence is important to differentiate between edges and diagonals of rectangular shaped objects.
- Angular positions of the objects are -4°, -2° and -4°. These values indicate that the objects are slightly tilted towards the right.

- Average noise is calculated by averaging color values of surface pixels of an object (excluding the edge pixels) in the gradient image. Average noise values for three objects are 22, 18 and 18. Each of the values is below the threshold value of 85. This concludes that all the object are physically good (not damaged)shaped.

- Hue, Saturation and Brightness (HSB) values are shown on top of each object in the image. For an over burnt object, the brightness (intensity) value is lower compared to that of a properly baked object. Average brightness color of the objects are 0.57, 0.55 and 0.56 calculated in 1 unit scale. The threshold value for brightness is set as 0.4 for this observation. Objects having brightness value below the threshold value are considered over burnt.

### 3.4.2 Case 2

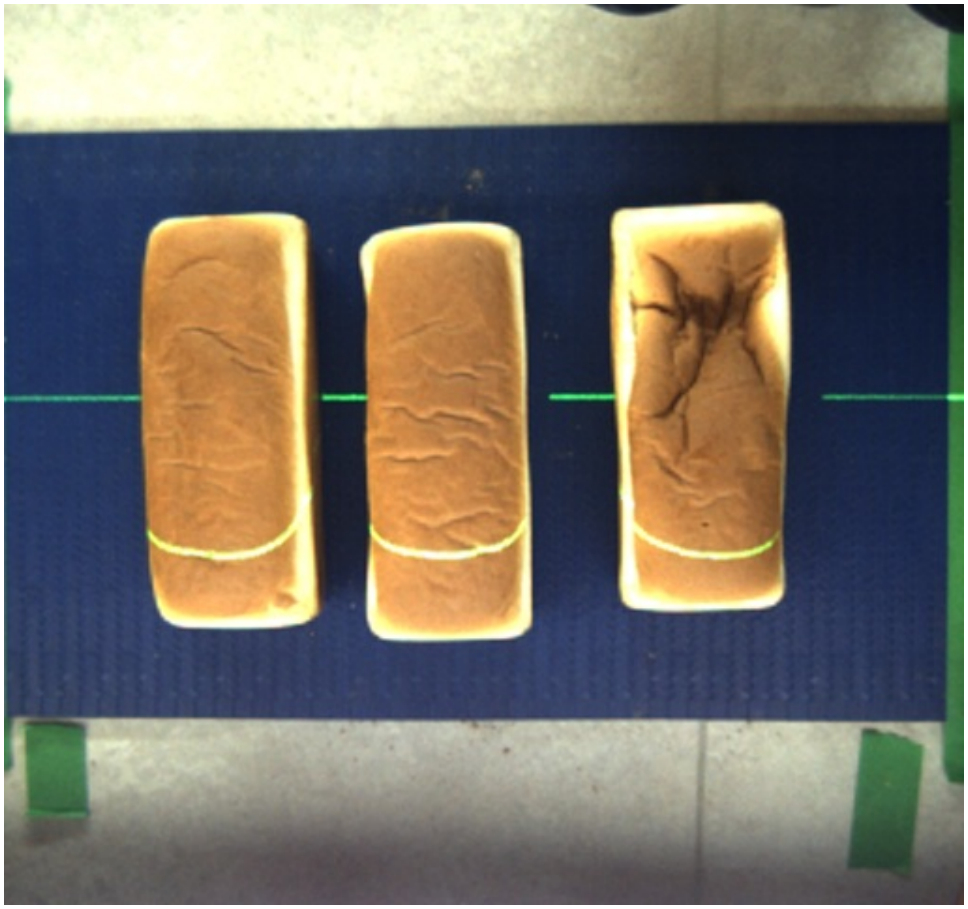
In this subsection we discuss Figure 3.6. This is an image of three objects as well. The application has detected all the three objects. With in the three objects, one object is reported as physically bad (damaged).

- Dimensions of all the objects are shown on the top respective object . Except for height of an object, the dimensions are similar.

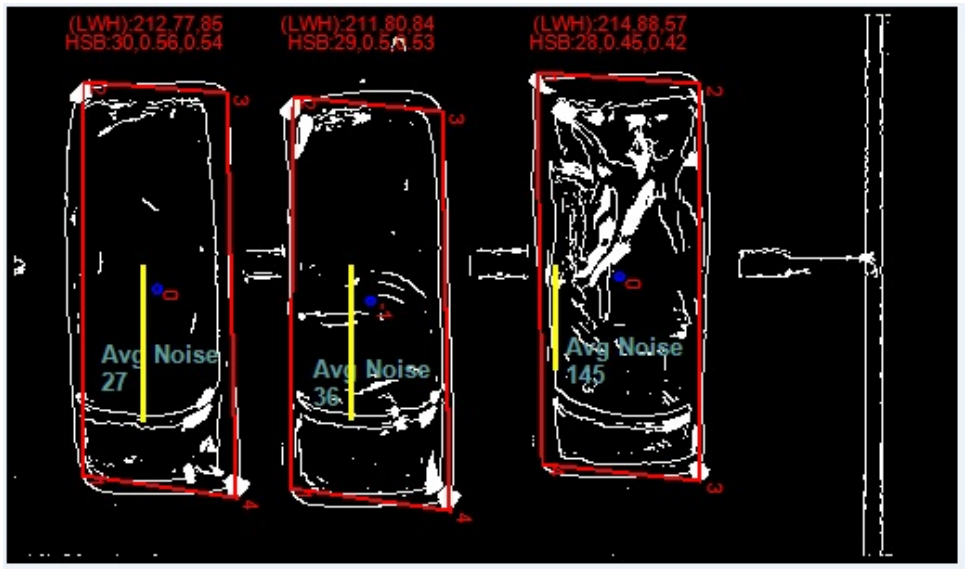
- 'Blue' marked centers are visible for all the objects.

- All the corner points are labeled properly for each object.

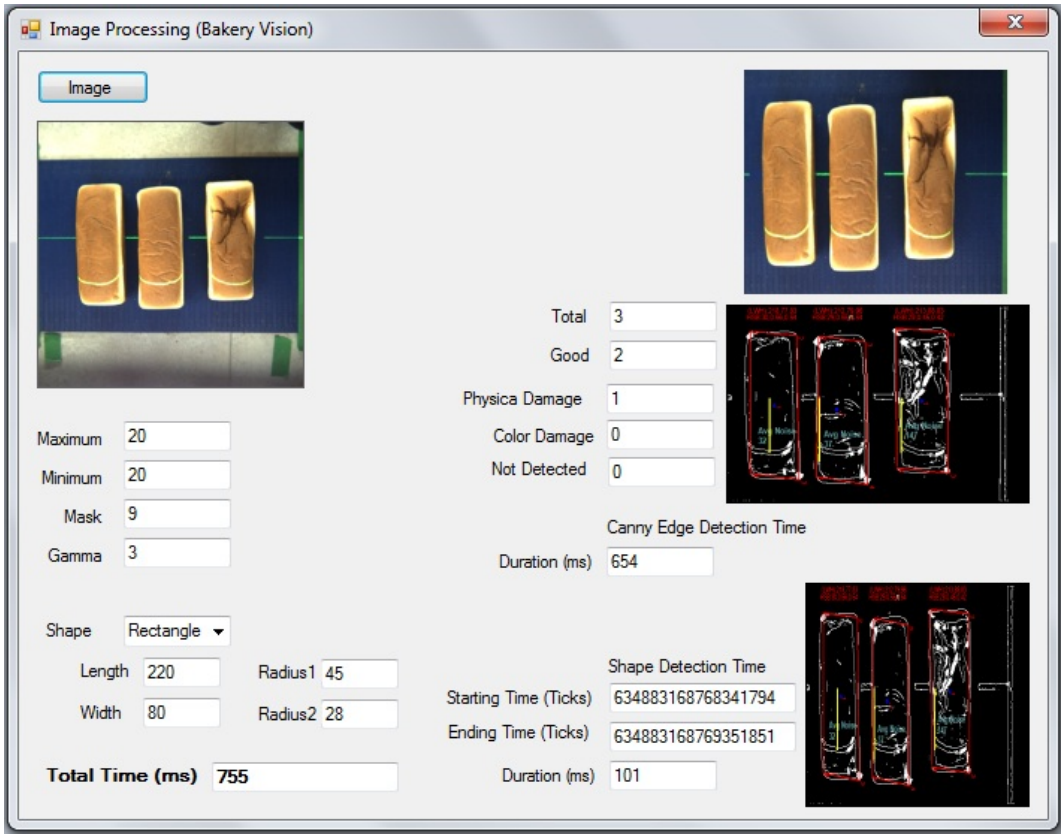
- Angular positions of the objects are 0°, +1° and 0°. The objects are almost positioned towards the direction of motion of the belt.



(a) Original Image




(b) Processed Image




(c) Application Output

Figure 3.6: Case 2 Result Summary

- Average Noise values are calculated as 27, 36 and 145. One of the objects is having noise value above the threshold value of 85. The application reports one physically damaged (bad) object for this scenario.

- Hue, Saturation and Brightness (HSB) values are shown on top of each object in the image. Average brightness color of the objects are 0.54, 0.53 and 0.42 (Maximum 1 scale basis). Brightness values for all the three objects are above the threshold value of 0.40. Therefore, the application reports no over burnt

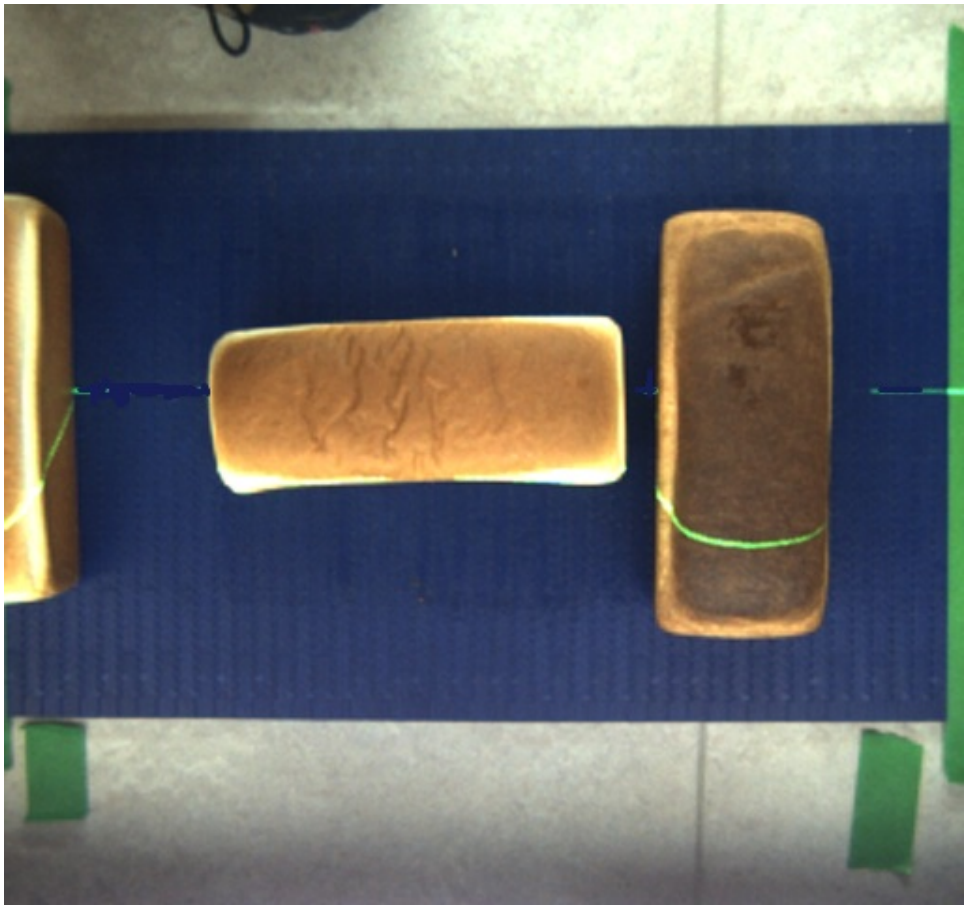
object for this observation.

### 3.4.3 Case 3

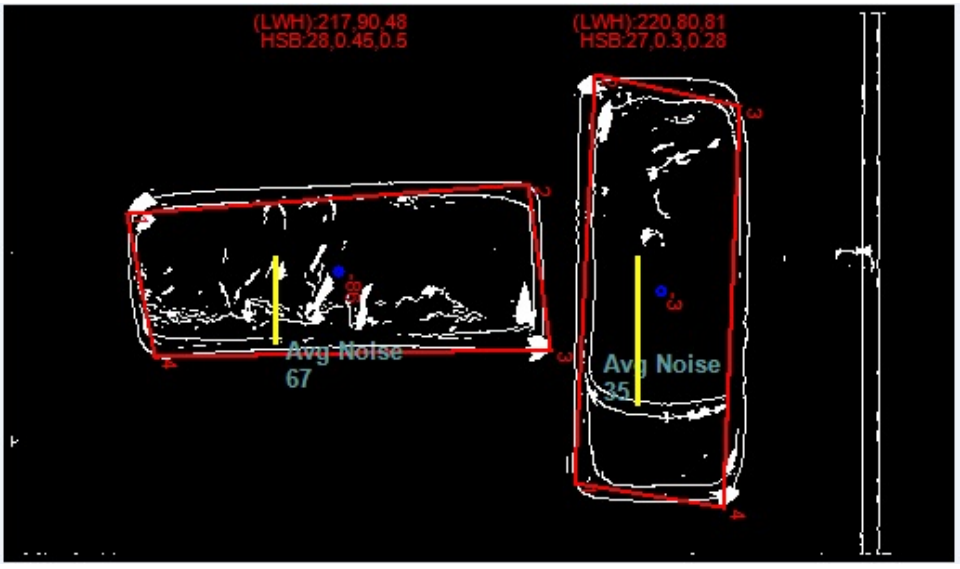
In this subsection we discuss Figure 3.7. There are in total two objects under this observation. All the objects are detected by the application. There is no physically damaged [29] object. But the application reports the presence of one burnt object.

- Average color values of all object are calculated and presented on the top of respective object. Hue values for both objects are the same. Saturation value for fist object is 0.45 and for second object the value is 0.3. Brightness value for first object is 0.50 and that for the second object is 0.28. Whether an object is considered over burnt or not depends on the brightness value: the lesser the value, the more the object is burnt. From the result it is obvious that second object is over burned (well below the threshold value of 0.40).

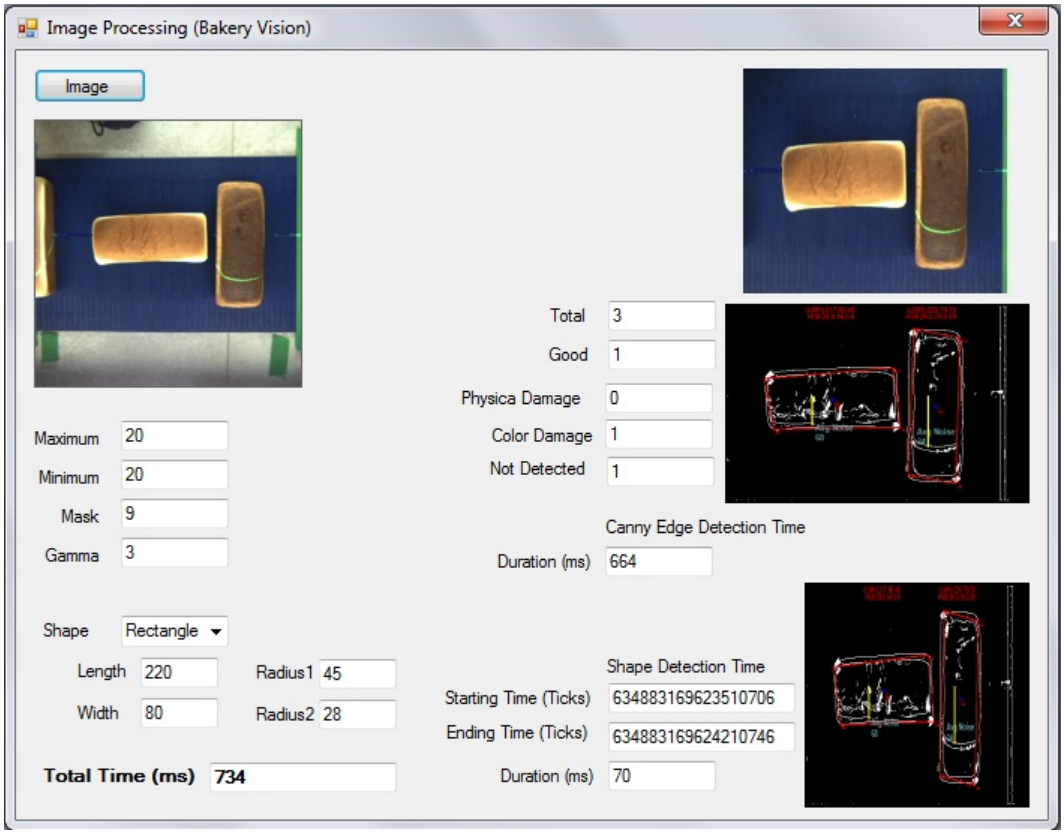
- On the top of image, dimensions of each object are shown. Length and width of the two objects are identical but their height differs significantly. This is due to the false height calculation of the first object. The laser ray can not be reflected on the upper surface of first object and blocked before the object side. The height shown for the first object is the distance calculated from the original laser ray ( before reflection) line to the remote edge of the object. And this is not true height. We have to be careful in positioning laser ray generating device. It should be positioned in such a way that laser ray is deflected by each object's upper surface.



(a) Original Image




(b) Processed Image




(c) Application Output

Figure 3.7: Case 3 Result Summary

- Centre of all objects are visible and marked with 'Blue' circle.
- Corner Points are marked from 1 to 4 clockwise.
- Angles of the objects are -86° and -3°. The objects are not oriented in the same direction.
- Average Noise [21] value is measured as 67 and 35. Both values are below the threshold value of 85.

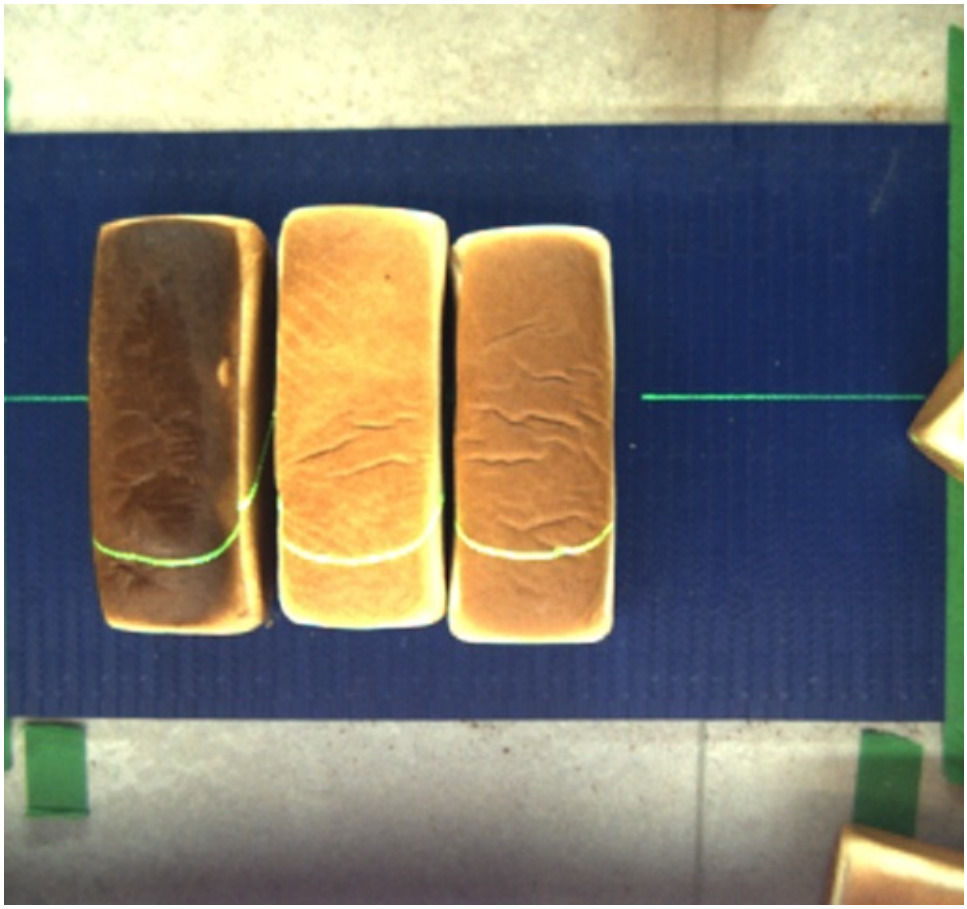
### 3.4.4 Case 4

In this subsection we discuss Figure 3.8. The Figure 3.8 is an image of three objects. The application detects two objects out of three objects for this observation.

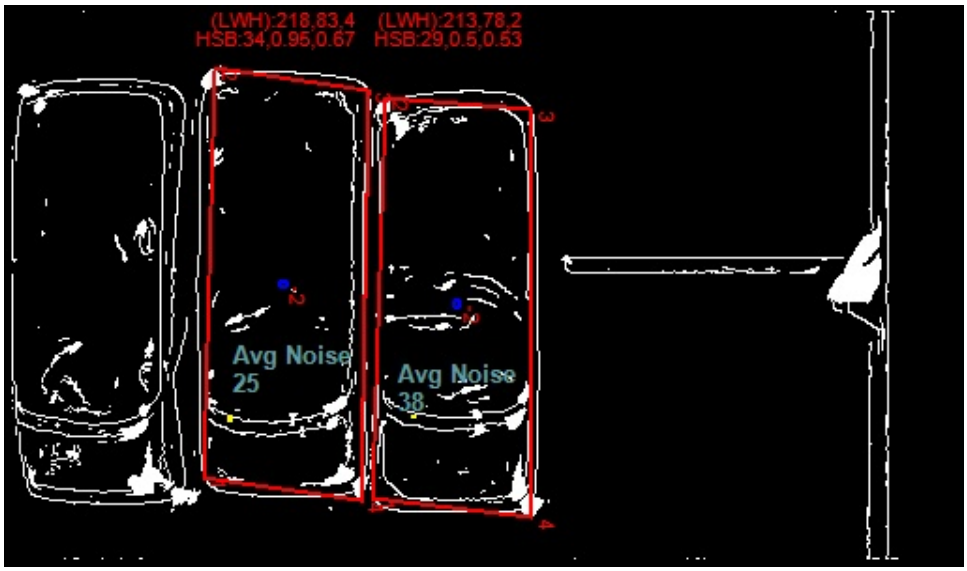
- The reasons behind the missing items can be one or many. It might be primarily due to excessive distortion for the missing object. Laser ray and vulnerable corners of the object always inject some additional noises and the increased noise value affects the distortion value for the object. This can result an object being undetected. The distortion value should not be set outside a reference window value. By setting distortion value outside a reference window value, improved result can be observed for few cases but it will have bad influence on the results for other cases.
- Remaining two objects have been detected and processed correctly. Both the objects are -2° tilted.
- Average Noise [21] values are measured as 25 and 38; both the values are well below the threshold value of 85.

### 3.4.5 Case 5

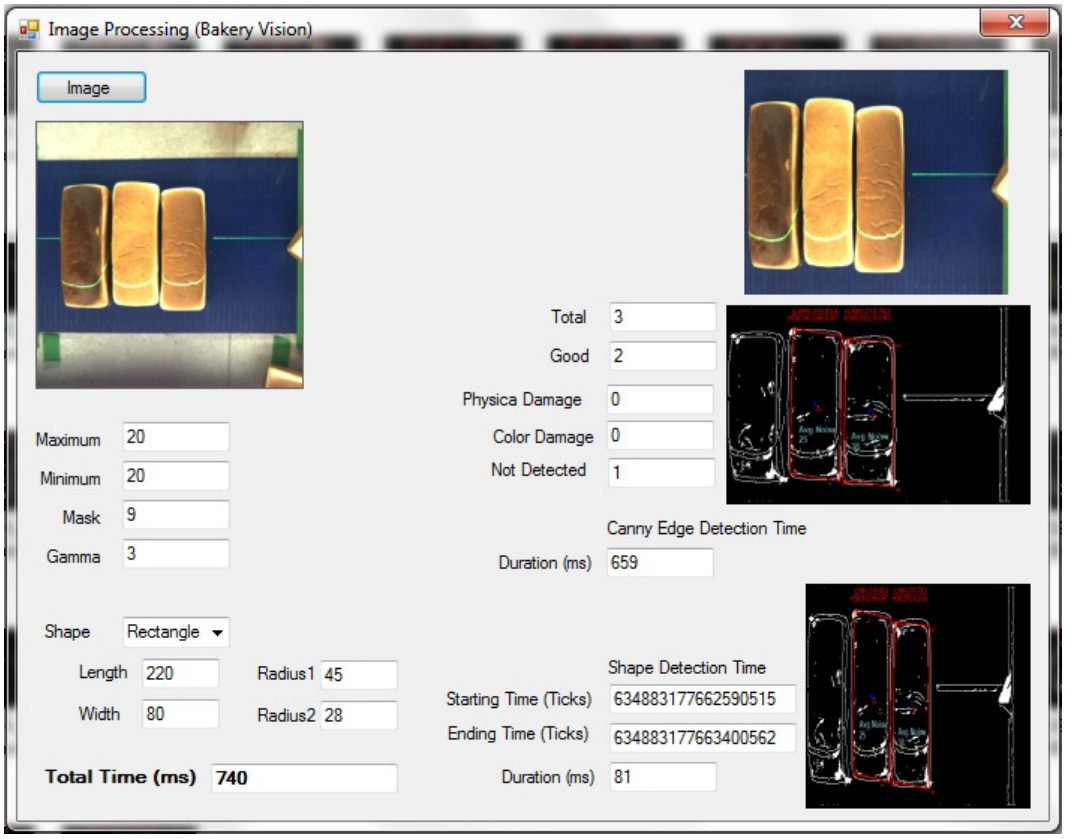
In this subsection we discuss Figure 3.9. In this observation, circular shaped objects are processed. Three circular shape objects have been detected. The input to the application was a image of three Muffins (object).



(a) Original Image




(b) Processed Image

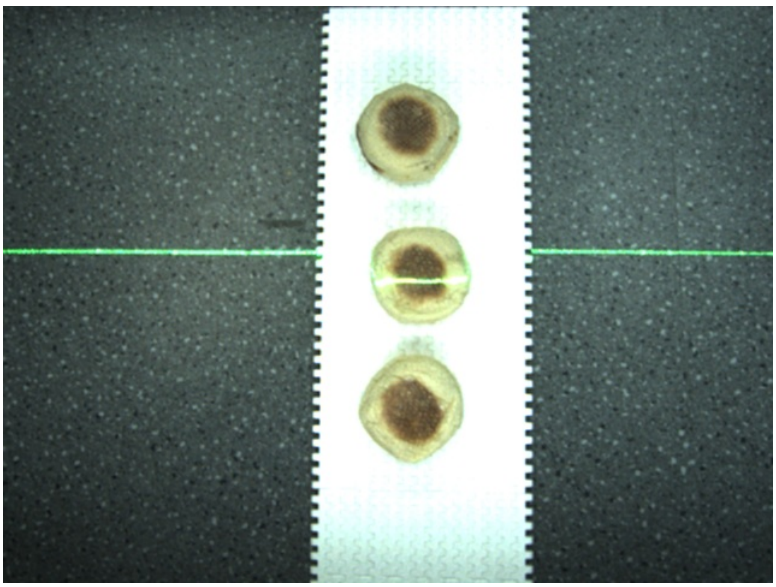



(c) Application Output

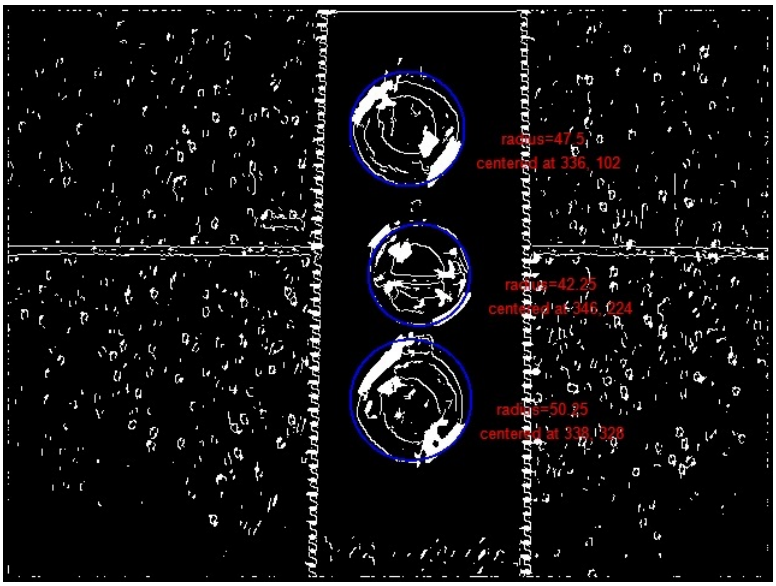
Figure 3.8: Case 4 Result Summary

- Radius of the objects are 47.5, 42.25 and 50.25 pixels length respectively. Center of the objects are shown as well.

- Though the boundary and centre of gravity is determined, physical properties of the shapes need to be processed.



(a) Original Image




(b) Processed Image

Figure 3.9: fig: Output Case 5

## 3.5 Summary

In this chapter, we have implemented optimized Canny edge detection method. Simpler shape detection method is applied on the detected Canny Edge image. This Canny based application provides more customization compared to plain Sobel (widely used in Bakery image processing) based application which is mostly used in the industry. This application can be synchronised with variable belt color, size, shape. In addition one can define his own acceptance range of distortion percentage level. The application shows robustness by changing outside environment (by turning on and off some led lights, changing the camera position etc.). The processing time is within the acceptance range (well below one second). Processed information can be stored in any of the database systems. In summary, this application provides flexibility and robustness in processing image of bakery system.

# Chapter 4

# Performance Evaluation of Edge & Shape Detection Techniques

## 4.1 Introduction

In this chapter, we show the comparative analysis of various image edge detection methods with Canny edge detection method. First we will thoroughly investigate the required time to detect edges with in an object. We will observe if there is any dependency on number or size of objects within images considering that all the images use the same memory space. We will also try to figure out the nature of probability distribution by inspecting edge detection time durations on a wide range of images.

Shape detection algorithm takes these edges and border points as input and provides the structure, position and centre of each object. Our experiment is limited to detection of rectangular shaped objects. For shape detection technique,

we will observe if the objects shape are traceable within a given time. We will observe how the processing time behaves by analyzing results of processing times taken on a group of images. We shall investigate if the application is able to find all the objects. We will compare the success rate of finding an object with other techniques. The crucial and very important point is the accuracy of output. We will investigate the ”False negative” and ”False positive” percentages in detecting damaged objects. Similar experiments will be carried on other techniques with the same inputs. Finally we will compare this important property performance.

## 4.2 Motivation

Industry is having existing applications which are mostly based on Sobel edge detection mechanism. Moreover, users have to pay higher license fee. In image processing, there is always some space to improve the performance. The developed application should be compared with currently available application used by the industry. We need to find how good or bad is our application in finding the accuracy. We need to see is there any big change in processing time as well.

If the application shows better accuracy within acceptable processing time, then the application will have some upper-hand. Therefore, the application could be used for better performance with probably lower commercial price.

## 4.3 Image Graying Edge Detection Time

For Image processing, Graying is very important step. In this step we usually convert three bytes color pixel value into one byte Gray value. This ensures less processing time and effectiveness of the algorithm to find edges. In this section we will mainly observe processing time duration. Our image samples range in various number of objects. We will analyze the performance of the following categories.

- Two objects per image.
- Three objects per image.
- Four objects per image.

**Data Analyzing Methodology**

We have plotted processing time series for each batch of the images. Probability distribution function has been drawn as well. We have analyzed the data by using a software name 'Easyfit'. For each data set we have initiated following tests.

- *Kolmogorov-Smirnov test:* Kolmogorov-Smirnov [7] test (K–S test) is a non-parametric test for the equality of continuous, one-dimensional probability distributions that can be used to compare a sample with a reference probability distribution (one–sample K–S test). It could also be used to compare two sample sets. To apply the Kolmogorov–Smirnov test, we have to calculate the cumulative frequency distribution (normalized by the sample size) of the observations as a function of class. Then cumulative frequency for a true distribution (most commonly, the normal distribution) is calculated. We need

to find the greatest discrepancy between the observed and expected cumulative frequencies, which is called the 'D-statistic'. Finally this is compared against the critical D-statistic for that sample size. If the calculated D-statistic is greater than the critical value, then we can reject the null hypothesis that the distribution is of the expected form. The empirical distribution function $F_n$ for $n$ independent and identically distributed observations $X_i$ is defined as,

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{X_i \leq x}$$

where $I_{X_i \leq x}$ is the indicator function, equals to 1 if $X_i \leq x$ and equals to zero otherwise. The Kolmogorov-Smirnov statistic for a given cumulative distribution function F(x) is,

$$D_n = sup_x |F_n(x) - F(x)|$$

- *Anderson-Darling test:* The Anderson-Darling test is an alternative to the chi-square and Kolmogorov–Smirnov goodness-of-fit tests. The Anderson-Darling test [27] is used to test if a sample of data came from a population with a specific distribution. It is a modification of the Kolmogorov–Smirnov (K–S) test and gives more weight to the tails than does the K–S test. The K–S test is distribution free in the sense that the critical values do not depend on the specific distribution being tested. The Anderson-Darling test makes use of the specific distribution in calculating critical values. This has the advantage of allowing a more sensitive test, but also the disadvantage that critical values must be calculated for each distribution. Tables for critical values are available for many of the distribution techniques.

- *Chi–Square test:* Chi-square [25] is a statistical test commonly used to compare observed data with data we would expect to obtain according to a specific hypothesis. For example, according to Mendel's laws, if we expect 10 of 20 offspring from a cross to be male and the actual observed number is 8 males, then one might require to know "goodness to fit" between the observed and expected. Then, we want to know whether the difference is due to chance (null hypothesis) or there is another cause for it. This is decided by comparing the value of $\chi^2$ statistic with the value from the table of critical values for the given sample size (i.e. the number of degrees of freedom). In this case, the $\chi^2$ is obtained as,

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

Where,$O_i$=Observed frequency for bin $i$. $k$ is total number of bins.

$$k = 1 + log_2 N$$

$N$ is the total number elements in the data set. $E_i$=Expected frequency for bin $i$.

$$E_i = F(x_2) - F(x_1),$$

$F$ is the CDF of the probability distribution being tested, and $x_1$, $x_2$ are the limits for bin $i$.

The value was tested against the values of a processing time table for objects. In all tests, graph for the probability distribution function was generated by using

a software tool named 'EasyFit' [18] which incorporates critical values required to reject or accept null hypothesis.

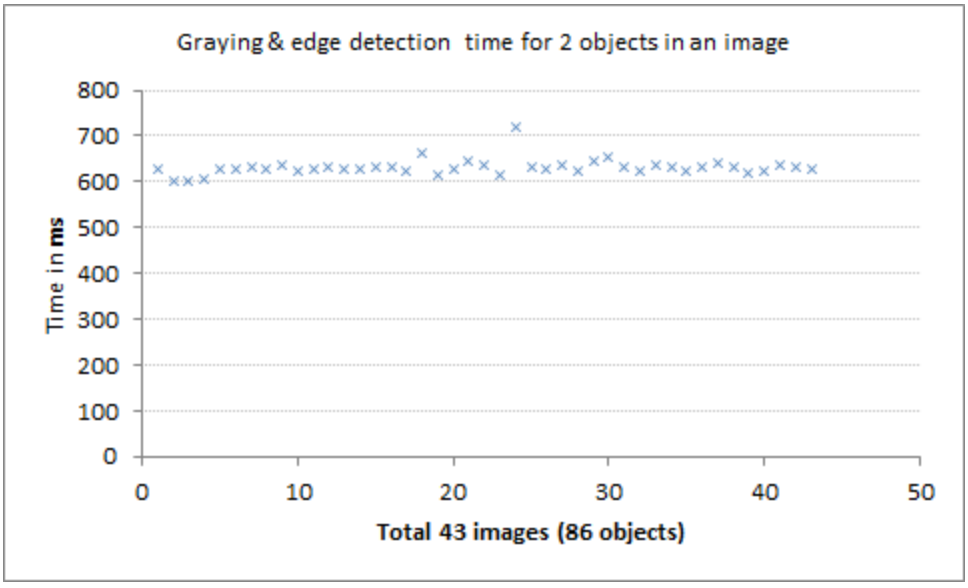
### 4.3.1 Two Objects per Image

In total 43 sample images have been processed. Edge detection time 4.1(b) varies from 601 milliseconds to 719 milliseconds. Mean value for the processing time $\mu$ is 631.51 milliseconds and standard deviation $\sigma$ is 17.67 milliseconds which is around 2.8%. If we look at the probability distribution function, longer tail is visible. That is why the distribution is closer to the 'Cauchy' [1] distribution than the 'Normal' distribution.

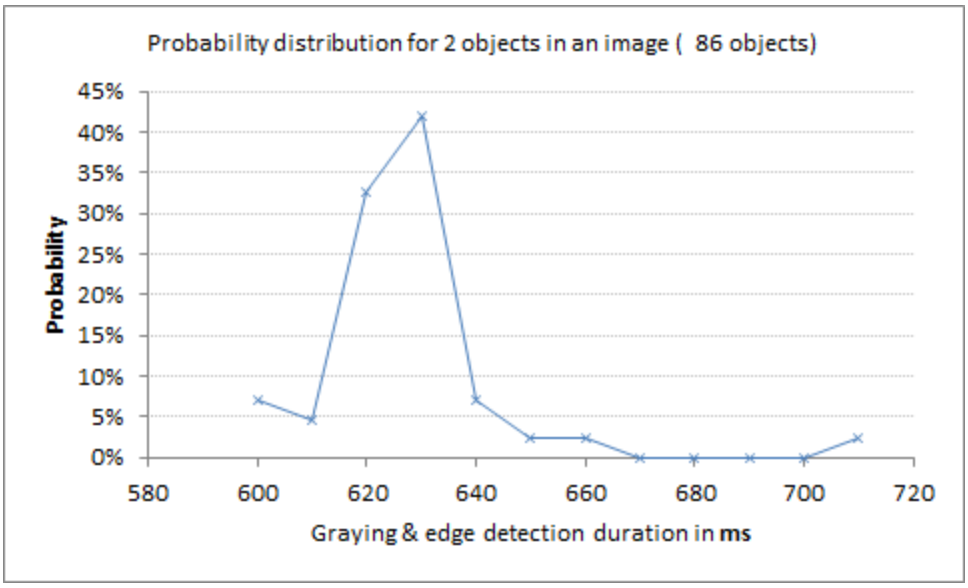
Shape detection time varies 4.2(b) from 50 milliseconds to 89 milliseconds. Mean value for the processing time $\mu$ is 66.09 milliseconds and standard deviation $\sigma$ is 8.56 milliseconds which is around 12.94%. The standard deviation has a higher percentage value than the Edge detection technique. That means shape detection time duration is spread out over a large range of values. If we look at the probability distribution function, a longer tail is visible for this case as well. And that is why the distribution is closer to 'Cauchy' distribution.

### 4.3.2 Three Objects per Image

For this scenario, we have processed 256 images with 768 objects in total. Edge detection time 4.3(b) ranges from 605 milliseconds to 798 milliseconds. Mean value for the processing time $\mu$ is 664.56 milliseconds and standard deviation $\sigma$ is 28.92 milliseconds which is around 4.35%. The distribution is more closer to the



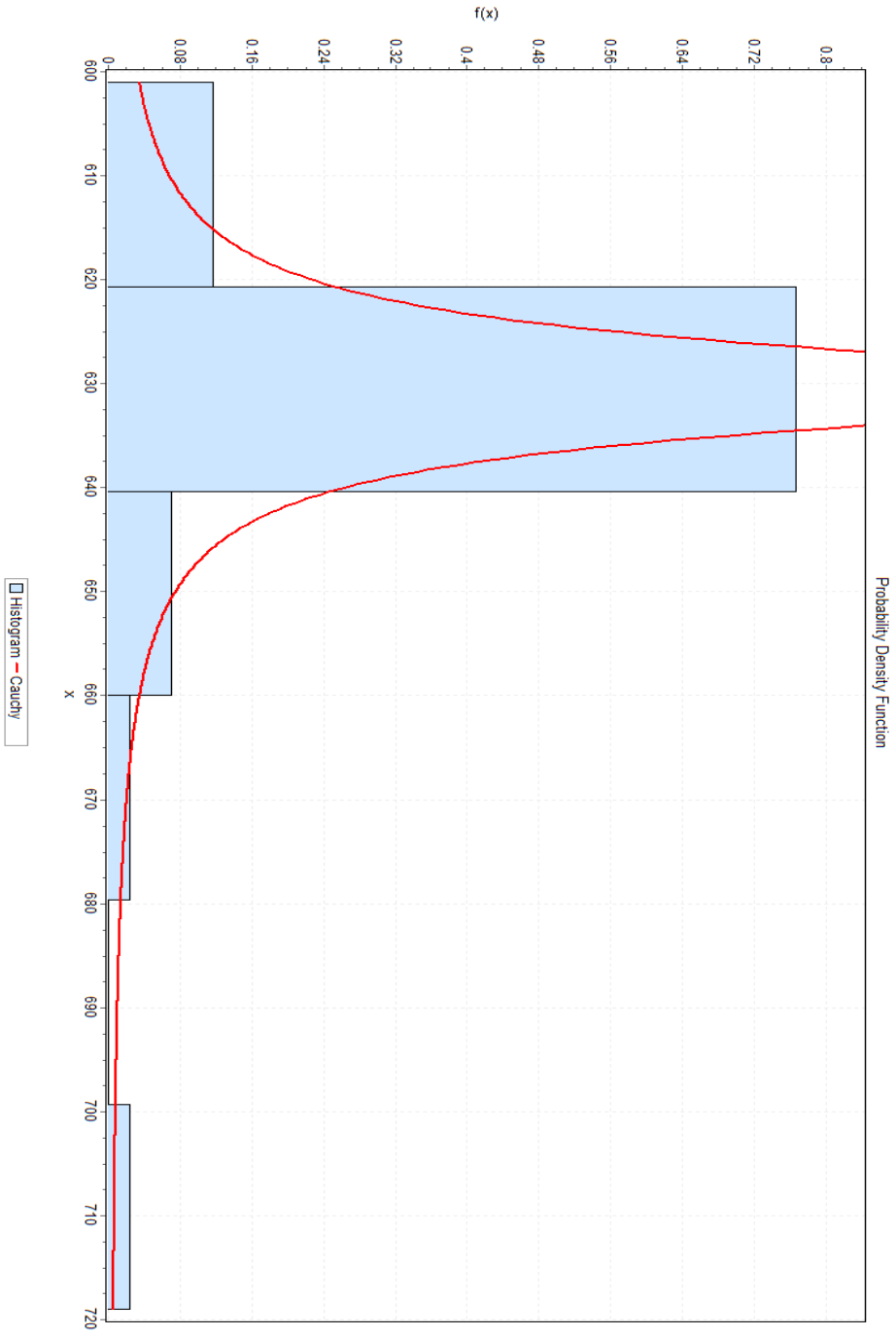

(a) Edge processing time (two objects per image)




(b) Probability distribution for edge detection (two objects per image)

'Cauchy' distribution due to its longer tail.

Shape detection time 4.4(b) varies from 57 milliseconds to 147 milliseconds. Mean value for the processing time $\mu$ is 97.67 milliseconds and standard deviation



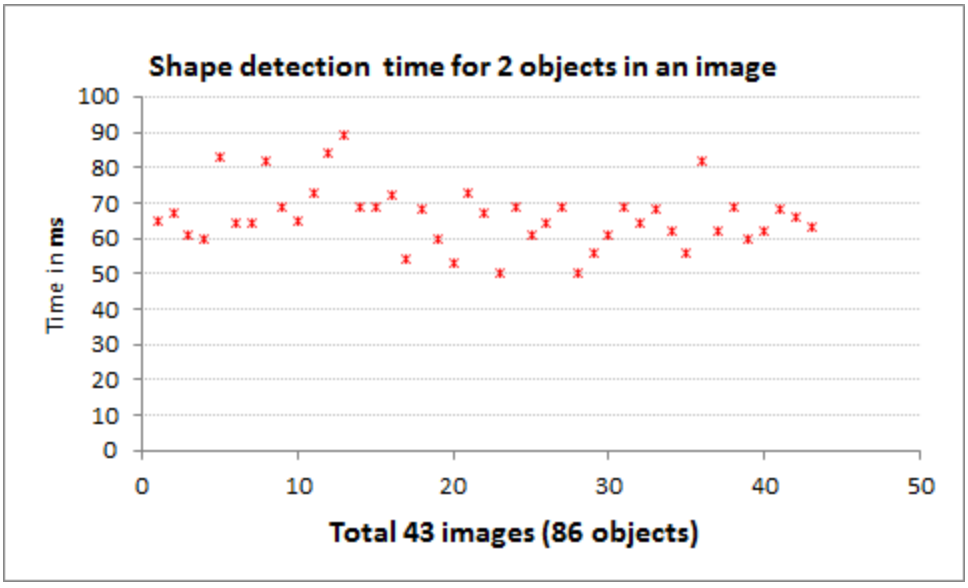

(c) Edge: Distribution function pattern for two objects per image (Cauchy)

| **Cauchy** [#4] | | | | | |
|---|---|---|---|---|---|
| Kolmogorov-Smirnov | | | | | |
| Sample Size<br>Statistic<br>P-Value<br>Rank | 43<br>0.07727<br>0.94202<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 0.15974 | 0.18257 | 0.20283 | 0.22679 | 0.24332 |
| Reject? | No | No | No | No | No |
| Anderson-Darling | | | | | |
| Sample Size<br>Statistic<br>Rank | 43<br>0.23666<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 1.3749 | 1.9286 | 2.5018 | 3.2892 | 3.9074 |
| Reject? | No | No | No | No | No |
| Chi-Squared | | | | | |
| Deg. of freedom<br>Statistic<br>P-Value<br>Rank | 4<br>2.494<br>0.64572<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 5.9886 | 7.7794 | 9.4877 | 11.668 | 13.277 |
| Reject? | No | No | No | No | No |

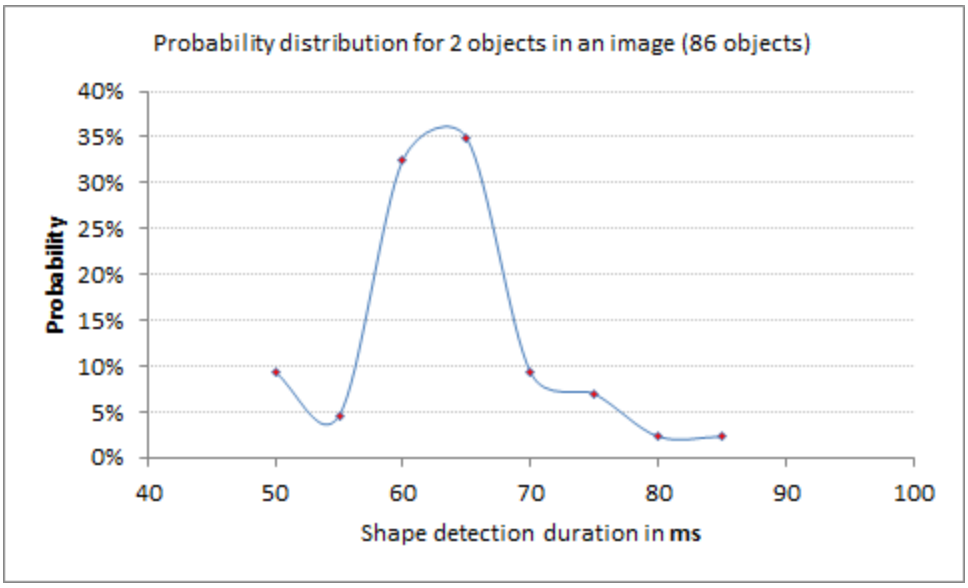
(d) Edge: Parameter analysis of Cauchy distribution for two objects per image

Figure 4.1: Analysis of edge processing time (two objects per image)

$\sigma$ is 13.31 milliseconds which is around 13.63%. The processing time range is wide as for few cases the algorithm fails to identify shape. If some of the objects in an image are not detected, the processing time will be less as it works on lower number of objects. That is why the standard deviation has higher percentage value than

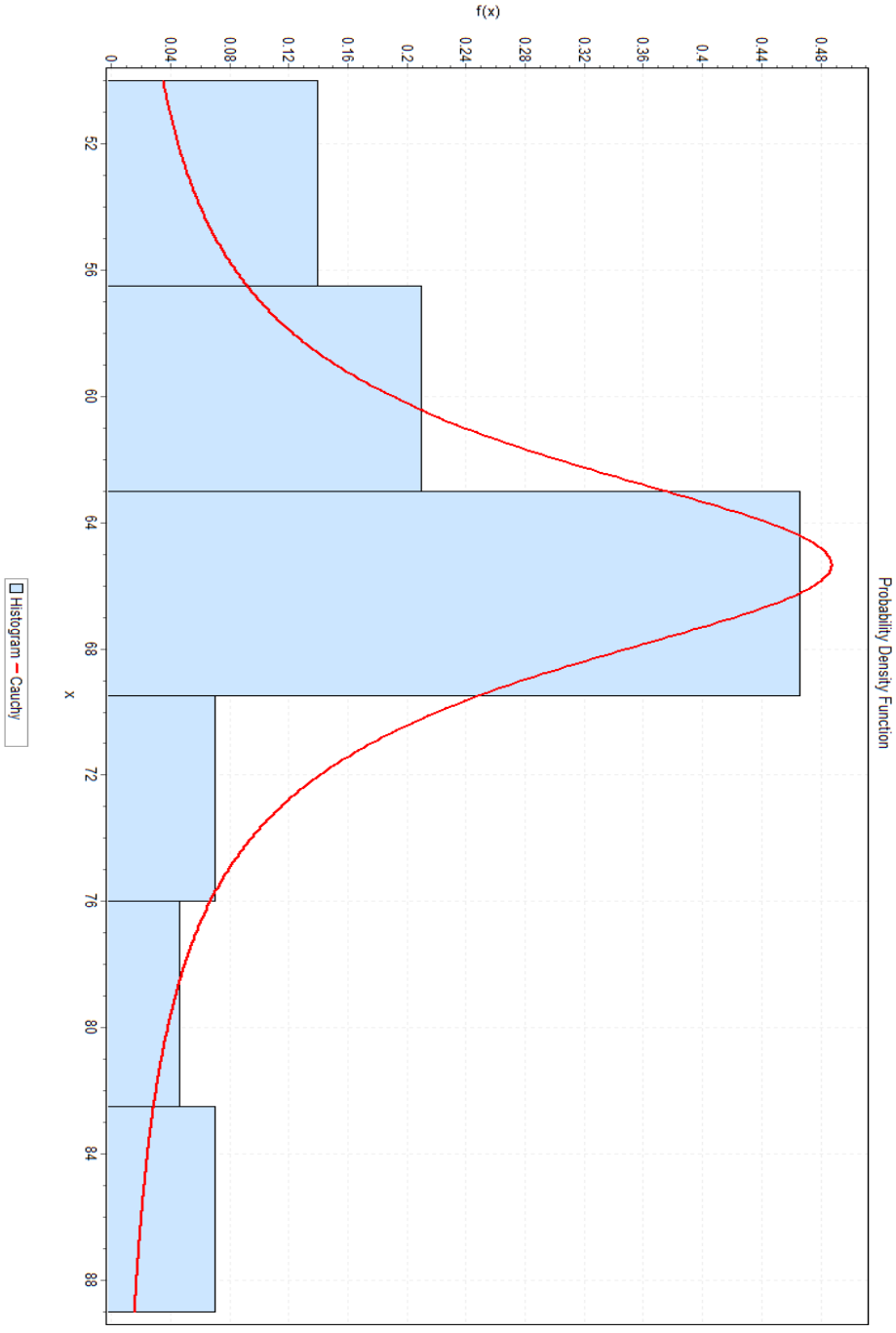



(a) Shape processing time (two objects per image)




(b) Probability distribution for shape detection (two objects per image)

the Edge detection technique. If we look at the probability distribution, it is right skewed. And that is why, the distribution is more closer to 'Burr' Distribution [28].




(c) Shape: Distribution function pattern for two objects per image (Cauchy)

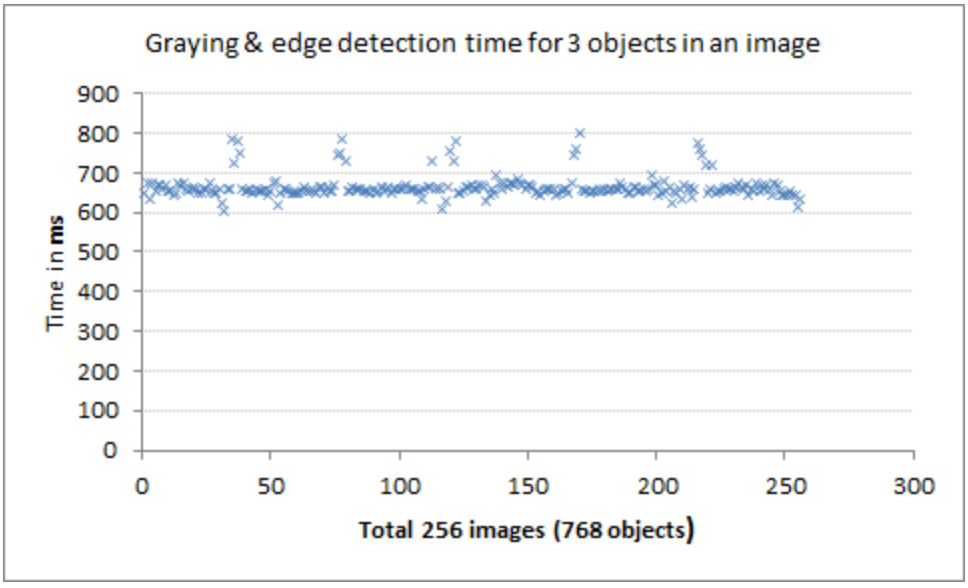
| **Cauchy** [#4] | | | | | |
|---|---|---|---|---|---|
| Kolmogorov-Smirnov | | | | | |
| Sample Size<br>Statistic<br>P-Value<br>Rank | 43<br>0.09698<br>0.7775<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 0.15974 | 0.18257 | 0.20283 | 0.22679 | 0.24332 |
| Reject? | No | No | No | No | No |
| Anderson-Darling | | | | | |
| Sample Size<br>Statistic<br>Rank | 43<br>0.57168<br>5 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 1.3749 | 1.9286 | 2.5018 | 3.2892 | 3.9074 |
| Reject? | No | No | No | No | No |
| Chi-Squared | | | | | |
| Deg. of freedom<br>Statistic<br>P-Value<br>Rank | 4<br>0.27515<br>0.99136<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 5.9886 | 7.7794 | 9.4877 | 11.668 | 13.277 |
| Reject? | No | No | No | No | No |

(d) Shape: Parameter analysis of Cauchy distribution for two objects per image

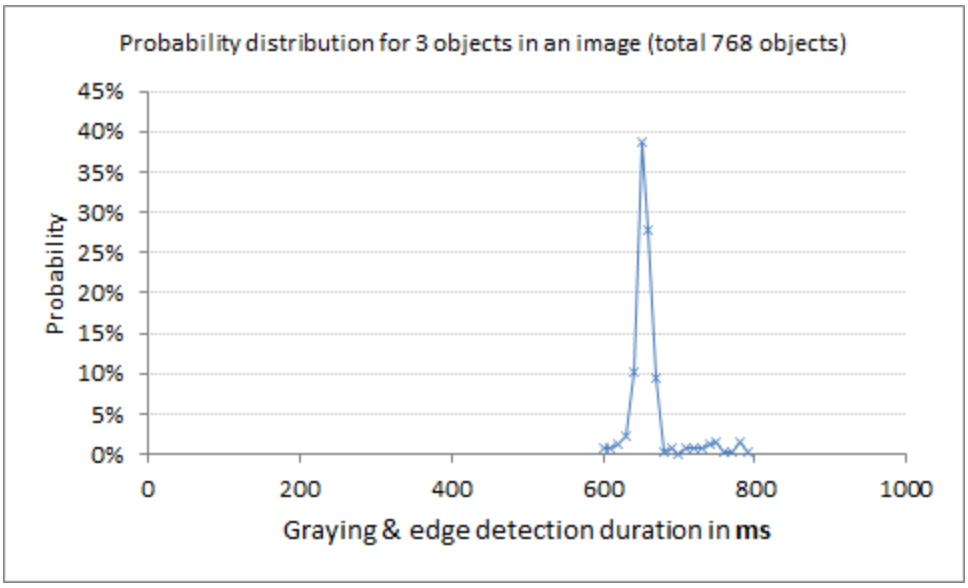
Figure 4.2: Analysis of shape processing time (two objects per image)

### 4.3.3 Four Objects per Image

Only seven sample images have been processed. The number of samples was not sufficient for valid results. Still we note that Edge detection time 4.5(b) varies from 676 milliseconds to 704 milliseconds. Mean value for the processing time

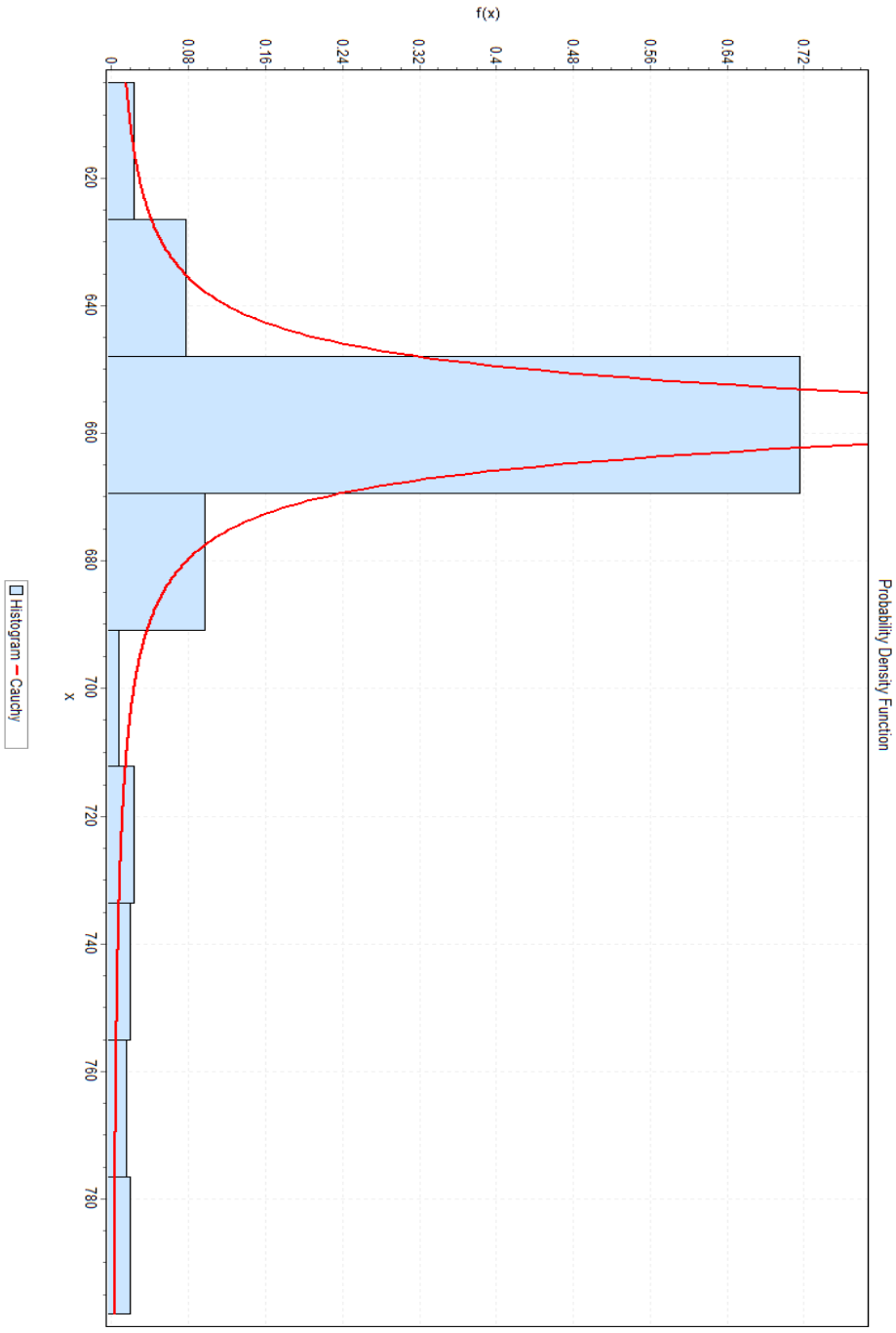



(a) Edge processing time (three objects per image)




(b) Probability distribution for edge detection (three objects per image)

$\mu$ is 690.15 milliseconds and standard deviation $\sigma$ is 8.21 milliseconds which is around 1.19%. The distribution is best fitted with 'Log-Logistic (3P)' [9] and has a longer tail. The probability function pattern is different from earlier two probability




(c) Edge: Distribution function pattern for three objects per image (Cauchy)

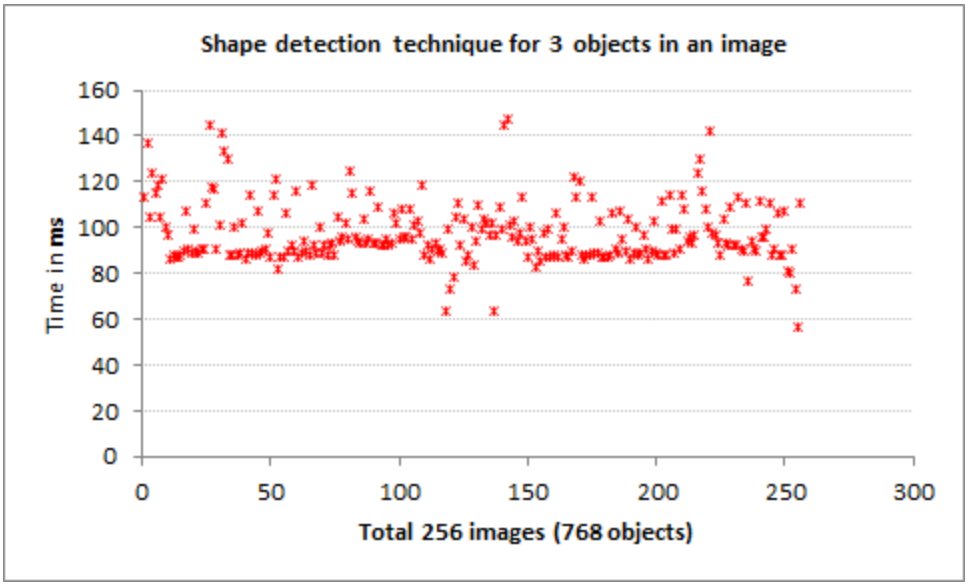
| **Cauchy** [#4] | | | | | |
|---|---|---|---|---|---|
| Kolmogorov-Smirnov | | | | | |
| Sample Size<br>Statistic<br>P-Value<br>Rank | 256<br>0.07883<br>0.07866<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 0.06706 | 0.07644 | 0.08488 | 0.09488 | 0.10181 |
| Reject? | Yes | Yes | No | No | No |
| Anderson-Darling | | | | | |
| Sample Size<br>Statistic<br>Rank | 256<br>3.0383<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 1.3749 | 1.9286 | 2.5018 | 3.2892 | 3.9074 |
| Reject? | Yes | Yes | Yes | No | No |
| Chi-Squared | | | | | |
| Deg. of freedom<br>Statistic<br>P-Value<br>Rank | 8<br>34.64<br>3.1092E-5<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 11.03 | 13.362 | 15.507 | 18.168 | 20.09 |
| Reject? | Yes | Yes | Yes | Yes | Yes |

(d) Edge: Parameter analysis of 'Cauchy' distribution for 3 objects per image

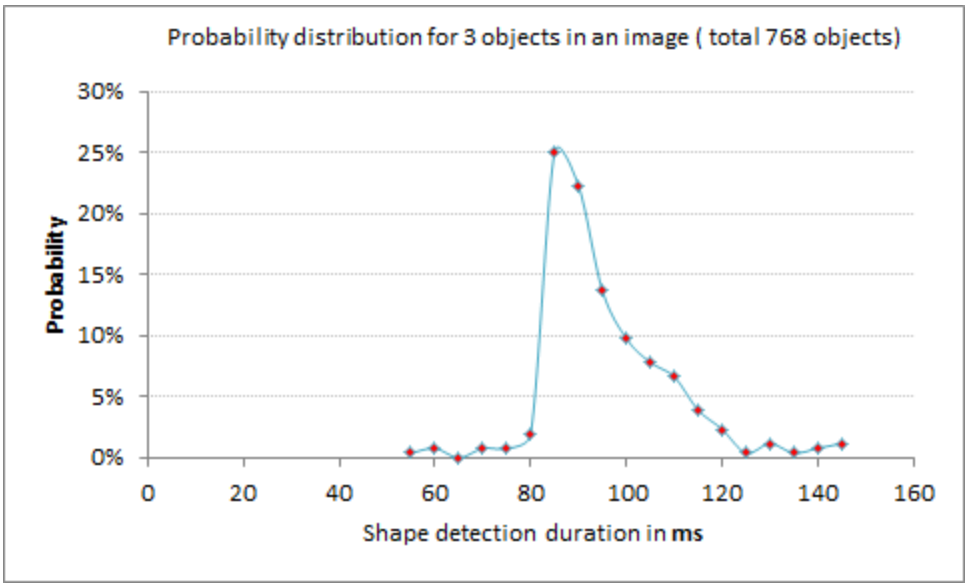
Figure 4.3: Analysis of edge processing time (three objects per image)

function patterns for edge detection technique.

In total seven sample images have been processed. Shape detection time 4.6(b) varies from 57 milliseconds to 147 milliseconds. Mean value for the processing time $\mu$ is 118.15 milliseconds and standard deviation $\sigma$ is 11.42 milliseconds which

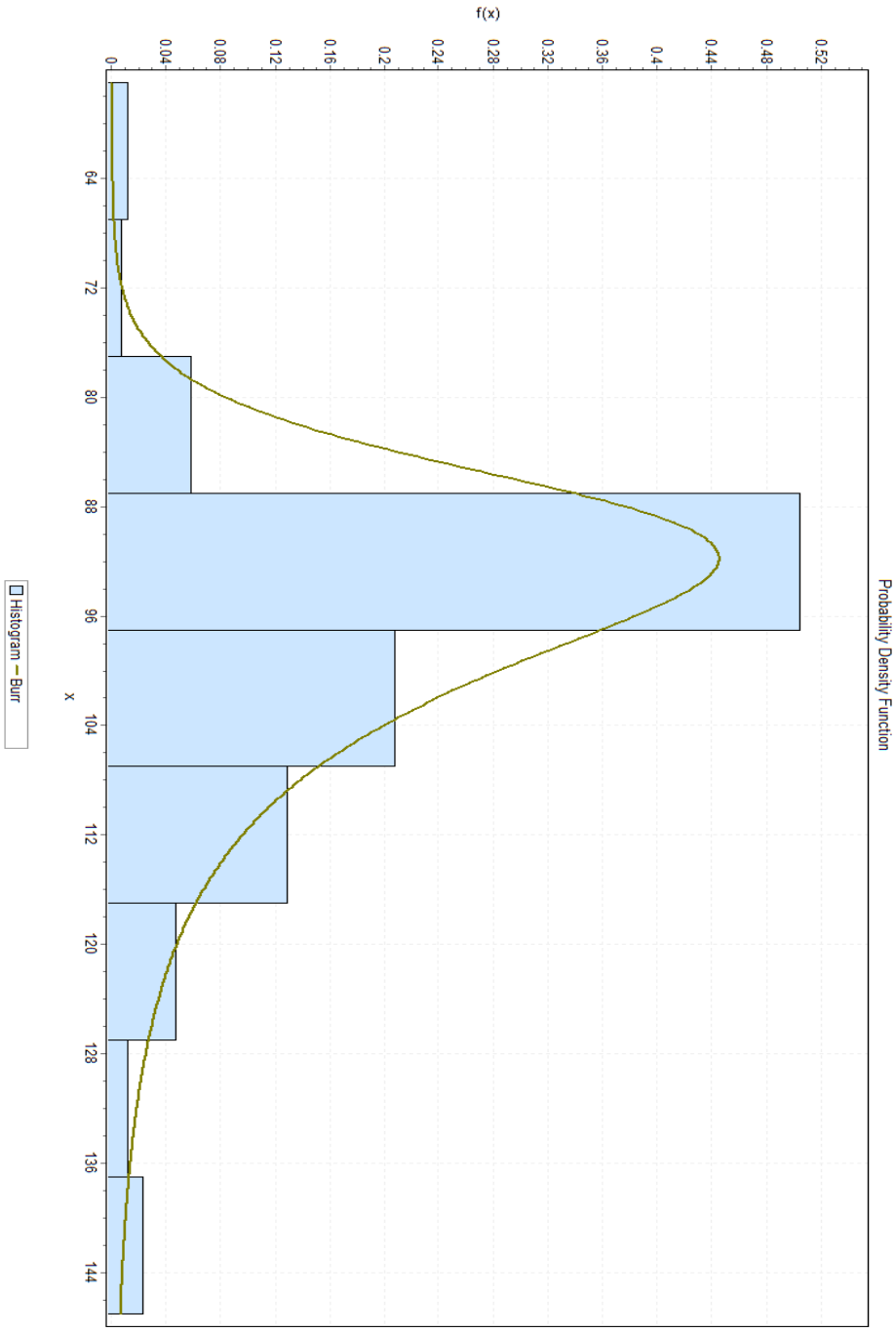



(a) Shape processing time (three objects per image)




(b) Probability distribution for shape detection (three objects per image)

is around 9.67%. The mean value is slightly higher compare to other two cases. The probability distribution is more fitted to Log-Logistic (3P) distribution. The sample value is not adequate enough in supporting the probability distribution.




(c) Shape: Distribution function pattern for three objects per image (Burr)

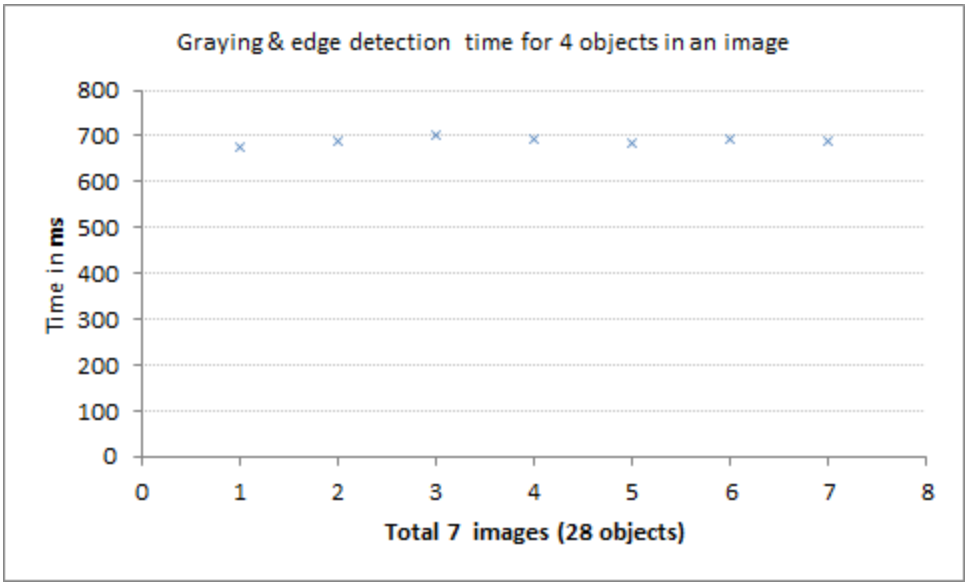
| **Burr** [#2] | | | | | |
|---|---|---|---|---|---|
| Kolmogorov-Smirnov | | | | | |
| Sample Size<br>Statistic<br>P-Value<br>Rank | 256<br>0.08829<br>0.03471<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 0.06706 | 0.07644 | 0.08488 | 0.09488 | 0.10181 |
| Reject? | Yes | Yes | Yes | No | No |
| Anderson-Darling | | | | | |
| Sample Size<br>Statistic<br>Rank | 256<br>2.2369<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 1.3749 | 1.9286 | 2.5018 | 3.2892 | 3.9074 |
| Reject? | Yes | Yes | No | No | No |
| Chi-Squared | | | | | |
| Deg. of freedom<br>Statistic<br>P-Value<br>Rank | 8<br>32.871<br>6.4960E-5<br>2 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 11.03 | 13.362 | 15.507 | 18.168 | 20.09 |
| Reject? | Yes | Yes | Yes | Yes | Yes |

(d) Shape: Parameter analysis of Burr distribution for three objects per image

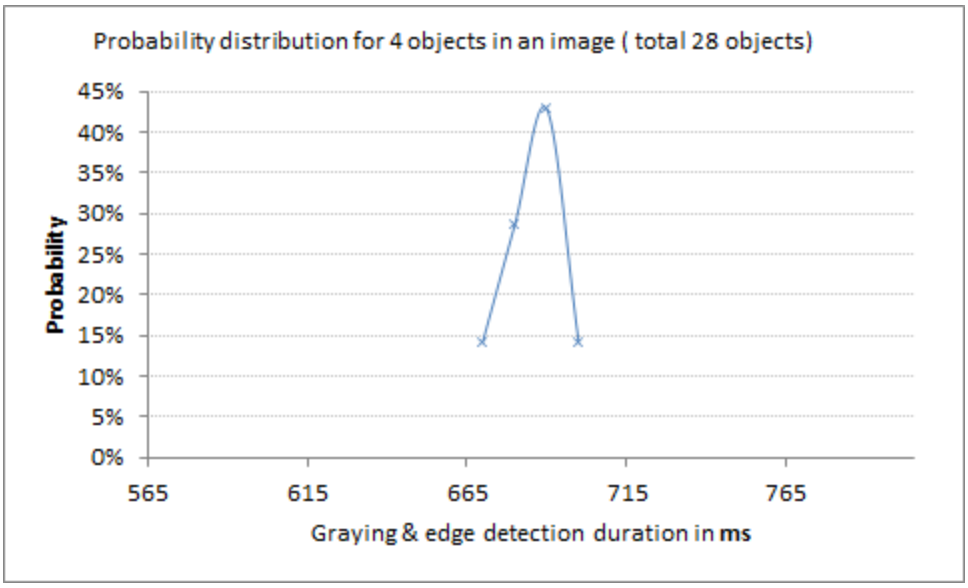
Figure 4.4: Analysis of shape processing time (three objects per image)

### 4.3.4 All Types of Images

Considering all type of images, edge detection time is almost same for the images with same size regardless of the number of objects or shapes in each image.

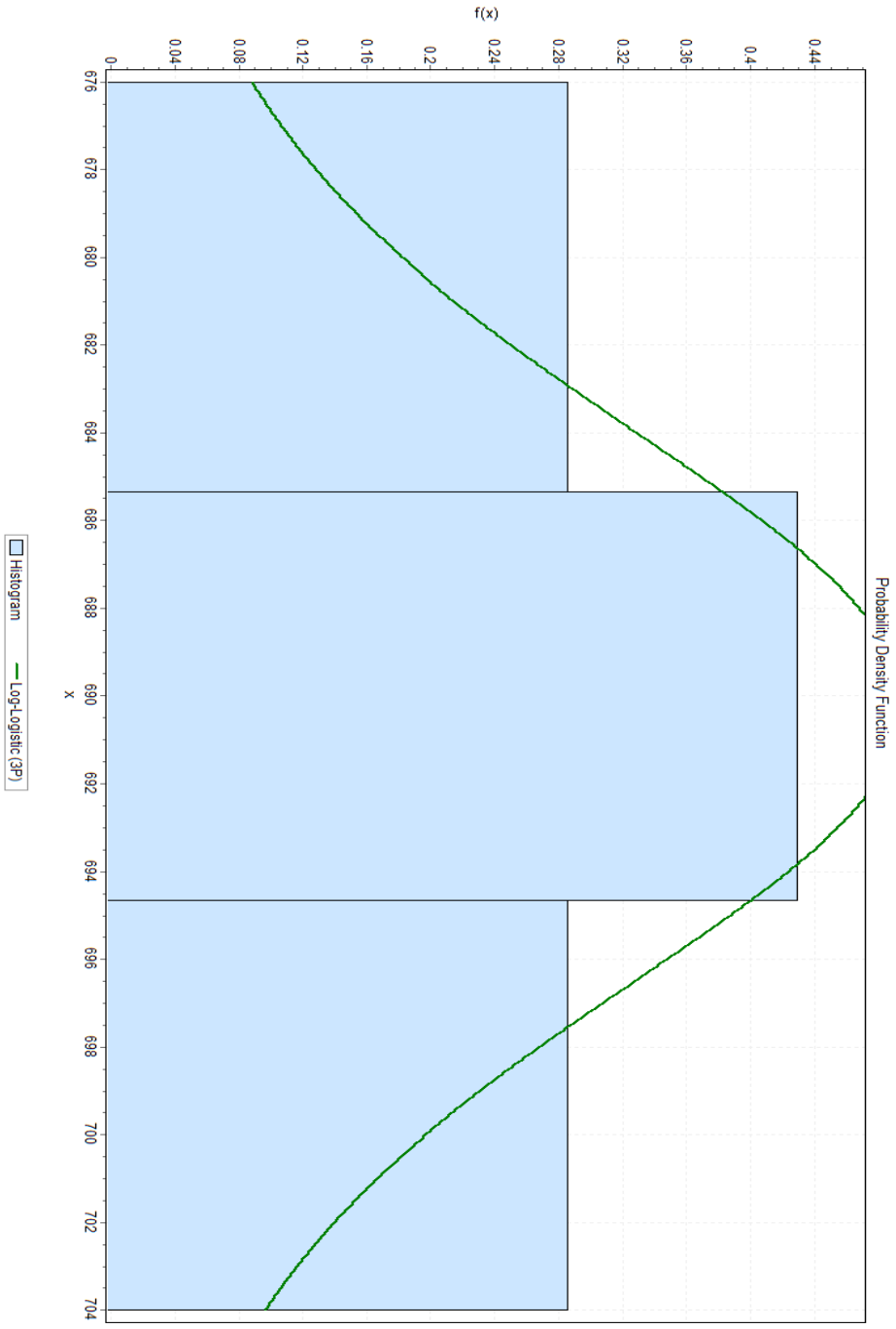



(a) Edge processing time (four objects per image




(b) Probability distribution for edge detection (4 objects per image)

This is quite understandable as edge detection technique processes each of the pixel in the image and hence the timing requirement will be almost same regardless of size and pattern of edges. If we look at the probability distribution, the pattern is




(c) Edge: Distribution function pattern for 4 objects per image (Log-Logistic(3P))

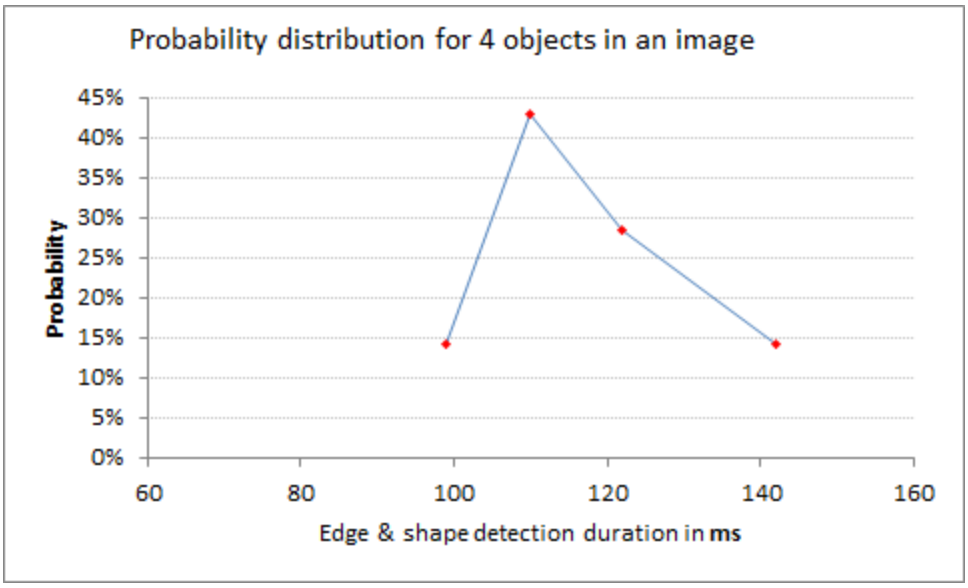
| **Log-Logistic (3P)** [#37] | | | | | |
|---|---|---|---|---|---|
| Kolmogorov-Smirnov | | | | | |
| Sample Size<br>Statistic<br>P-Value<br>Rank | 7<br>0.12415<br>0.99927<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 0.38148 | 0.43607 | 0.48342 | 0.53844 | 0.57581 |
| Reject? | No | No | No | No | No |
| Anderson-Darling | | | | | |
| Sample Size<br>Statistic<br>Rank | 7<br>0.14879<br>5 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 1.3749 | 1.9286 | 2.5018 | 3.2892 | 3.9074 |
| Reject? | No | No | No | No | No |

(d) Edge: Parameter analysis of Log-Logistic(3P) distribution for 4 objects per image

Figure 4.5: Analysis of edge processing time (four objects per image)

more similar to 'Cauchy' distribution as we have mentioned earlier that there is longer tail especially at the right side. We have experimented on 306 images (882 objects). Edge detection time varies between 584 milliseconds to 798 milliseconds. The edge detection time range seems longer as few points fall within the long tail. More than 90% of the samples fall within the range of 608 milliseconds to 695 milliseconds. The time may vary slightly as the processor can perform some internal tasks. Note that we have postponed all the external task so that the processor can allocate maximum time for this image processing. The standard

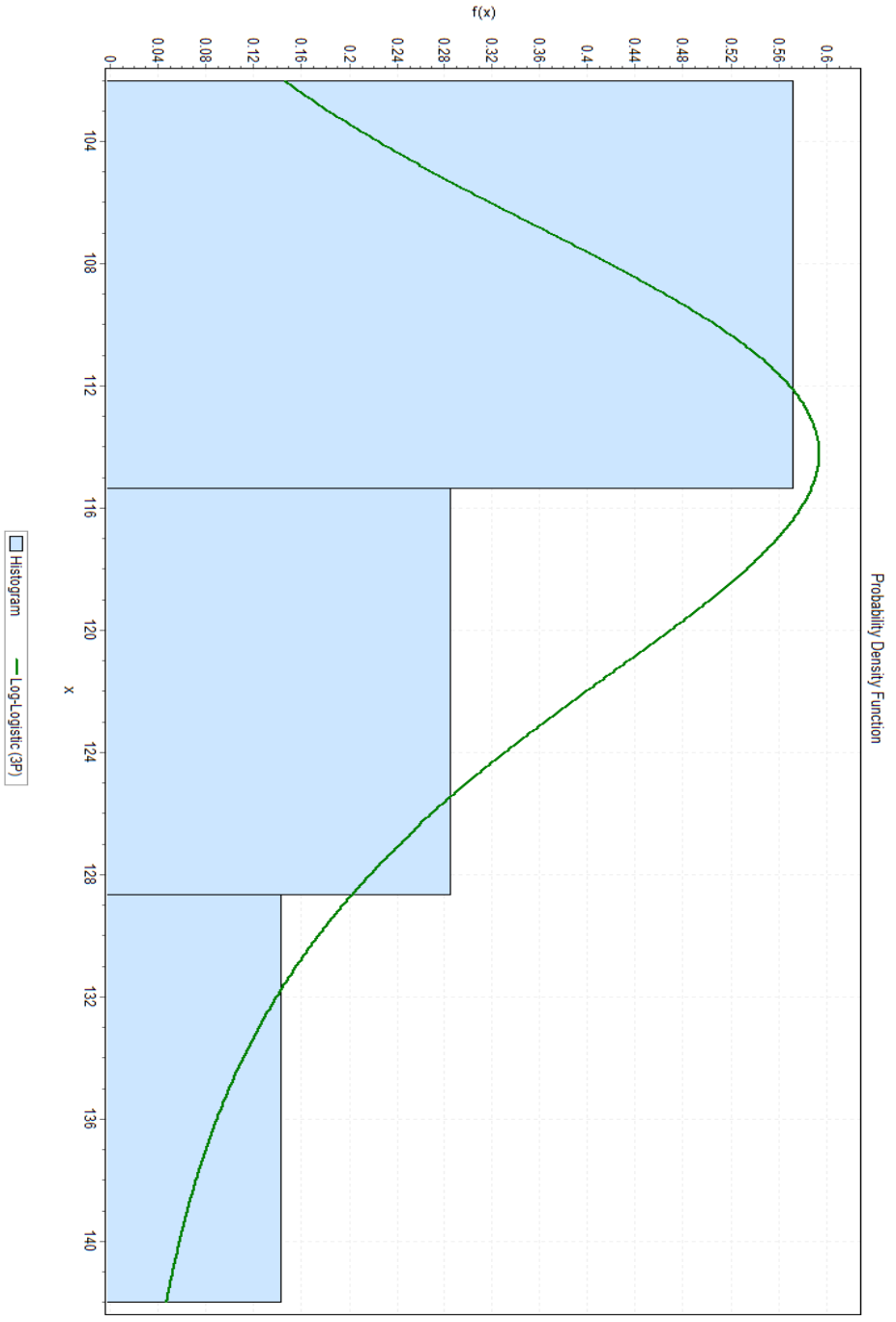

(a) Shape processing time (four objects per image)




(b) Probability distribution for shape detection (four objects per image)

deviation $\sigma$ is 30.22 milliseconds which is 4.58% of the processing time of 660.25 milliseconds and therefore, quite acceptable.

If we consider the shape detection time, the time varies with number of object



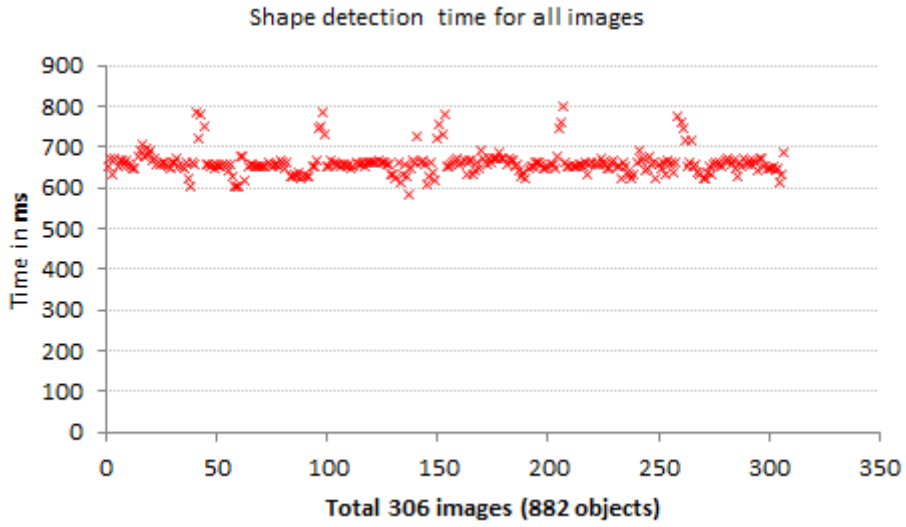

(c) Shape: Distribution function pattern for four objects per image (Log-Logistic(3P))

| **Log-Logistic (3P)** [#37] | | | | | |
|---|---|---|---|---|---|
| Kolmogorov-Smirnov | | | | | |
| Sample Size<br>Statistic<br>P-Value<br>Rank | 7<br>0.16682<br>0.97119<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 0.38148 | 0.43607 | 0.48342 | 0.53844 | 0.57581 |
| Reject? | No | No | No | No | No |
| Anderson-Darling | | | | | |
| Sample Size<br>Statistic<br>Rank | 7<br>0.24952<br>2 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 1.3749 | 1.9286 | 2.5018 | 3.2892 | 3.9074 |
| Reject? | No | No | No | No | No |

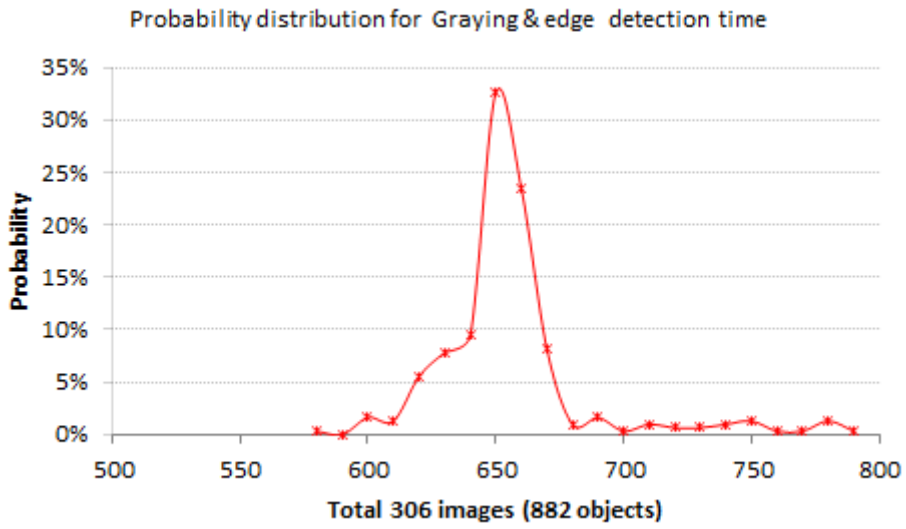
(d) Shape: Parameter analysis of Log-Logistic(3P) distribution for four objects pe image

Figure 4.6: Analysis of shape processing time (four objects per image)

in an image. In case of lower number of objects the processing time is lower with respect to the higher number of objects in a image. If we look at the algorithm of shape detection technique, it actually processes the object's edge points. In case of fewer objects in an image it requires less time. If we closely monitor the probability distribution point, three peak points are visible. The left side peak points denote actually two objects per image zone, the middle high peak denotes the three objects zone and the right hand side peak represents the four objects per image. The processing time varies from 50ms to 147 milliseconds. The range is long

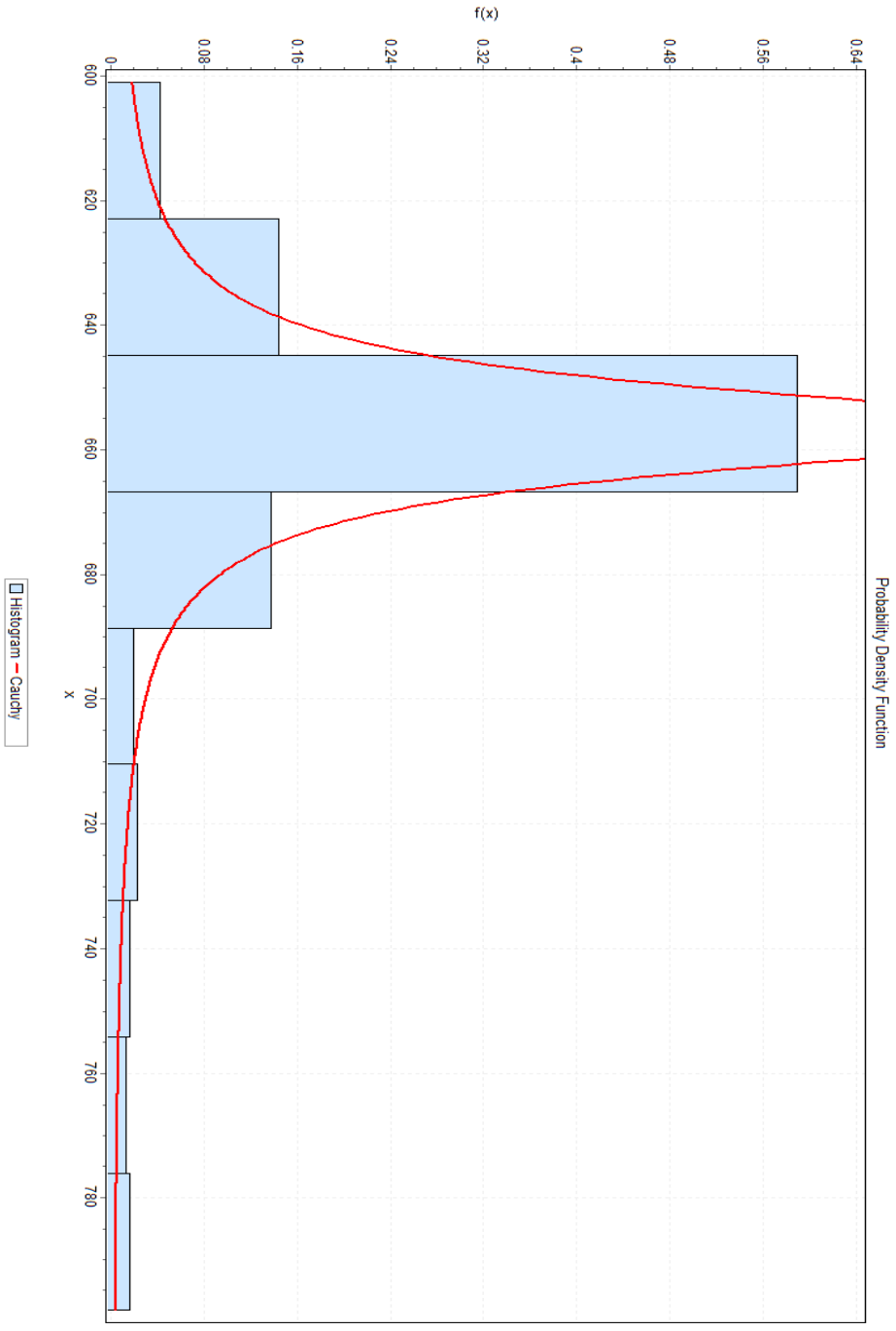



(a) Edge processing time (all types of objects)




(b) Probability distribution for edge detection (all types of objects)

as different category requires different processing time. The range is further more widened as rare but for some cases it can not identify partial number of objects in an image. This results in less processing time. In overall, the average processing



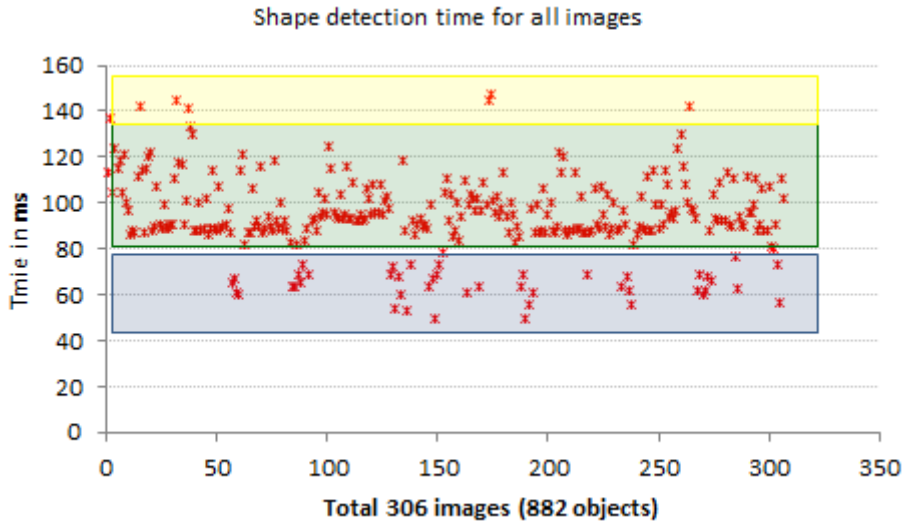

(c) Edge:Pattern of distribution for all types of objects (Cauchy)

| **Cauchy** [#4] | | | | | |
|---|---|---|---|---|---|
| Kolmogorov-Smirnov | | | | | |
| Sample Size<br>Statistic<br>P-Value<br>Rank | 306<br>0.04592<br>0.52377<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 0.06134 | 0.06991 | 0.07763 | 0.08678 | 0.09312 |
| Reject? | No | No | No | No | No |
| Anderson-Darling | | | | | |
| Sample Size<br>Statistic<br>Rank | 306<br>1.0502<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 1.3749 | 1.9286 | 2.5018 | 3.2892 | 3.9074 |
| Reject? | No | No | No | No | No |
| Chi-Squared | | | | | |
| Deg. of freedom<br>Statistic<br>P-Value<br>Rank | 8<br>13.26<br>0.1032<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 11.03 | 13.362 | 15.507 | 18.168 | 20.09 |
| Reject? | Yes | No | No | No | No |

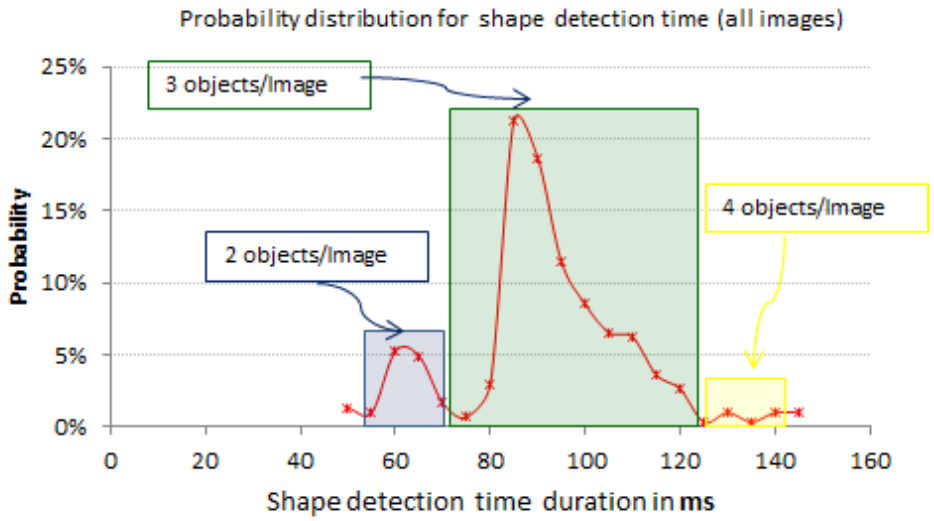
(d) Edge:Parameter analysis of Cauchy distribution for all types of objects

Figure 4.7: Analysis of edge processing time (all types of objects)

time is 93.7 milliseconds and $\sigma$ deviation is 18.75 milliseconds which is around 20% of the mean value. The probability distribution function is a combination of three sets of images' shape processing time and does not resemble to any specific type. Nevertheless, this data set resembles most closely Laplace distribution.

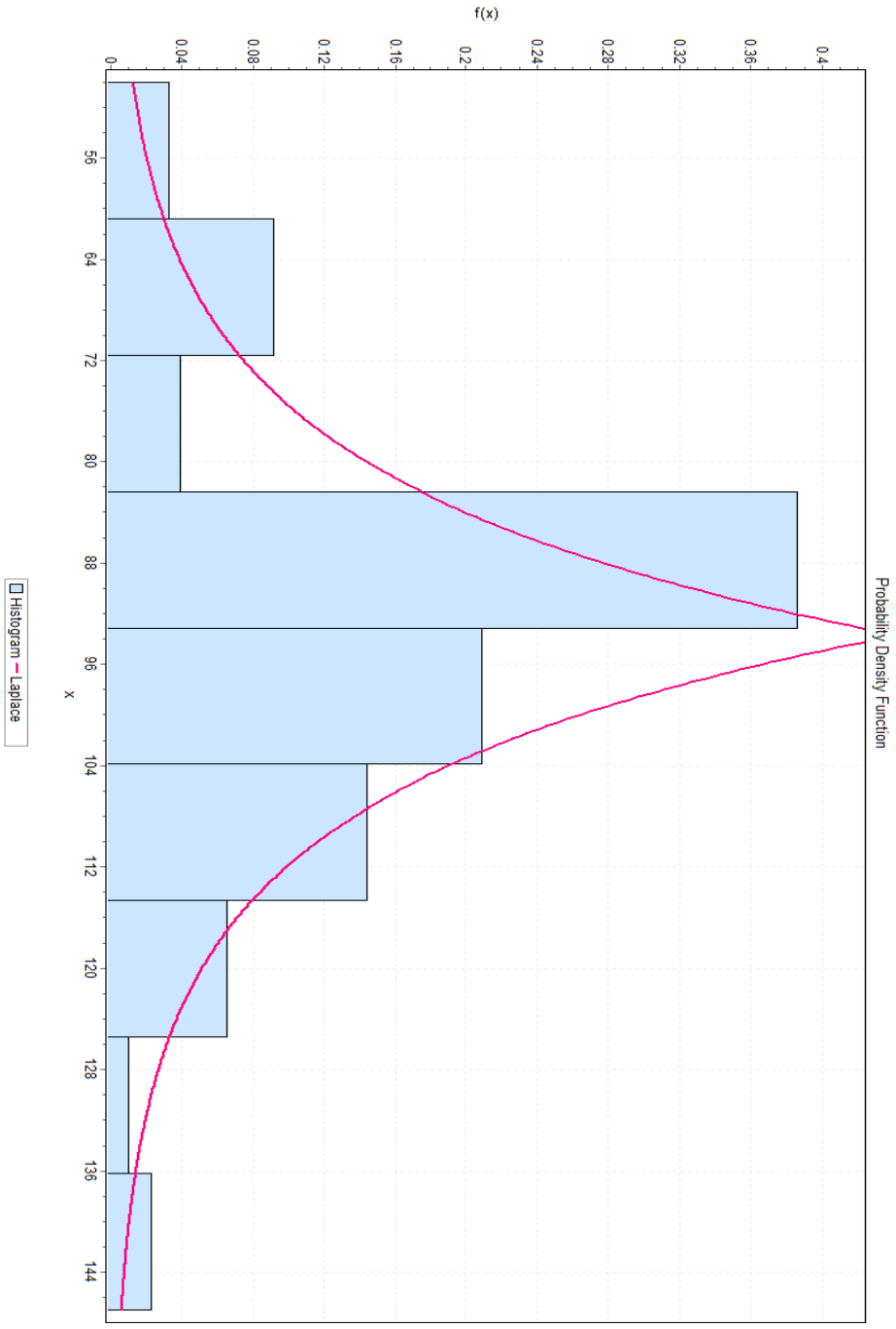



(a) Shape processing time (all types of objects)




(b) Probability distribution for shape detection for all types of objects

Now, we need to inspect if the total processing time including edge detection and shape detection is more than one second or not. The Figure 4.9(a) shows that total processing time is below one second. Mean time for processing the image is



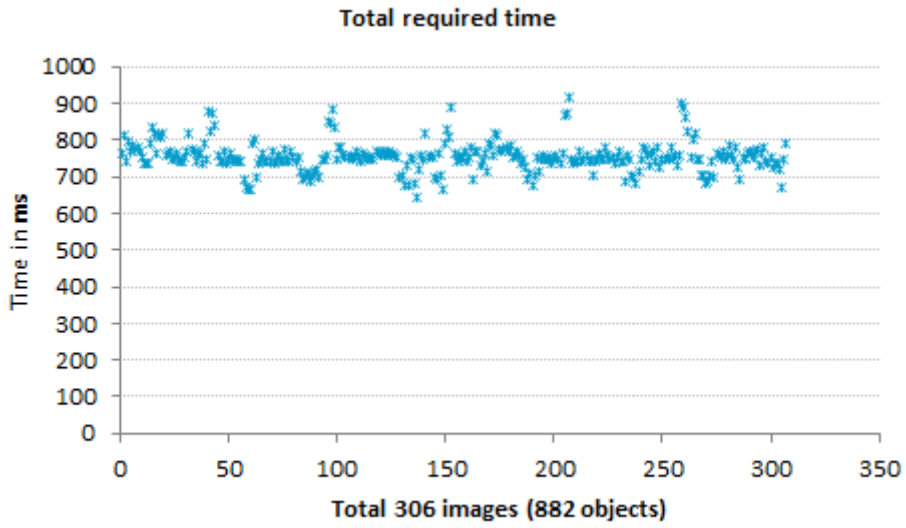

(c) Shape: Pattern of distribution for all types of objects (Laplace))

| **Laplace** [#32] | | | | | |
|---|---|---|---|---|---|
| Kolmogorov-Smirnov | | | | | |
| Sample Size<br>Statistic<br>P-Value<br>Rank | 306<br>0.08569<br>0.02102<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 0.06134 | 0.06991 | 0.07763 | 0.08678 | 0.09312 |
| Reject? | Yes | Yes | Yes | No | No |
| Anderson-Darling | | | | | |
| Sample Size<br>Statistic<br>Rank | 306<br>2.3774<br>1 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 1.3749 | 1.9286 | 2.5018 | 3.2892 | 3.9074 |
| Reject? | Yes | Yes | No | No | No |
| Chi-Squared | | | | | |
| Deg. of freedom<br>Statistic<br>P-Value<br>Rank | 8<br>57.031<br>1.7761E-9<br>2 | | | | |
| α | 0.2 | 0.1 | 0.05 | 0.02 | 0.01 |
| Critical Value | 11.03 | 13.362 | 15.507 | 18.168 | 20.09 |
| Reject? | Yes | Yes | Yes | Yes | Yes |

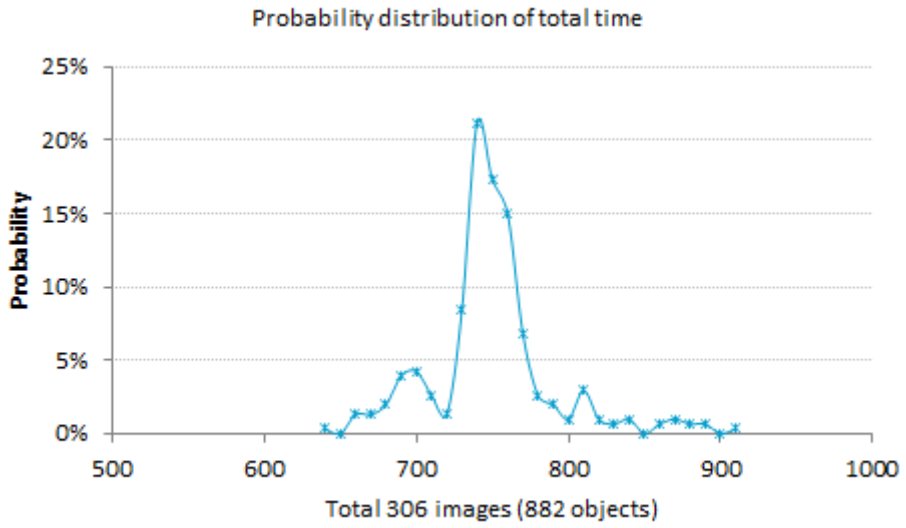
(d) Shape: Parameter analysis of Laplace distribution for all types of objects

Figure 4.8: Analysis of shape processing time (all types of objects)

753.85 milliseconds and standard deviation $\sigma$ is 40.17 milliseconds which is 5.32% of the mean time. In the probability distribution function there are three peaks which are due to varying shape detection time. But still more than 80% values are within second quartile of standard deviation.




(a) Total Processing Time




(b) Probability Distribution

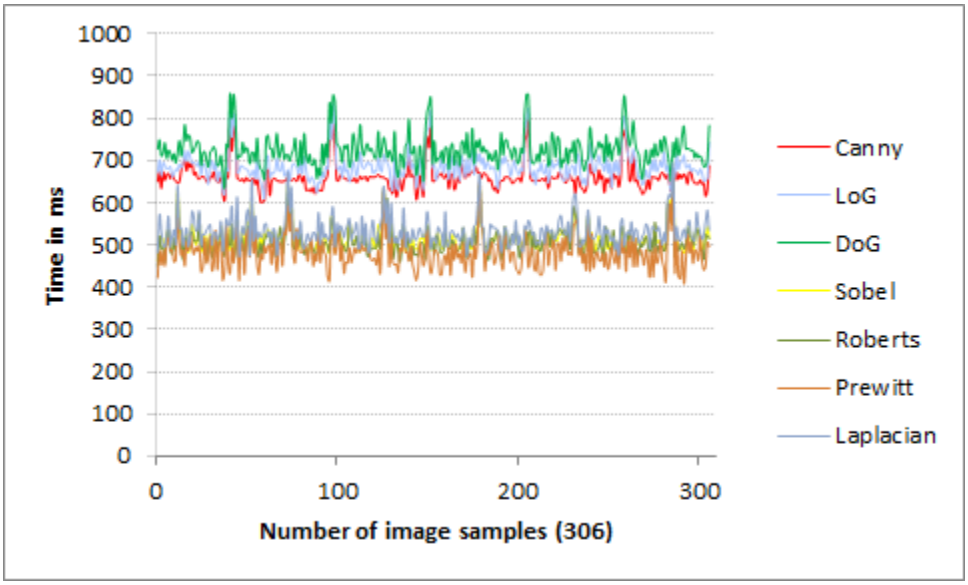
Figure 4.9: Total Processing time for all Images

## 4.4 Comparison of Edge Detection Techniques

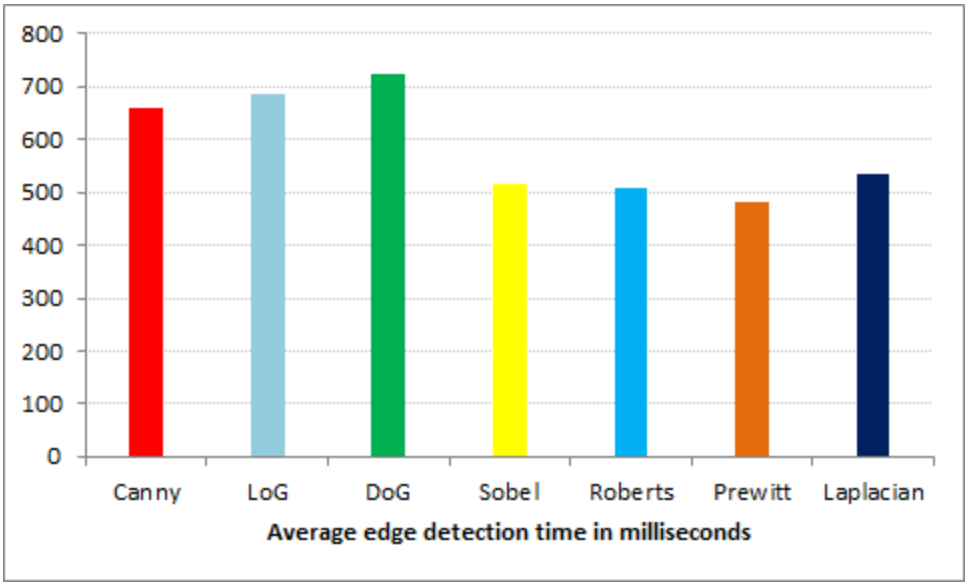
In chapter 2 we have discussed on various edge detection techniques. In our implementation, we have kept the shape detection mechanism unique and that is aforge.net shape detection method. But we have to keep in our mind that, the output of the edge detection technique is the input of shape detection technique. So if the edge detection mechanism is erroneous, than the shape detection would be. We have experimented in total seven detection methods including the Canny method. Our Canny Edge detection implementation is modular basis and we can easily apply new technique switching of some modules and adding the respective matrix for the algorithm.

First, we have analysed the processing time. If we look at the figure 4.10(a); mean processing time of Canny, Difference of Gaussian (DoG) and Laplacian of Gaussian (LoG) methods take slightly higher time than those of the Sobel, Roberts, Prewitt and Laplacian methods. The former three methods perform additional smoothing of the image to reduce noise. In order to do so each pixel is multiplied with small kernel values which costs additional processing time. We have analyzed if this additional cost is worthy or not later in this section. The shape detection technique might not detect all the objects. Now the big question is, are all the detected objets categorised appropriately? These detected objects are either physically good or bad. The main purpose of developing this application is to generate output if the objects are detected with accuracy. To experiment the physical condition of the object, we need to inspect the pixel value. For a regular undamaged object the body color will be maintained almost same level. There is little change in neighbouring

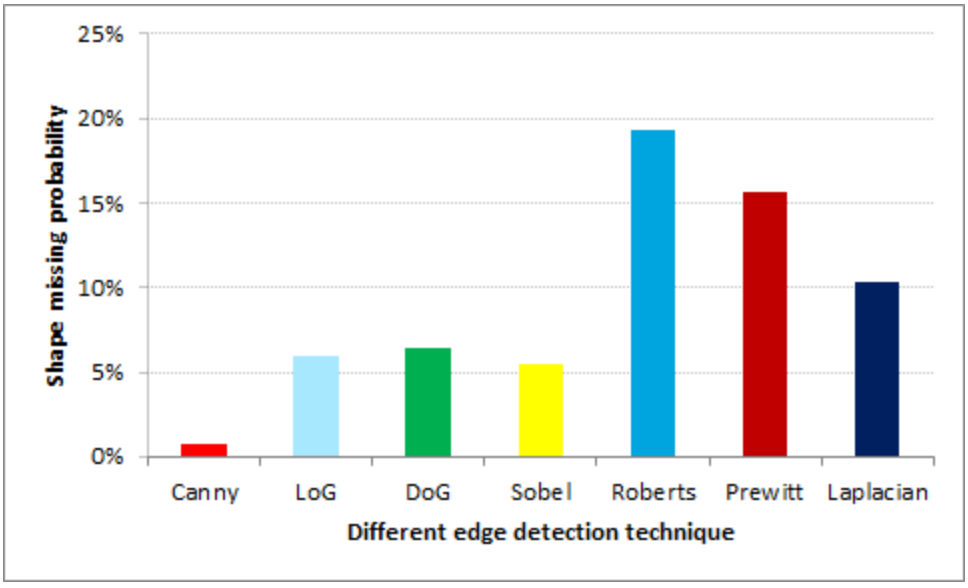
pixel color values. And after edge detection technique, this area will be almost black as there is no pixel value change. For physically damaged object, the surface area will be abrupt.



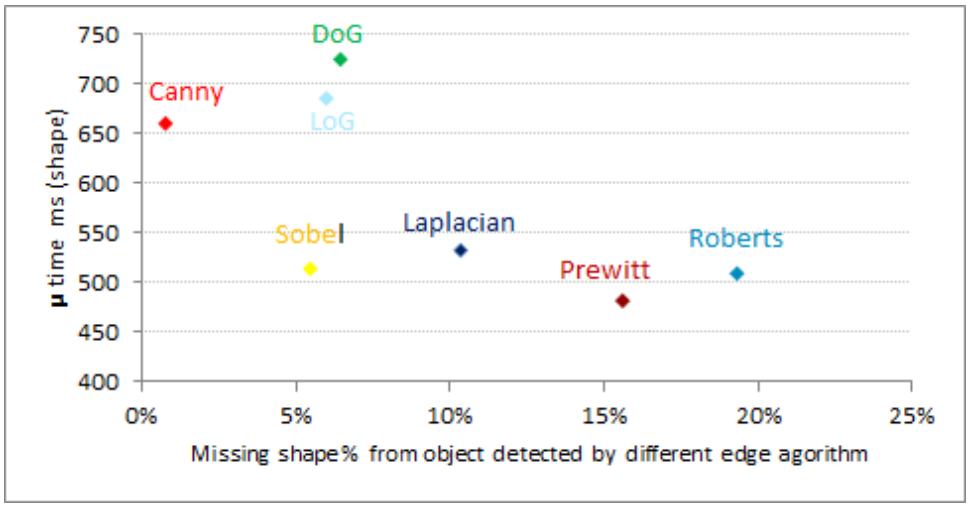

(a) Edge processing times (actual values)




(b) Mean processing time for different techniques




(c) Probability that shape detection will not be successful




(d) Processing time vs probability of unsuccessful shape detection

Figure 4.10: Analysis of Different Edge Detection Techniques

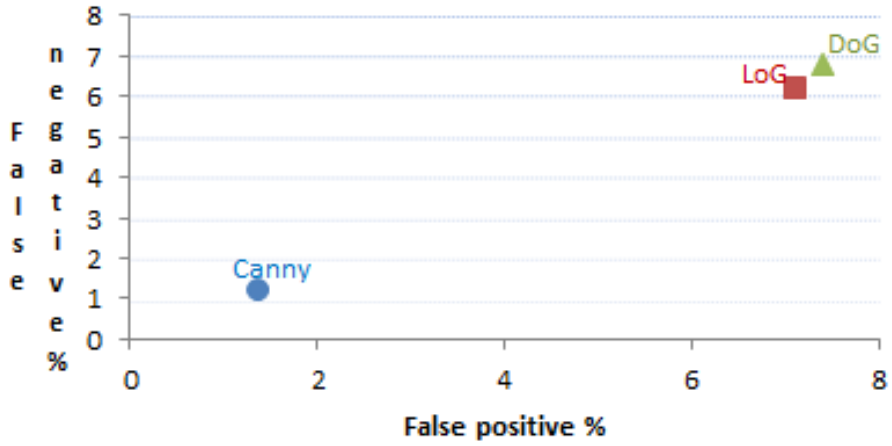



Figure 4.11: Physical damage error percentage

The pixel values for damage object will not be regular. And hence there will be more whitish area for damaged object. We have counted the pixels value for an object. If the value is higher than the threshold value than we declare a damaged object. There could be scenario that the object is not physically damaged, but the surface area pixel color is not homogeneous. It could be due to heavy external light reflection on the object or Edges of two surface area of an object may produce different color as these part of the object is more vulnerable and attaching to dices of the bakery. Nevertheless, we will monitor the performance.

As mentioned earlier, Sobel, Roberts, Prewitt and Laplacian methods are vulnerable to noise. Namely, they allow us to detect an object but not to get meaningful information about its interior. The interior, especially in case of damaged items, is blurred due to noise and it is impossible to decide whether an item is damaged or not.

For calculating the error percentage in detecting damaged items, our experiment was limited to Canny edge detection, DoG, and LoG methods. The Figure 4.11 shows the error percentages (in detecting damaged items) of 'False positive' and 'False negative'. For the perfect matching, both "False positive" and "False negative" error percentage values are zero. From the experiment, in case of Canny edge detection method, both the error percentages are less than 1.5%. 'False positive' and 'False negative' error percentages for DoG and LoG are more than 6%; i.e overall error percentage is more than 12% for these two techniques. Therefore, Canny is superior to any other technique in terms of accuracy. Though Canny edge detection requires additional processing time, it is worthy as it shows the highest accuracy in the experiment.

## 4.5 Summary of Analysis

In this chapter we have compared the performance of a number of edge detection techniques. The most important goal of this application is to inspect a bakery product with accuracy without human interruption. One of the key requirements was to perform the whole processing in less than one second. We have achieved that goal: not a single image sample has taken more than one second. Though Canny processing time is a little bit high with respect to some other techniques, but it is paid off. From the experiment it is clear that, Canny Edge detection technique shows superior accuracy with respect to all other techniques.

# Chapter 5

# Conclusion and Future Work

This chapter concludes the dissertation with a brief summary of contributions and future direction of the research.

## 5.1 Contributions

The key contribution of this thesis is to develop a new improved application which can process images of bakery system. We have applied Canny Edge detection method in this application. Performance wise it shows better with respect to Sobel and other industrial application. Our experiment has shown 4 to 5 fold improvement in accuracy compared to other techniques. The processing time is within the acceptance level. The application is modular basis and hence any new attribute can be added with minimum effort. Customization is one of the key point. One can tune the parameters as per requirements. Nevertheless, the cost of the application will be lower and thus bakery industry will be beneficial by using this.

## 5.2 Future Work

We have already handed over the developed application to the Industrial Partner [15]. Further fine tuning of the parameters will be required before commercial use. In addition, we would like to extend our development to detect more object shapes (like Ficelle, Baton etc.). We would like to process temperature of object body by adding new module in the application and by introducing infrared camera in the hardware system. We would analyze the light effect on object extraction capabilities and shade reduction techniques. Edge detection in our application is calculated by applying gradient method (change in value between neighbour pixels). Two significant features of Canny edge detection method are introduction of NMS (non-maximum suppression) and double thresholding of the gradient image. Due to poor illumination, the region boundaries in an image may become vague, creating uncertainties in the gradient image. We would upgrade the existing gradient based Canny to Fuzzy Logic based Canny method which would likely enhance the performance.

# Appendix A

# Abbreviations

| | |
|---|---|
| *ADO* | Activex Data Objects |
| *DB* | Data Base |
| *DLL* | Dynamic Link Library |
| *DoG* | Difference of Gaussian |
| *GHT* | Generalized Hough Transform |
| *HSB* | Hue Saturation and Brightness |
| *HT* | Hough Transform |
| *K − S* | Kolmogorov–Smirnov |
| *LED* | Light Emitting Diode |
| *LoG* | Laplace of Gaussian |
| *MS* | Microsoft |
| *NMS* | Non Maximum Suppression |
| *PDF* | Probability Distribution Function |
| *SDK* | Software Development Kit |

| | |
|---|---|
| *SQL* | Structured Query Language |
| *SNR* | Signal to Noise Ratio |
| $T_H$ | High Threshold |
| $T_L$ | Low Threshold |
| *RGB* | Red Green and Blue |
| *VS* | Visual Studio |

# Bibliography

[1] W. Arendt, C. Batty, M. Hieber, and F. Neubrander, "Cauchy Problems," in *Vector-valued Laplace Transforms and Cauchy Problems*, ser. Monographs in Mathematics. Springer Basel, 2011, vol. 96, pp. 107–238.

[2] D. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111 – 122, 1981.

[3] M. Barni, F. Bartolini, and A. Piva, "Improved wavelet-based watermarking through pixel-wise masking," *Image Processing, IEEE Transactions on*, vol. 10, no. 5, pp. 783 –791, May 2001.

[4] R. Biswas and J. Sil, "An Improved Canny Edge Detection Algorithm Based on Type-2 Fuzzy Sets," *Procedia Technology*, vol. 4, no. 0, pp. 820 – 824, 2012.

[5] V. Chickanosky and G. Mirchandani, "Wreath products for edge detection," in *Proceedings of the 1998 IEEE International Conference on*, USA,WA,Seattle, 1998.

[6] L. Ding and A. Goshtasby, "On the Canny edge detector," *Pattern Recognition*, vol. 34, no. 3, pp. 721 – 725, 2001.

[7] Fermanian, J.D. and Radulovic, D. and Wegkamp, M.H., "An improved

Kolmogorov-Smirnov test for copulas," in *Workshop Copulae in Mathematical and Quantitative Finance Preliminary Book of Abstracts May 20, 2012*, 2012, p. 13.

[8] H. Fernandes, P. Costa, V. Filipe, L. Hadjleontiadis, and J. Barroso, "Stereo vision in blind navigation assistance," in *World Automation Congress (WAC)*, Vallarta, Mexico, 2010, pp. 1–6.

[9] D. Fitzgerald, "Analysis of extreme rainfall using the log logistic distribution," *Stochastic Environmental Research and Risk Assessment*, vol. 19, pp. 249–257, 2005.

[10] O. Golan, S. Kiro, and I. Horovitz, "Method and System for Edge Detection," U.S. Patent 20 120 243 793, September, 2012.

[11] P. Hough, "Method and means for recognizing patterns represented in logarithmic polar coordinates," Tech. Rep., 1962.

[12] I. S. Icon Group International and I. G. Ltd., *Keyence Corp.: Labor Productivity Benchmarks and International Gap Analysis*, 1st ed. Icon Group International, Incorporated, 2000.

[13] B. Kamgar-Parsi and A. Rosenfeld, "Optimally isotropic Laplacian operator," *Image Processing, IEEE Transactions on*, vol. 8, no. 10, pp. 1467–1472, 1999.

[14] A. Kirillov, "AForge .NET framework," 2009. [Online]. Available: http://www.aforgenet.com

[15] G. Kundacina and M. Kundacina, "More automation solution inc. ," 2012. [Online]. Available: http://www.moreautomation.com

[16] X. Li, H. Zhao, X. Jin, and X. Qin, "Real-Time Image Smoothing Based on True Edges," in *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2011 12th International Conference on*, sept. 2011, pp. 141 –145.

[17] J. Maldacena, "Gravity, particle physics and their unification," *International Journal of Modern Physics A*, vol. 15, no. supp01b, pp. 840–852, 2000.

[18] MathWave Technologies, "Easyfit: Distribution Fitting Software," 2012. [Online]. Available: http://www.mathwave.com

[19] I. Mayergoyz, "Mathematical models of hysteresis," *Magnetics, IEEE Transactions on*, vol. 22, no. 5, pp. 603–608, 1986.

[20] J. Milnor, "Eigenvalues of the Laplace operator on certain manifolds," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 51, no. 4, p. 542, 1964.

[21] S. Nabavi, B. Vijaya Kumar, J. Bain, C. Hogg, and S. Majetich, "Application of Image Processing to Characterize Patterning Noise in Self-Assembled Nano-Masks for Bit-Patterned Media," *Magnetics, IEEE Transactions on*, vol. 45, no. 10, pp. 3523 –3526, oct. 2009.

[22] R. ONeil, "Convolution operators and $L(p, q)$ spaces," *Duke Mathematical Journal*, vol. 30, no. 1, pp. 129–142, 1963.

[23] O. Paquet-Durand, D. Solle, M. Schirmer, T. Becker, and B. Hitzmann, "Monitoring baking processes of bread rolls by digital image analysis," *Journal of Food Engineering*, vol. 111, no. 2, pp. 425 – 431, 2012.

[24] I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. Kim, "Design and Performance Evaluation of Image Processing Algorithms on GPUs," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 1, pp. 91 –104, jan. 2011.

[25] M. Reiser and M. VandenBerg, "Validity of the chi-square test in dichotomous variable factor analysis when expected frequencies are small," *British Journal of Mathematical and Statistical Psychology*, vol. 47, no. 1, pp. 85–107, 1994.

[26] E. Sandra, L. and I. Sheila, M., "Bakery foods are the major dietary source of trans-fatty acids among pregnant women with diets providing 30 percent energy from fat," *Journal of the American Dietetic Association*, vol. 102, no. 1, pp. 46 – 51, 2002.

[27] H. Shin, Y. Jung, C. Jeong, and J.-H. Heo, "Assessment of modified AndersonDarling test statistics for the generalized extreme value and generalized logistic distributions," *Stochastic Environmental Research and Risk Assessment*, vol. 26, pp. 105–114, 2012.

[28] K. Takahasi, "Note on the multivariate burrs distribution," *Annals of the Institute of Statistical Mathematics*, vol. 17, pp. 257–260, 1965.

[29] D. Torreggiani and G. Bertolo, "Osmotic pre-treatments in fruit processing: chemical, physical and structural effects," *Journal of Food Engineering*, vol. 49, no. 2-3, pp. 247 – 253, 2001.

[30] J. Weaver, Y. Xu, D. Healy, and L. Cromwell, "Filtering noise from images

with wavelet transforms," *Magnetic Resonance in Medicine*, vol. 21, no. 2, pp. 288–295, 2005.

[31] Q. Ying-Dong, C. Cheng-Song, C. San-Ben, and L. Jin-Quan, "A fast subpixel edge detection method using Sobel-Zernike moments operator," *Image and Vision Computing*, vol. 23, no. 1, pp. 11 – 17, 2005.