CIRCLE DETECTION IN IMAGES

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Electrical Engineering

by

Prajwal Shetty

Summer 2011

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Prajwal Shetty:

Detection of Circles in Images

[illegible]

Jay Harris, Chair
Department of Electrical and Computer Engineering

[illegible]

Fred Harris
Department of Electrical and Computer Engineering

C. Paoli

Chris Paolini
Department of Computer Science

4/1/11

Approval Date

ABSTRACT OF THE THESIS

Detection of Circles in Images
by
Prajwal Shetty
Master of Science in Electrical Engineering
San Diego State University, 2011

Detection of circles in basic gray scale images is explored in this thesis. Impetus for the work derives from an interest in high speed detection of tires on cars. The circle detection process used is based on the pixel direction concept of J. F. Canny and the non-maximum suppression process for contour tracing described by Neubeck and Gool. In the detection process, edge segments are evaluated to determine if they are part of a candidate circle and the center and radius of the circle is computed. The edge threshold property of Canny edge detection is not used.

The algorithm has been tested on a group of 30 images of cars using Matlab. The image set includes identical mages having different image intensities. Edge images of tires are compared with images obtained using Canny edge detection. Useful results for detecting circles in images are found.

VHDL implementation of the extraction algorithm is discussed. Synthesis results indicate that the design code can perform evaluations at a 30 frame per second rate. Memory usage is small compared to that required for the circular Hough transform. The time taken to determine the center and radius of a circle, and memory usage by the proposed algorithm, are discussed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PAGE

ACKNOWLEDGEMENTS

I take this opportunity to sincerely thank my thesis advisor, Professor Jay H Harris, for the guidance, unrelenting support, and backing me during every step of my research work. Without his help it would not have been possible for me to accomplish this feat.

I would also like to thank Professor fred harris and Professor Christopher Paolini for being the readers of this thesis and for giving their valuable suggestions.

A special thanks to my incredible Albert’s family for their support and thoughtful discussion.

I would also like to thank the Electrical and Computer Engineering Department, Chair and Graduate Advisor.

CHAPTER 1

INTRODUCTION

Extracting the edges of objects and identifying basic shapes in an image is of importance in industrial applications of image processing. This thesis deals with detection of circles in the images. Extracting the edges from an image simplifies the analysis of the images by dramatically reducing the amount of data to be processed, while at the same time preserving useful information about the boundaries. Many edge detectors have been proposed, such as Canny [1], Robert [2], Prewitt [3] and Driche [4]. Of these, the Canny edge detector has been widely successful in extracting the edge feature from an image, as discussed in detail in Chapter 2. The edge extracted image is used by many algorithms, such as the Circular Hough Transform [5] and the Linear Square method [6], to further extract shape information like straight lines, ellipses, circles, etc. The most widely used circle detection algorithm is the Circular Hough Transform, which is discussed in Chapter 3.

In this thesis, the Canny edge detector is modified to detect circles in an image. This is discussed in Chapter 4. The proposed algorithm is implemented in VHDL. The practicality of implementing this algorithm is discussed in Chapter 5. We discuss the outcome of Canny edge detection algorithms in Chapter 6 and also show how the proposed algorithm is less sensitive to intensity changes than Canny when extracting circles from images with different intensities. Calculation of the time taken to detect circles in an image in a worst-case scenario and also calculation of the size of memory being used is discussed in Chapters 5 and 6, respectively. The next chapter provides a review of the Canny edge detector, which is modified to obtain the algorithm developed in this thesis.

CHAPTER 2

CANNY EDGE DETECTION

Canny edge detection is one of the basic algorithms used in shape recognition. The algorithm uses a multi-stage process to detect a wide range of edges in images. The stages, as enumerated by Rao and Venkatesan [7: 843] are:

a. Image smoothing. This is done to reduce the noise in the image.
b. Calculating edge strength and edge direction.
c. Directional non-maximum suppression to obtain thin edges across the image.
d. Invoking threshold with hysteresis [1] to obtain only the valid edges in an image.

A block diagram of the Canny edge detection algorithm is shown in Figure 2.1. The input to the detector can be either a color image or a grayscale image. The output is an image containing only those edges that have been detected.

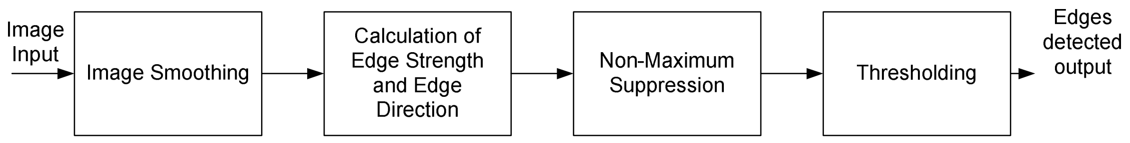


Figure 2.1. Block diagram of the stages of the Canny edge detector.

2.1 IMAGE SMOOTHING

Image smoothing is the first stage of the Canny edge detection. The pixel values of the input image are convolved with predefined operators to create an intermediate image. This process is used to reduce the noise within an image or to produce a less pixilated image. Image smoothing is performed by convolving the input image with a Gaussian filter [8]. A Gaussian filter is a discrete version of the 2-dimensional function shown in equation (2.1).

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right) \tag{2.1}$$

In (2.1), σ is the standard deviation of the Gaussian filter, which describes the narrowness of the peaked function, and x and y are spatial coordinates.

An example of the conversion of (2.1) to a 2-dimensional 5x5 discrete Gaussian filter function is shown in equation (2.2). The function is obtained for σ = 1.4 by substituting integer values for x and y and renormalizing [8].

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}. \tag{2.2}$$

In (2.2), A is the 2-dimensional array of input pixel values and B is an output image. The process of smoothing an image using an MxM block filter yields an image in which each pixel value is a weighted average of the surrounding pixel values, with the pixel value at the center weighted much more strongly than those at nearby points.

Image smoothing is done primarily to suppress noise and to get a continuous edge contour during the non-maximum suppression process. The output thus obtained is a blurred intermediate image. This blurred image is used by the next block to calculate the strength and direction of the edges.

2.2 Calculation of the Strength and Direction of Edges

In this stage, the blurred image obtained from the image smoothing stage is convolved with a 3x3 Sobel operator [9]. The Sobel operator is a discrete differential operator that generates a gradient image. Horizontal and vertical Sobel operators that are used to calculate the horizontal and vertical gradients, respectively, are shown in equations (2.3a) and (2.3b).

$$gx = \begin{array}{|c|c|c|} \hline -1 & 0 & +1 \\ \hline -2 & 0 & +2 \\ \hline -1 & 0 & +1 \\ \hline \end{array} \tag{2.3a}$$

$$gy = \begin{array}{|c|c|c|} \hline +1 & +2 & +1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} \qquad (2.3b)$$

2.2.1 Calculating Edge Strength

The Sobel operators from equation (2.3) are used to obtain a gradient image. To obtain the gradient image, a smoothened image from the first stage is convolved with the horizontal and vertical Sobel operators as shown in equations (2.4a) and (2.4b), respectively.

$$Gx = (I*gx) \qquad (2.4a)$$

$$Gy = (I*gy) \qquad (2.4b)$$

In (2.4) I is the image obtained after Image smoothing; Gx and Gy are images with pixel values that are the magnitude of the horizontal and vertical gradient, respectively. Images Gx and Gy from equations (2.4a) and (2.4b) are used in equation (2.5) to obtain the “edge strength” of a pixel in an image. The edge strength G is

$$G = \sqrt{}(Gx\text{^}2 + Gy\text{^}2) \qquad (2.5)$$

2.2.2 Calculating Edge Direction

“Edge direction is defined as the direction of the tangent to the contour that the edge defines in 2-dimensions” [1: 690]. The edge direction of each pixel in an edge direction image is determined using the arctangent

$$A = \arctan\ (Gy/Gx) \qquad (2.6)$$

The edge strength for each pixel in an image obtained from equation (2.5) is used in non-maximum suppression stage, which is discussed in section 2.1.3. The edge directions obtained from equation (2.6) are rounded off to one of four angles--0 degree, 45 degree, 90 degree or 135 degree--before using it in non-maximum suppression, as shown in Figure 2.2.

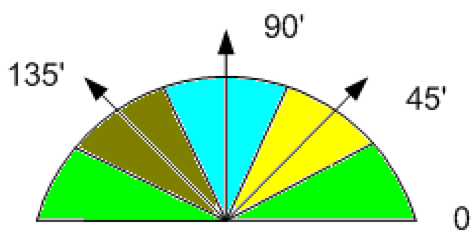


Figure 2.2. Edge directions are rounded off to one of the four angles to be used in non-maximum suppression.

Example for calculating Edge Strength and Edge Direction:

Consider the block of an image I .

I = (2.7)

100	120	100
70	100 (x,y)	90
100	90	80

Then from (4a), (4b), (5) and (6) Gx, Gy, G and A are found as follows:

a. Gx = | I ∗gx |.

= | (100*-1) + (70*-2) + (100*-1) + (100*1) + (90*2) + (80*1) |

= 20.

b. Gy = |I∗gy|.

= | (100*-1) + (120*-2) + (100*-1) + (100*1) + (90*2) + (80*1)|

= 80.

c. G = √(80^2 + 20^2)

= √(6400 + 400).

= 82.46.

d. A = arctan (Gy/Gx)

= arctan (80/20)

= arctan (4)

=75.96. (Rounded off to 90 degree for Non Maximal suppression)

The edge direction and the edge strength calculated are used to obtain the image with thin edges, as discussed in the next section.

2.3 NON-MAXIMUM SUPPRESSION

Non-maximum suppression (NMS) [1] is used normally in edge detection algorithms. It is a process in which all pixels whose edge strength is not maximal are marked as zero within a certain local neighborhood. This local neighborhood can be a linear window at different directions [10] of length 5 pixels. The linear window considered is in accordance with the edge direction of the pixel under consideration [10] for a block in an image as shown in Figure 2.3.

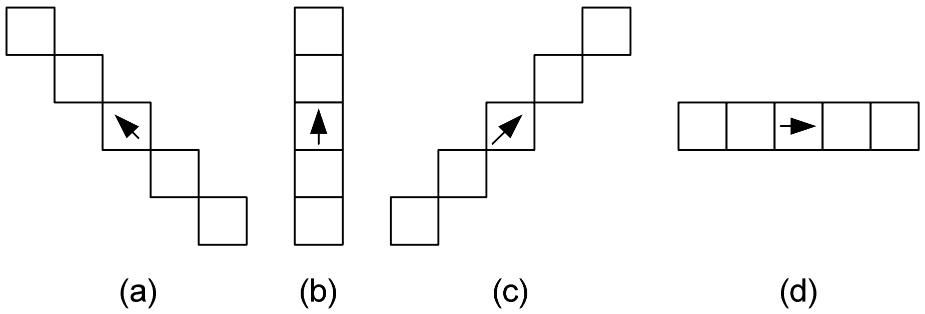


Figure 2.3. Linear window at the angle of (a) 135° (b) 90° (c) 45° (d) 0°.

For example, if the edge direction of the center pixel is 135° then the linear window at an angle of 135° is used, as shown in Figure 2.4.

The non-maximum suppression is used to suppress all the image information which is not the local maxima. For each pixel in the linear window, edge strength of the neighbors is compared, and if the pixels are not part of the local maxima then they are set to zero. Only the local maximum is considered. The result of non-maximum suppression is shown in Figure 2.5.

A dynamic block [11] of linear window of size 5 pixel is used to scan the image to obtain the thin edges throughout the image. The thin edges found are the result of non-maximum suppression.

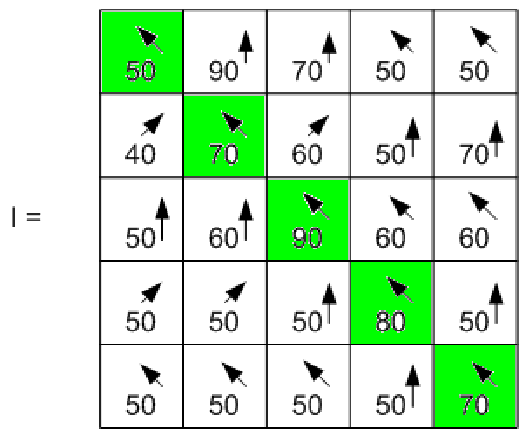


Figure 2.4. Image with edge strength and edge direction.

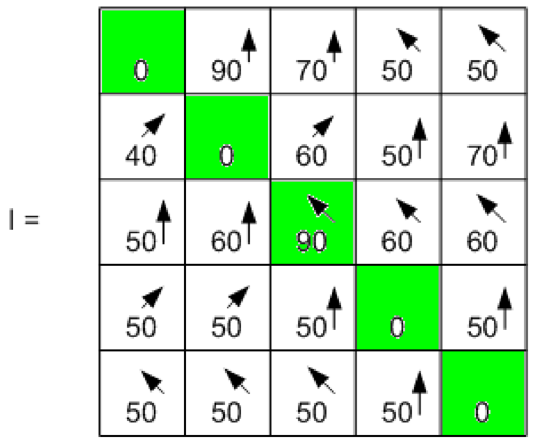


Figure 2.5. Image after non-maximum suppression.

Results of non-maximum suppression can be different for different kinds of edges. According to [12], usually, there are three types of edges, one of which is the Roof type edge, which has a slow change, whether increase or decrease. The non-maximum suppression process could thin the wider roof ridge [2] to the width of one pixel and eliminate false edge points [12]. The resulting image with thin edges is used as input to the fourth stage, i.e., thresholding with hysteresis, which is explained in section 2.1.4.

2.4 Thresholding with Hysteresis

Thresholding with hysteresis is the last stage in Canny edge detection, which is used to eliminate spurious points and non-edge pixels from the results of non-maximum suppression. The input image for thresholding with hysteresis has gone through Image smoothing, Calculating edge strength and edge pixel, and the Non-maximum suppression stage to obtain thin edges in the image. Results of this stage should give us only the valid edges in the given image, which is performed by using the two threshold values, T1 (high) and T2 (low), for the edge strength of the pixel of the image. Edge strength which is greater than T1 is considered as a definite edge. Edge strength which is less than T2 is set to zero. The pixel with edge strength between the thresholds T1 and T2 is considered only if there is a path from this pixel to a pixel with edge strength above T1. The path must be entirely through pixels with edge strength of at least T2. This process reduces the probability of streaking. As the edge strength is dependent on the intensity of the pixel of the image, thresholds T1 and T2 also depend on the intensity of the pixel of the image. Hence, the thresholds T1 and T2 are calculated by the Canny edge detectors using adaptive algorithms, which are approximations as defined in [13].

Thus all the edges in an image are detected using the Canny edge detection.

2.5 ADVANTAGES OF EDGE DETECTION

Edge detection forms a pre-processing stage to remove the redundant information from the input image, thus dramatically reducing the amount of data to be processed while at the same time preserving useful information about the boundaries. Thus edge detection provides basic information which is used for extracting shapes like lines, circles and ellipses by techniques such as Hough Transform. In Chapter 3 we will discuss how the edge detected image is used by the Circular Hough Transform to detect the circles in an image.

CHAPTER 3

EXTRACTION OF CIRCLES

Detecting and recognizing the shapes in an image is extremely important in industrial applications in recognizing the object. Detecting circles in an image is one of the problems that is discussed in this thesis. Many algorithms, such as Linear Square Method [6] and Hough Transform, have been proposed to detect circles. These algorithms detect circles from the edge detected images. Among these algorithms, Circular Hough Transform has been widely successful in meeting the real time requirement of being able to detect the circles in noisy environments. In this chapter, we will be discussing how Circular Hough Transform is used to detect the circles in an image.

3.1 CIRCULAR HOUGH TRANSFORM

One of the most commonly used algorithms to recognize different shapes in an image is Hough Transform [13]. Hough Transform was introduced by Paul Hough in 1962 and patented by IBM. In 1972 Richard Duda and Peter Hart modified Hough Transform, which is used universally today under the name Generalized Hough Transform [14]. An extended form of General Hough Transform, Circular Hough Transform (CHT) [13], is used to detect circles, which is discussed in section 3.1.1.

The edge detected from the Canny edge detector forms the input to extract the circle using the Circular Hough Transform.

In Circular Hough Transform, voting procedure is carried out in a parameter space. The local maxima in accumulator space, obtained by voting procedure, are used to compute the Hough Transform. Parameter space is defined by the parametric representation used to describe circles in the picture plane, which is given by equation (3.1). An accumulator is an array used to detect the existence of the circle in the Circular Hough Transform. Dimension of the accumulator is equal to the unknown parameters of the circle.

The equation of the circle in parametric form is given by

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{3.1}$$

Equation (3.1) implies that the accumulator space is three-dimensional (for three unknown parameters x0, y0 and r).

This equation (3.1) defines a locus of points (x,y) centered on an origin (x0,y0) with radius r. Each edge point in Figure 3.1a defines a set of circles in the accumulator space [13]. These circles are defined by all possible values of the radius and they are centered on the coordinates of the edge point. Figure 3.1b shows three circles defined by three edge points labeled 1, 2 and 3. These circles are defined for a given radius value. Each edge point defines circles for the other values of the radius, also. These edge points map to a cone of votes in the accumulator space. Figure 3.1c illustrates this accumulator.

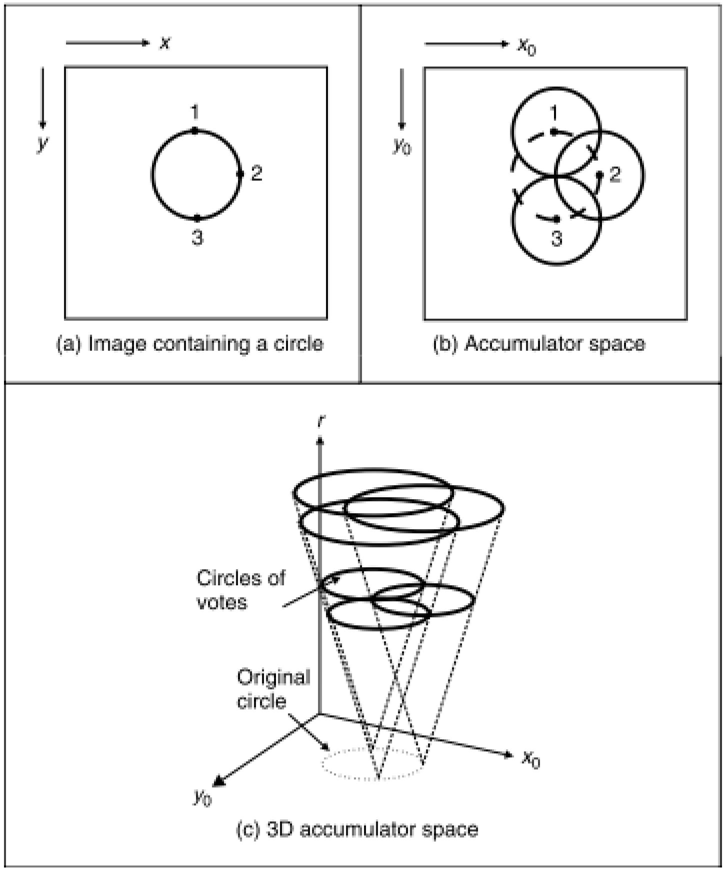


Figure 3.1. Illustration of circular hough transform.

Points corresponding to x0, y0 and r, which has more votes, are considered to be a circle with center (x0, y0) and radius r.

3.2 PRACTICAL PROBLEMS WITH HOUGH TRANSFORM

Some of the disadvantages of Hough Transform are [15]:

Quantization errors: An appropriate size of accumulator array is difficult to choose.

Difficulties with noise: The advantage of Hough Transform is that it connects edge points across the image to some form of parametric curve. However, this is also its weakness, because it is possible that good phantom circles might also be detected with a reasonably large voting array.

Time taken: A high dimensional parameter space can slow the Hough Transform process.

Memory: As the accumulator is used in random-accessed fashion, it can easily overrun the available memory.

CHAPTER 4

THREE STAGE PROCESS FOR DETECTING LOCATION OF CIRCLE

In this chapter, we discuss a three stage process for detecting the position and radius of the circles. The algorithm that incorporates the process involves some of the concepts from the Canny edge detection. We consider the input image to be a noise free grayscale image. The noise free image considered for input does not have any random variations in intensity. In the next section, we will discuss the block diagram of the proposed algorithm, which is a three stage process for detecting the location of circles.

4.1 BLOCK DIAGRAM

The block diagram is shown in Figure 4.1. The first stage of the process is convolution and non-maximum suppression. Input to this block is a noise free grayscale image. In this block horizontal and vertical convolution kernels are used to obtain horizontal and vertical directions for each pixel. These values are rounded to one of the eight directions for each pixel. The vertical and horizontal convolution kernels are also used to get the edge strength for the image. The pixel direction and edge strength obtained are used as input to non-maximum suppression to get the thin edges defined, as discussed in section 4.1.1.

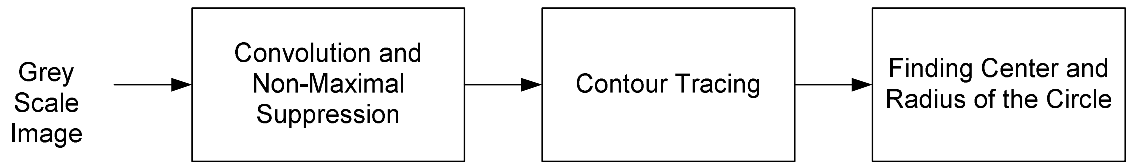

Grey Scale Image
Convolution and Non-Maximal Suppression
Contour Tracing
Finding Center and Radius of the Circle

Figure 4.1. Three stages of the algorithm.

In the second stage, the thin edges determined from the first stage are contour traced using the pixel direction of each pixel to find the arcs, which satisfies the conditions of being part of a circle. The spurious points and the edge pixels which do not satisfy the conditions of being part of a circle are discarded in this stage. Hence, the arcs which have the higher probability of being part of the circle are retained.

In the third stage, these arcs are checked for their probability of being part of the circle and then the center and radius of the circle are found. Hence, the circle is detected and located.

4.2 First Stage: Convolution and Non-Maximal Suppression

Block diagram of Convolution and Non-Maximal Suppression is as shown in Figure 4.2.

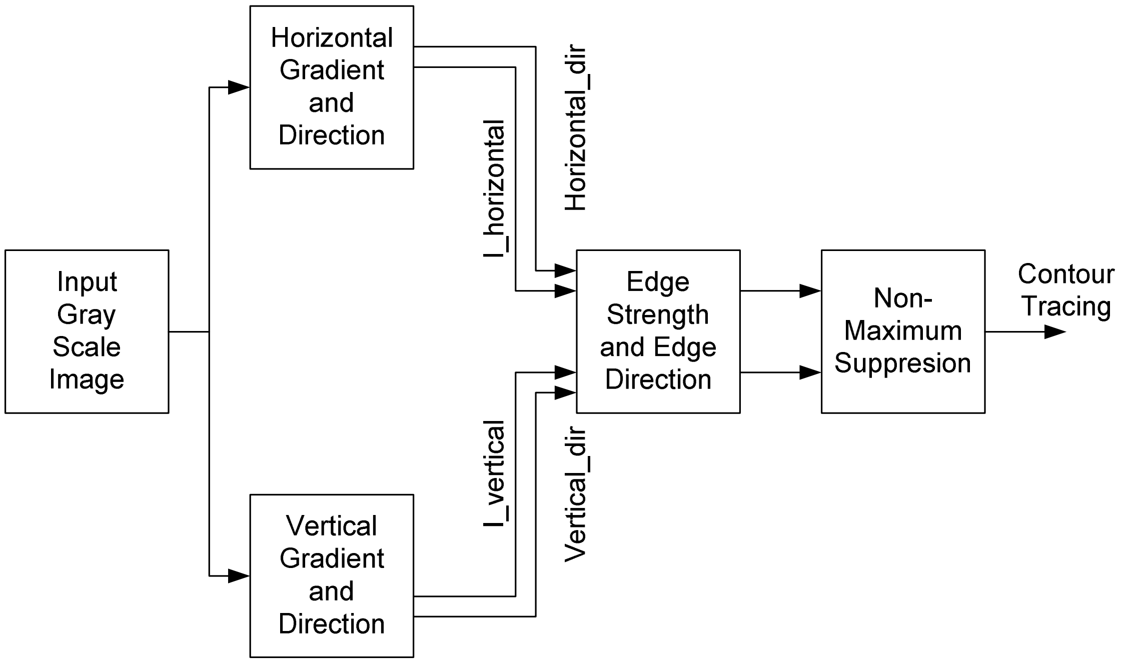


Figure 4.2. Block diagram of convolution and non-maximum suppression.

The goal of this algorithm is to locate the circles in an image more quickly and with efficient use of resources by finding the edges. The edges in an image are expected to be the boundaries of objects that tend to produce sudden changes in the image intensity [9]. For example, different objects in an image usually have different colors, hues or light intensity and this causes a change in image intensity. Thus, much of the geometric information that would be conveyed in a line drawing is captured by the intensity changes in an image [9]. The input to this algorithm is assumed to be a noise free grayscale image. Hence, there is no very high random variation in intensity from one pixel to another.

How edges are detected by detecting the change in the intensity is shown in Figures 4.3a and 4.3b. Consider that we have a signal [1] with a one dimensional edge shown by the jump in intensity, as shown in the Figure 4.3a.

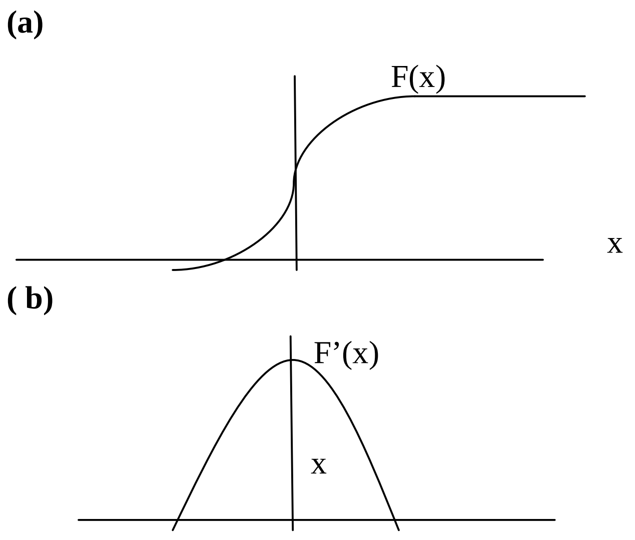


Figure 4.3. (a) One-dimensional change in intensity (b) Derivative of the change in intensity.

Using the convolution kernel on this signal, i.e., the derivative of this signal, we get the following signal in Figure 4.3b.

As shown in Figure 4.3b, the derivative has the maximum located at the center of the edge of the original signal [1]. Based on one dimensional analysis, the theory can be extended to two dimensions. For a grayscale digital image, four 3x1 window operators are used to get two pixel directions, Horizontal (Kx) and Vertical (Ky), for each pixel, taking into account the neighbors of the pixel.

The kernels used to find Horizontal gradient orientation are

$$Kx1= \begin{array}{|c|c|} \hline 1 & \\ \hline 2 & X \\ \hline 1 & \\ \hline \end{array} \qquad 4.1a$$

$$Kx2= \begin{array}{cc} & 1 \\ X & 2 \\ & 1 \end{array} \tag{4.1b}$$

where X is the pixel under consideration. In Kx1, the value of the center pixel of the column is 2, whereas the remaining values in the column are 1. Similarly, in Kx2, the value of center pixel of the column is 2, whereas the remaining values in the column are 1. This is to place more emphasis on the values of the horizontal neighbor pixels [8] compared to other pixels. Similarly, kernels used to find vertical gradient orientation are

$$Ky1= \begin{array}{ccc} 1 & 2 & 1 \\ & X & \end{array} \tag{4.2a}$$

$$Ky2= \begin{array}{ccc} & X & \\ 1 & 2 & 1 \end{array} \tag{4.2b}$$

In equation 4.2a and 4.2b, X is the pixel under consideration. In Ky1, the value of the center pixel of the row is 2, whereas remaining values in the row are 1. Similarly, in Ky2, the value of center pixel of the row is 2, whereas the remaining values in the row are 1. This is to place more emphasis on the values of the vertical neighbor pixels compared to other pixels. Kx1, Kx2, Ky1 and Ky2 kernels are used to get the horizontal and vertical pixel direction and also pixel edge strength.

4.2.1 Determination of Pixel Direction and Edge Strength

In this section, we will discuss how the pixel direction and edge strength are calculated for a pixel. To calculate pixel direction we first have to determine the horizontal and vertical orientation, and to calculate edge strength, horizontal orientation and vertical gradient magnitude are calculated. Horizontal orientation and horizontal gradient magnitude are determined as follows.

Kernel Kx1 from equation (4.1a) and Image I is convolved to get a set of values Ix1, corresponding to the image points as given by the equation (4.3a). Similarly, kernel Kx2 from equation (4.1b) and Image I is convolved to get a set of values Ix2, corresponding to the image points as given by the equation (4.3b). These two sets of values, Ix1 and Ix2, determine the horizontal orientation of a pixel point (x,y) of the image I. The horizontal orientation is determined as follows:

$$Ix1 = I*Kx1 \tag{4.3a}$$

$$Ix2 = I*Kx2 \tag{4.3b}$$

If Ix1(x,y) > Ix2(x,y) then the orientation I_h_ore is $\rightarrow$ naming it as 1.
If Ix2(x,y) > Ix1(x,y) then the orientation I_h_ore is $\leftarrow$ naming it as 2.
If Ix2(x,y) = Ix1(x,y) then the orientation is 0.

$$I\_hor(x,y) = |Ix2(x,y) - Ix1(x,y)| \tag{4.4}$$

In (4.4) I_hor(x,y) is the horizontal gradient magnitude of the point I(x,y). Thus two pieces of needed information--I_hor(x,y), horizontal gradient magnitude, and I_h_ore(x,y), horizontal orientation--are obtained for a pixel (x,y).

The vertical orientation and vertical gradient magnitude is determined as follows. Kernel Ky1 and Image I and Ky1 and Image I are convolved to get a set of values, Iy1 and Iy2 respectively, corresponding to the image points as given by the equations (4.5a) and (4.5b). These two sets of values determine the vertical orientation of the pixel points of the image I. The vertical orientations are determined as follows:

$$Iy1 = I*Ky1 \tag{4.5a}$$

$$Iy2 = I*Ky2 \tag{4.5b}$$

If Iy1(x,y) > Iy2(x,y) then the orientation I_v_ore is $\downarrow$ naming it as 2.
If Iy2 (x,y) > Iy1(x,y) then the orientation I_v_ore is $\uparrow$ naming it as 1.
If Iy2(x,y) = Iy1(x,y) then the orientation is 0.

$$I\_ver(x,y) = |Iy2(x,y) - Iy1(x,y)| \tag{4.6}$$

In (14), I_ver(x,y) is the vertical gradient magnitude of the point I(x,y). Thus four needed pieces of information--I_hor(x,y), horizontal gradient magnitude, I_ver(x,y), vertical gradient magnitude, I_h_ore(x,y), horizontal orientation and I_v_ore(x,y), vertical orientation--are obtained for a pixel (x,y). Thus by knowing horizontal and vertical gradient magnitude, the edge strength is calculated by the equation (15):

$$I\_mag = |I\_hor + I\_ver| \tag{4.7}$$

The vertical and horizontal orientation for each point in the image is used to determine the pixel orientation. The pre-defined conditions used to get the pixel direction are as shown in Table 4.1.

Table 4.1. Conditions Used to Determine the Pixel Direction

No.	Vertical Orientation Iy	Horizontal Orientation Ix	Conditions	Range	Result
1	↑ or ↓	→	I_hor > 4*I_ver	346' - 14'	→ 1
2	↑	→	I_hor < 4*I_ver I_ver < 4*I_hor	14' - 76'	↗ 2
3	↑	← or →	I_ver > 4*I_hor	76' - 104'	↑ 3
4	↑	←	I_hor < 4*I_ver I_ver < 4*I_hor	94' - 166'	↖ 4
5	↑ or ↓	←	I_hor > 4*I_ver	166' - 184'	← 5
6	↓	←	I_hor < 4*I_ver I_ver < 4*I_hor	184' - 256'	↙ 6
7	↓	← or →	I_ver > 4*I_hor	256' - 284'	↓ 7
8	↓	→	I_hor < 4*I_ver I_ver < 4*I_hor	284' - 346'	↘ 8

An example of using the table to determine the direction of a pixel using the vertical and horizontal orientation is as follows. If Ix1 > Ix2 for a pixel, then the horizontal orientation is "→", as discussed earlier in this section, which is rows 1, 2 and 8 in the table. If

Iy1 > Iy2 for the same pixel then vertical orientation is "↓", which is rows 6, 7 and 8 in the table. But as the horizontal and vertical directions are for the same pixel, row 8 satisfies both the conditions for horizontal and vertical direction. Hence, the pixel direction for the pixel under consideration is 8.

This process is repeated for all the pixels in the image, which gives us the pixel direction for every pixel (including edges and non-edges) in the image. Range of angles for each pixel direction is as shown in Figure 4.4.

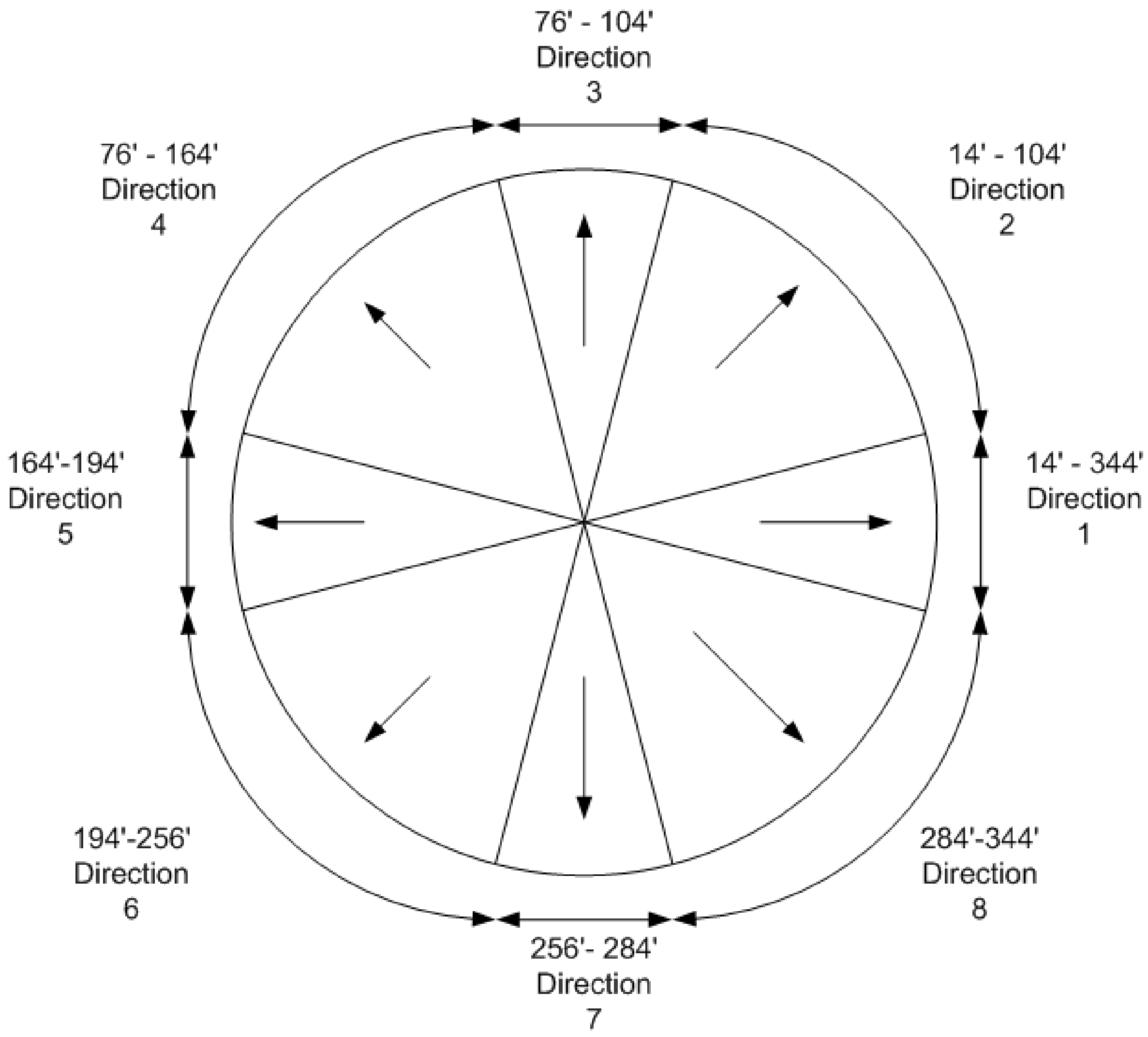
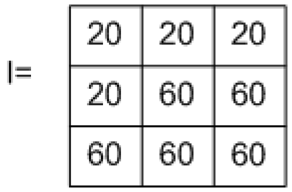


Figure 4.4. Range of angles of pixel direction.

Example of calculating pixel direction and edge strength is given below.

Let us consider a 3x3 block of an image I (Figure 4.5).



I=

20	20	20
20	60	60
60	60	60

Figure 4.5. 3x3 block of an image I.

From equations (4.3a) and (4.3b) we have

As Ix2 > Ix1, the horizontal orientation, I_h_ore is 2, i.e., $\longleftarrow$ direction from Table 4.1 (p. 17).

From equation (4.4), horizontal gradient magnitude is calculated as

I_hor = |Ix2 – Ix1| = 120.

Similarly, from equations (4.5a) and (4.5b) we have

$$\begin{aligned} Ix1 &= (I * Kx1) \\ &= (1*20) + (2*20) + (1*60) \\ &= 120. \\ Ix2 &= (I*Kx2) \\ &= (1*60) + (2*60) + (1*60) \\ &= 240. \end{aligned}$$

As Iy2 > Iy1, the vertical orientation, I_v_ore is 1 i.e.,$\longleftarrow$ direction from Table 4.1 (p. 17). From equation (4.6), vertical gradient magnitude is given by

I_ver = |Iy2 – Iy1| = 160.

From equation (15), edge strength is given by

$$\begin{aligned} I\_mag &= |I\_hor + I\_ver| \\ &= |120 + 160| \\ &= 280. \end{aligned}$$

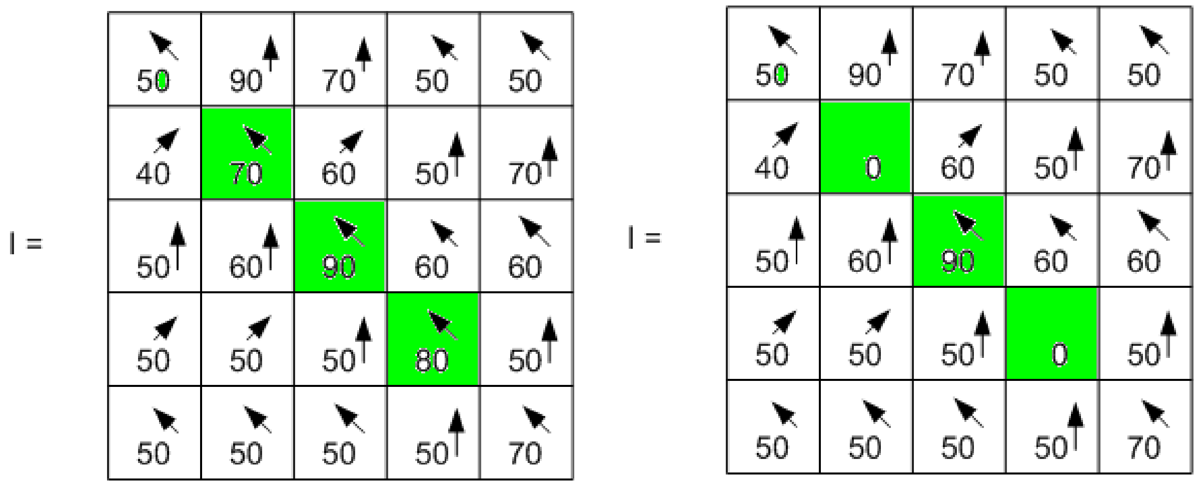
To determine the pixel direction, the horizontal and vertical orientations determined are I_h_ore, $\longleftarrow$ and I_v_ore,$\uparrow$. From the table, row 4 matches these orientations. Hence, the pixel direction is 4.

The pixel direction and edge strength obtained are used by non-maximum suppression in section 4.2.2 to determine thin edges for an object.

4.2.2 Proposed Non-Maximum Suppression

In this stage, referred to as non-maximum suppression (NMS) [16], a set of thin edge points, in the form of a binary image, is obtained. These are referred to as "thin edges", as after this process edge width is only 1 pixel.

To execute NMS we scan the convolved image, with edge strength and pixel direction obtained from section 4.2.1, for pixel directions. When we find any pixel with pixel direction other than 0 we look for other pixels in the same area having the same pixel direction. A linear window, as discussed in section 2.1.3, having immediate neighbors of pixels under consideration, is used. Hence the size of the linear window is 3 pixels. The pixel having the highest edge strength within the linear window is considered and rest of the pixels in the linear window are made zero. An example of NMS used in the proposed algorithm is shown in Figure 4.6.



(a) (b)

Figure 4.6. (a) Image with linear window in color corresponding to the pixel direction and gradient magnitude (b) Result of non-maximum suppression.

If we observe the Figure 4.6(a), the linear window is displayed in color. Notice the pixel direction and the linear window are both 135°. In NMS, we retain only the pixel which has highest edge strength within the linear window and convert the rest of the pixels in the linear window to zero. Hence, in Figure 4.6b pixels with edge strength 90 are retained and edge strength and pixel direction of rest of the pixels in linear window are made zero. From this operation we thereby get thin edges of the objects in an image. These thin edges found can form a boundary of a circular object in the image. In the next section, we use these thin edges found to find the arcs which can be part of a potential circle.

4.3 CONTOUR TRACING

In this section, we will focus primarily on finding and validating the segments or arcs of a potential circle. We also discuss connecting the arcs which may fall on the same circle to finally calculate the radius and center of the circle and confirm that these arcs form a circle. In the process, we also eliminate edge pixels of non-circular objects and also eliminate isolated edge pixels.

As a result of NMS we have thin edges for all the possible object boundaries in the image. These thin edges are contour traced to retain only the potential circle’s edge pixels and discard the rest of the edge points, even though these edges are part of an object. It is very important that we eliminate the edge pixels which are not part of the circle. At the same time, it is important to retain all the information required to track the circle.

Some methods have been researched for contour tracing [17]. The key problem here is to set criteria according to angles and relative positions of line segments [18]. One of the well known algorithms for contour tracking is Chain code [17]. Chain code is discussed in the following section to get a better understanding of how the contour tracing works. The contour tracing technique used in the proposed algorithm is discussed in section 4.3.2.

4.3.1 Chain Code Algorithm for Contour Tracking

Chain code is a list of codes ranging from 0 to 7 in clockwise direction [17]. These codes represent the direction of the next pixel connected in 3x3 windows, as shown in the Table 4.2. For example, if a current pixel in an image is located at coordinate (5,5), the coordinate of the next pixel based on the chain code is given by Table 4.2. The coordinates of the next pixel are calculated based on addition and subtraction of column and row by 1, depending on the value of the chain code, as shown in Table 4.3.

The disadvantage here is that we have to scan all the eight neighboring pixels while contour tracing. In the proposed algorithm we scan only three neighboring pixels.

Table 4.2. Relation of Pixel and Chain Code with Current Pixel

	Column-1	Column	Column+1
Row-1	5	6	7
Row	4	Current Pixel	8
Row+1	3	2	1

Table 4.3. Coordinates of Next Pixel Calculated Based on the Chain Code for Current Pixel (5,5)

Code	Next Row	Next Column
0	5	5
1	6	6
2	6	5
3	6	4
4	5	4
5	4	4
6	4	5
7	4	6

4.3.2 Contour Tracing to Find Arcs

In this section, we discuss how contour tracing is done to find the arcs that can be part of a potential circle. There are two principles to track the edges which form the boundary of an object: one based on edge strength, and the other based on pixel direction [9]. In this proposed algorithm we use the property based on pixel direction. This property states, "*An edge pixel at (x0,y0) in the predefined neighborhood of (x,y) has an angle similar to the pixel at (x,y) if*

$$|angle(x,y) - angle(x0,y0)| < A \tag{4.8}$$

Where A is a nonnegative angle threshold." [9: 586]. From this principle we can derive that pixel direction of an arc belonging to a circle will be almost the same, as the pixel directions used in this algorithm are obtained by rounding off edge direction to one of the eight directions, as discussed in section 4.1. Hence, in this proposed algorithm while contour tracing we look for neighboring cells with same pixel direction to find a line segment. Here, we take advantage of knowing the pixel direction of each pixel in the image. Let us discuss how knowing the pixel direction can reduce the calculations and increase the speed.

The image is scanned to find a pixel with any pixel direction. For example, let us consider that the pixel direction we are looking for is located at (x,y) with the pixel direction 2, as shown in Figure 4.7. We know that pixels (x-1, y), (x-1, y-1), (x, y-1) and (x+1, y-1) have already been scanned, as these pixels come before the pixel (x,y). In rest of the neighboring pixels, pixel (x-1,y+1) is discarded as, if this pixel had pixel direction 2 then this would have been made zero during the NMS stage. Hence, remaining neighbors are (x+1,y), (x+1,y+1) and (x,y+1), as shown in Figure 4.7. In this case, as the pixel direction of pixel (x,y) is 2, the probability of (x+1,y+1) having the gradient direction 2 is higher compared to other neighboring pixels [8], as specified in Figure 4.7. Hence, the first priority is given to (x+1, y+1) pixel while contour tracing. Both (x+1,y) and (x,y+1) have equal probability, hence it will not matter which is given higher priority. Here we are going to give pixel (x, y+1) second priority and pixel (x+1,y) the third priority.

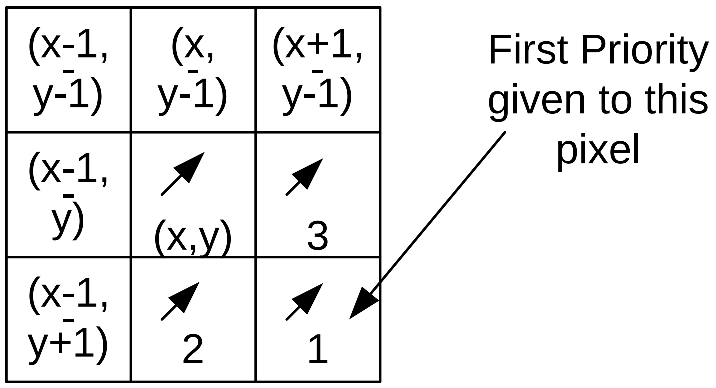


Figure 4.7. Representing the priority of the pixels to be scanned based on the pixel direction 2.

Hence, when (x,y) pixel is found having pixel direction 2, we first check if pixel (x+1,y+1) has the same pixel direction. If yes, then we update x to x+1 and y to y+1 and repeat the same process till we find neighboring pixels with a pixel direction other than 2. We consider this arc only if the arc length is at least 10 pixels; otherwise the arc is ignored and the scan for pixels with direction 2 is continued.

Let us consider one more example of Contour tracing the arc with pixel direction 6. If (x,y) is the pixel with pixel direction 6, then highest priority is given to pixel (x-1,y-1) , second priority is given to (x-1,y) and third priority is given to (x,y-1). Hence, when (x,y) pixel is found having pixel direction 6, we first check the pixel (x-1,y-1) then pixel (x-1,y) and then pixel (x,y-1) for the pixel direction 6. If any of these pixels have pixel direction 6 then x and y values are updated to the pixel which has pixel direction 6. This process is continued to the point there are no neighboring pixels with pixel direction 6. This is explained in Figure 4.8.

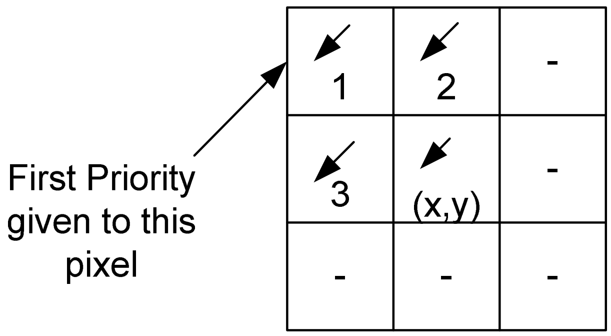


Figure 4.8. Representing the priority of the pixels to be scanned based on the pixel direction 6.

Similarly, contour tracing for pixels with direction 1, 3, 4, 5, 7 and 8 is shown in Figure 4.9.

Thus, arcs having same edge directions and pixel lengths of at least 10 pixels are found throughout the image.

4.3.3 Condition for Arc to be Part of a Potential Circle

In this section, we use a cluster of connected and related edge pixels meeting certain criteria to calculate the center of the potential circle that the pixel cluster falls on. It is possible to detect multiple circular objects in an image. To understand how the arcs are related to each other in a circle, let us consider a circle in ideal conditions. In ideal conditions, after finding the pixel direction, applying non-maximal suppression and finding

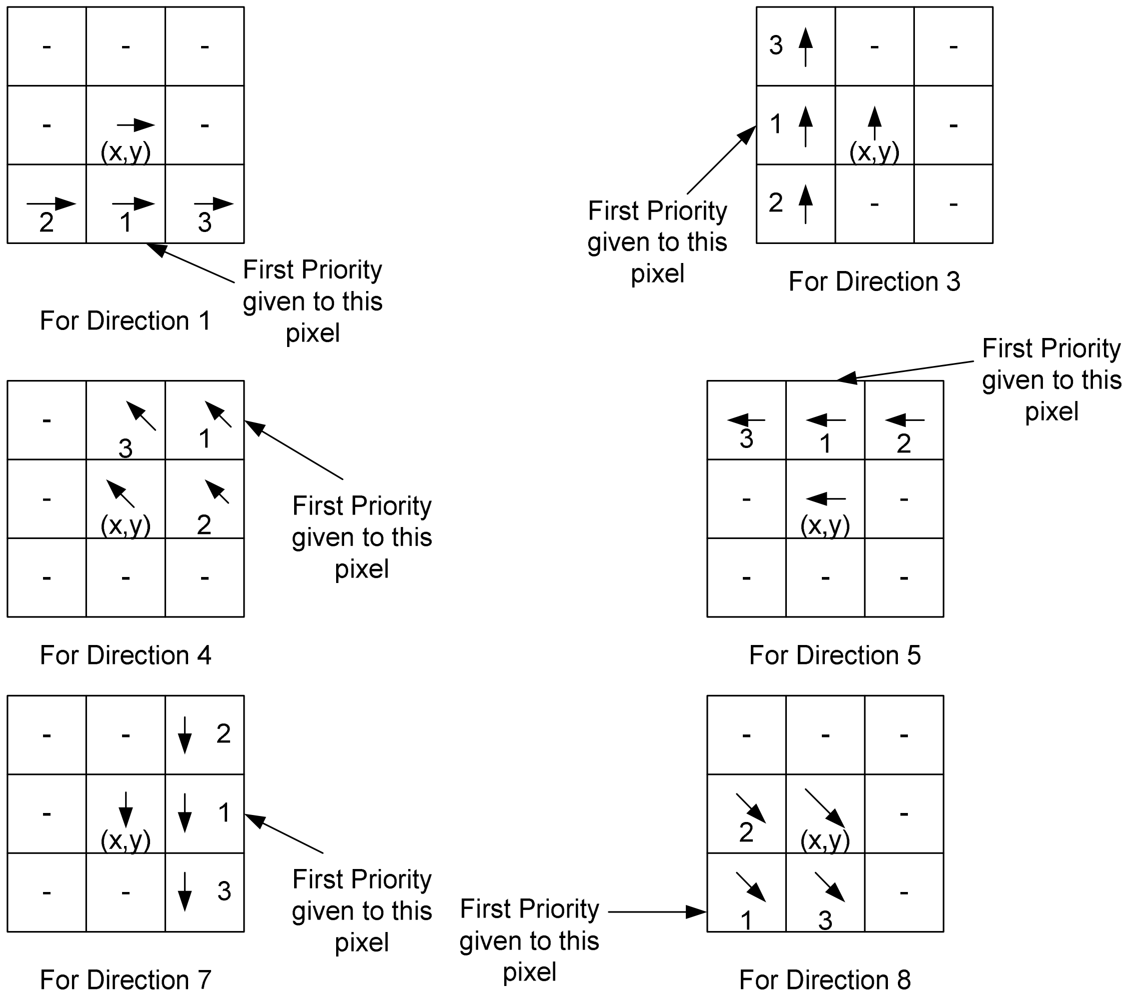

-
(x,y)
2
1
3
First Priority given to this pixel
For Direction 1
3
1
(x,y)
2
First Priority given to this pixel
For Direction 3
3
1
(x,y)
2
First Priority given to this pixel
For Direction 4
3
1
2
(x,y)
First Priority given to this pixel
For Direction 5
2
(x,y)
1
3
First Priority given to this pixel
For Direction 7
First Priority given to this pixel
2
(x,y)
1
3
For Direction 8

Figure 4.9. Represents the priority of the pixels to be scanned based on the pixel direction 1, 3, 4, 5, 7 and 8 to find an arc with same pixel direction.

arcs by contour tracing, a circle will look like Figure 4.10, with every pixel in the arc having its corresponding edge direction.

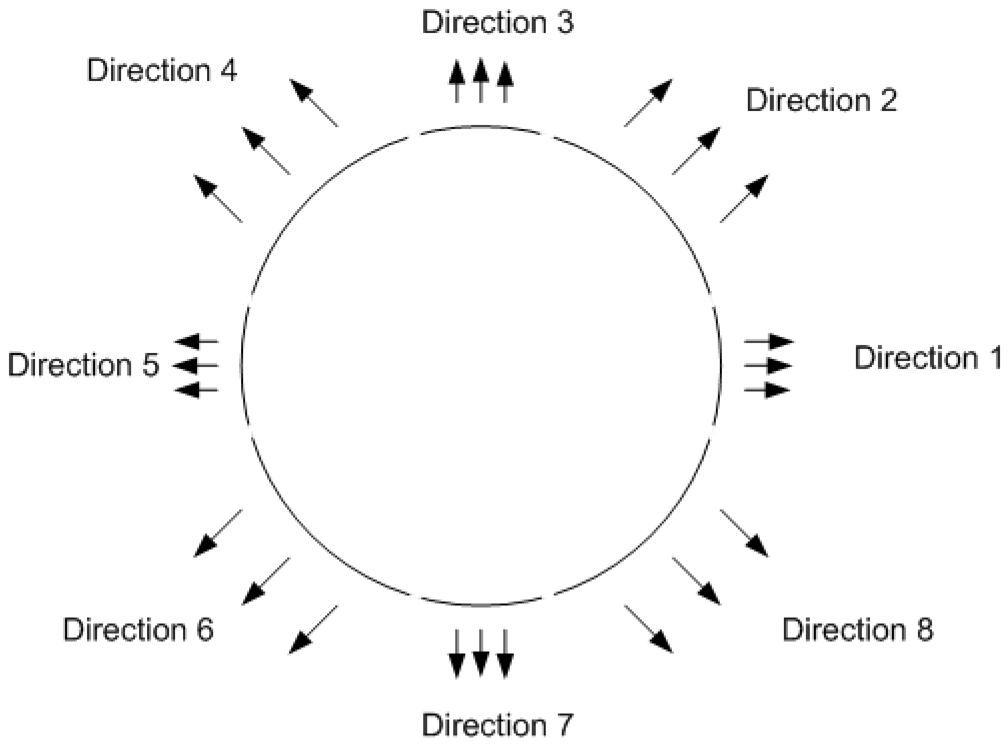


Figure 4.10. Arcs of the circle with edge direction in ideal conditions.

If we observe closely, a major part of the circle is formed by arcs with pixels directions 2, 4, 6 and 8 with arcs with pixel directions 1, 3, 5 and 7 acting as connectors. Hence to find the circle we need to find the arcs with directions 2, 4, 6 and 8 connected to arcs with directions 1, 3, 5 or 7.

To find the potential circle, we start scanning the image to find the arc with pixel direction 2 or 8. Once the arc with pixel direction 2 is found, we scan at the end point of the arc using the block of size 3x3, for a pixel part of an arc with pixel direction 1, as shown in Figure 4.11 Once the arc with pixel direction 1 is found, we scan the other end of the arc, using the block of size 3x3, for a pixel part of an arc with gradient direction 8. Once the arc with pixel direction 8 is found, we scan other end of the arc, using the block of size 3x3, for a pixel part of an arc with pixel direction 7. Once the arc with pixel direction 7 is found, wescan other end of the arc, using the block of size 3x3, for a pixel part of an arc with pixel direction 6. Once the arc with pixel direction 6 is found, three major arcs are found, which

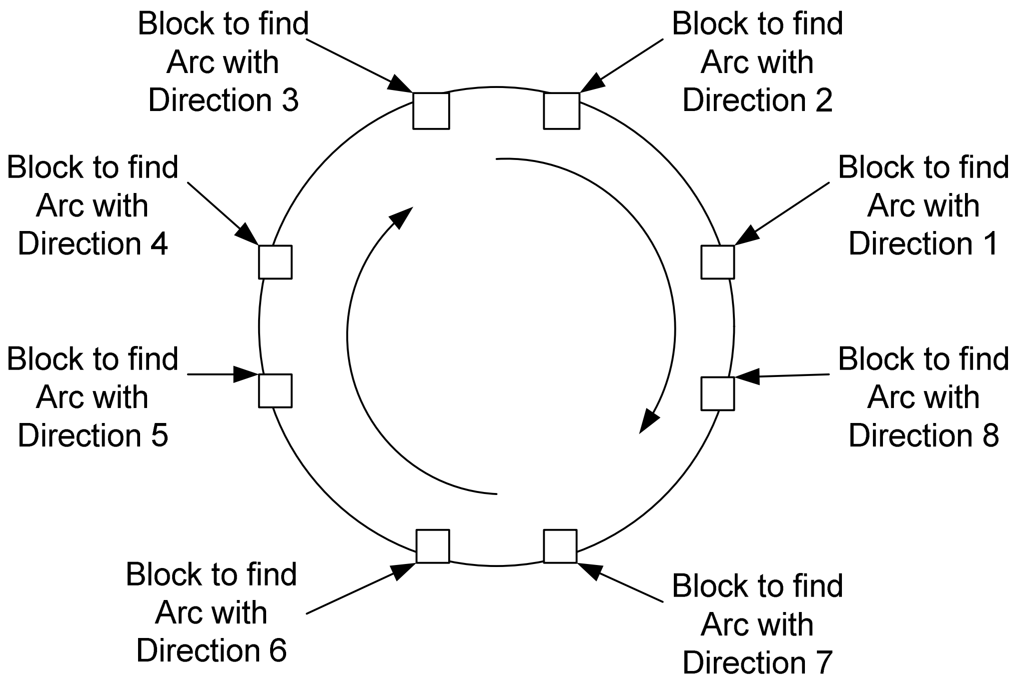

Block to find Arc with Direction 3
Block to find Arc with Direction 2
Block to find Arc with Direction 4
Block to find Arc with Direction 1
Block to find Arc with Direction 5
Block to find Arc with Direction 8
Block to find Arc with Direction 6
Block to find Arc with Direction 7

Figure 4.11. Block 3x3 connecting the arcs to find 4 major arcs.

satisfies the condition of being part of a circle. These major arcs are the input to the next block, where the center and radius of the circle are found. Hence the contour tracing continues to search for arcs with direction 2→1→8→7→6→5→4→3 as shown in Figure 4.11.

The condition for considering the arc as part of a potential circle is that there should be at least 3 arcs present with pixel directions 2, 4, 6 or 8 which are connected to each other by arcs with pixel directions 1, 7, 5 or 3. All the possible combinations of major arcs to be part of a potential circle are shown in Figure 4.12.

This condition of having at least 3 major arcs to consider the arcs to be part of a circle can be satisfied by a vertically oriented ellipse, also. To confirm that the arcs found are part of a circle only, we apply the properties of the circle to these arcs to find the center and radius. If the center and radius found are satisfied for all the arcs, then it is confirmed that these arcs form a circle. Calculating the center and radius is discussed in the next section.

4.4 Finding Center and Radius of the Circle

There are many algorithms to find the center and radius of the circle, like “A New Method of Circle’s Center and Radius Detection in Image Processing” [19]. In this proposed

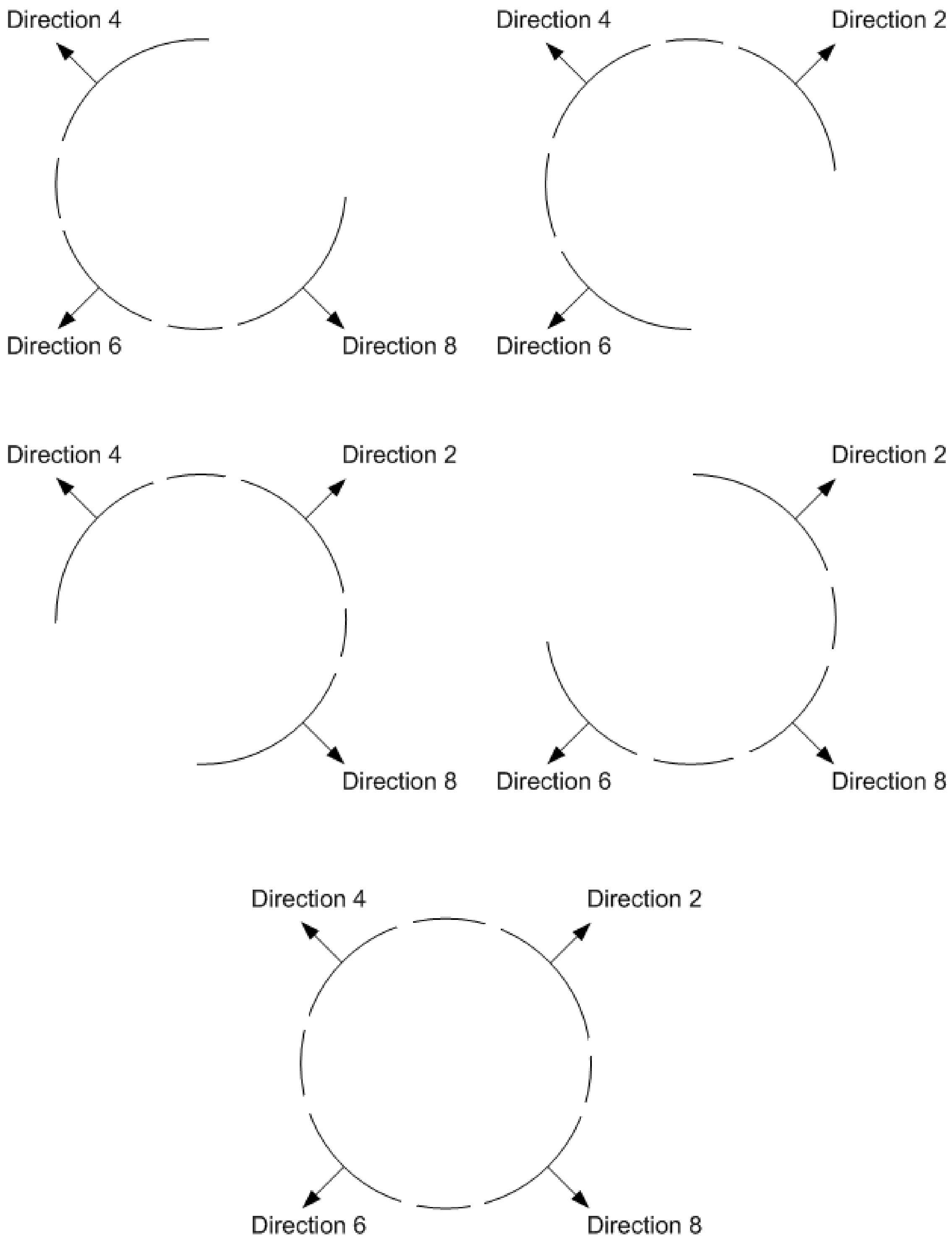


Figure 4.12. All the combinations of major arcs which can form a circle.

algorithm, we use one of the properties of the chord of the circle to determine the center of the circle. The proposed method uses simple calculations of adding and subtracting to calculate the radius of the circle.

4.4.1 Finding Center of the Circle

In this algorithm, we use the property of the circle which states, “Perpendicular bisector of a chord contains the center of the circle” [20]. The center of the circle is found as shown in Figure 4.13.

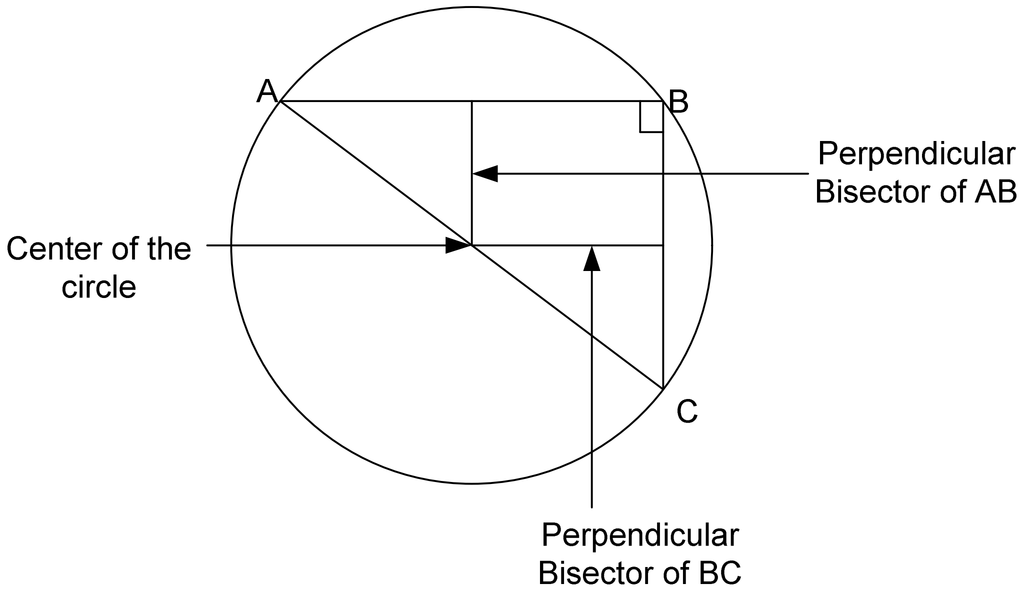


Figure 4.13. Finding center of the circle using the perpendicular bisector property of the chord.

From the arcs stored, depending on the position of the arc, either arc with direction 2 and arc with direction 8 (or arc with direction 6 and arc with direction 4) are scanned to find the points B and C, i.e., points with same x value in the spatial domain. Midpoint of BC gives us the y0 value of the center, as mentioned by the property of the circle. This procedure is carried out for all the points in the arcs to find an array of y0 values for the center of the circle. Similarly, depending on the position of the arc, either arc with direction 2 and arc with direction 4 (or arc with direction 8 and arc with direction 6) are scanned to find the points A and B, i.e., points with same y value in the spatial domain. Midpoint of AB gives us the x0 value of the center, as mentioned by the property of the circle. This procedure is carried out for all the points in the arcs to find an array of x0 values for the center of the circle. Hence, we have the array of the x0 and y0 values for the center of the circle for the given values of arcs.

The values of x0 and y0 which are repeated most number of times are noted. If these x0 and y0 values have been repeated more than 50% of the length of the array, then the value is considered as the center of the circle.

4.4.2 Finding Radius of a Circle:

The distance between the arc points and the points on the circumference of the circle gives us the radius of the circle. If x0 is the x value of center of the circle and x1 is the x value on the arc with direction 1 or 5, then the radius r2 is the difference between x1 and x0, as shown in Figure 4.14. If y0 is the y value of the center of the circle and y1 is the y value on the arc with direction 3 or 7, then the radius r1 is the difference between y1 and y0. If the both the values of radius found are same, then it is confirmed that these arcs form a circle.

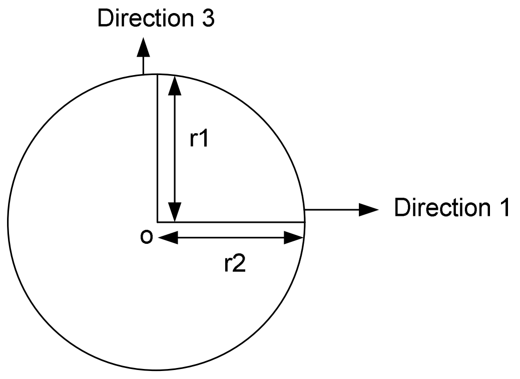


Figure 4.14. r1 and r2 radius found using the arc with pixel direction 3 and 1 respectively.

CHAPTER 5

DISCUSSION OF ARCHITECTURE

This chapter deals with the implementation of the algorithm in VHDL and synthesized on FPGA Vertex II pro. Functionalities of each block in terms of hardware implementation are discussed. Hardware is designed with a goal of minimum usage of memory and to achieve the results within 33ms, i.e., to match the frequency of 30 frames/s. The Architecture of the proposed algorithm is as shown in Figure 5.1.

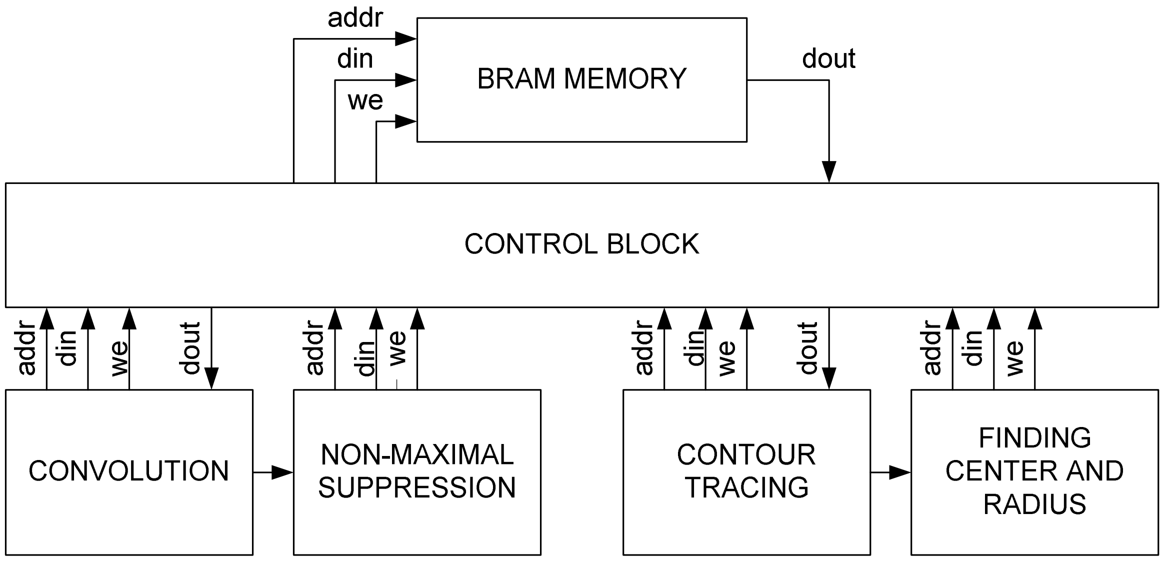


Figure 5.1. Architecture of the proposed algorithm.

There are six blocks in the proposed algorithm:

- BRAM memory is the storage space where input images and output results are stored.
- Convolution block calculates the direction and strength of edges of input images
- Non-maximum suppression, to get the thin edges.
- Control Block, to link the signals between BRAM and other blocks.
- Contour Tracing, to contour trace the edges to get the potential circle arcs.
- Finding center and radius of the circle.

Convolution, Non-Maximum Suppression, Contour Tracing and Finding Center and Radius block together are grouped and referenced in this thesis as “Processing block”. The grayscale image is stored in BRAM Memory, which is taken as input to the Convolution Block.

5.1 BRAM MEMORY

Block memory [21], as shown in Figure 5.2, is used to store the image data and the results of the processing blocks. Image stored in BRAM is of size 100x100, with each pixel data of size 12 bits. The memory used to store the image is calculated as:

Memory used = 100*100*12

= 120000 bits

= 120000/8 = 15Kb.

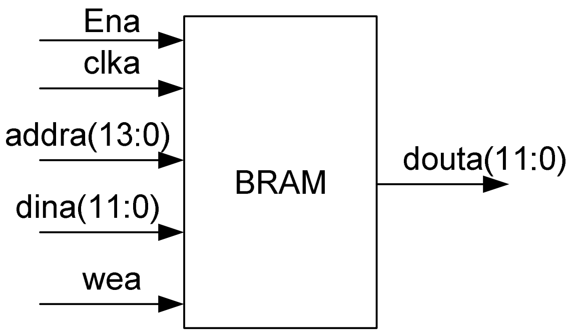


Figure 5.2. Block diagram of BRAM.

Signal “Ena” should always be enabled to use the memory. Reading or writing the data from or to the memory is controlled by the write enable “we” signal. When the write enable signal “we” is disabled, the data stored in the address “addra” is read from the output signal “douta”. When the write enable signal “we” is enabled, input data is written using the signal “dina” to the memory address “addra”. Memory used here is in read-first mode wherein when write enable is activated, data previously stored in the address is read first, while the input data is written. This is illustrated in the waveform [21] in Figure 5.3.

5.2 CONVOLUTION

A convolution block is used to calculate the pixel direction and the edge strength for a pixel which is used as input by NMS block and contour tracing block to determine the arcs which can be part of a potential circle. In this section we also derive an equation to calculate

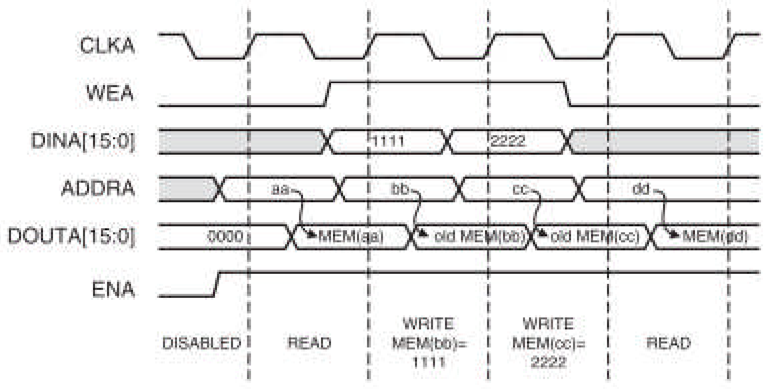


Figure 5.3. Read-first mode example.

the time it takes to convolve an image. Figure 5.4 shows the block diagram of the convolution used to calculate the pixel direction and the edge strength.

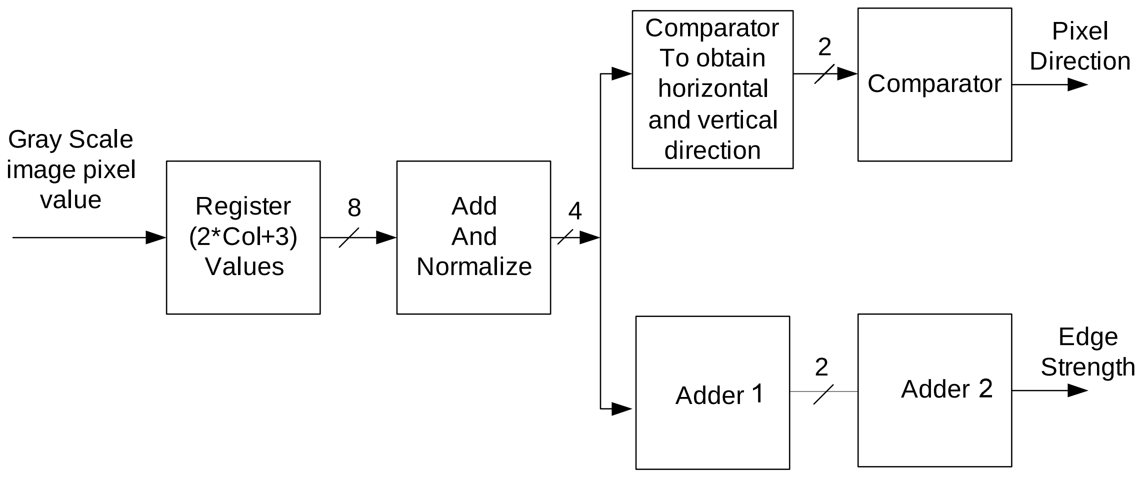

Gray Scale image pixel value
Register (2*Col+3) Values
8
Add And Normalize
4
Comparator To obtain horizontal and vertical direction
2
Comparator
Pixel Direction
Adder 1
2
Adder 2
Edge Strength

Figure 5.4. Block diagram of convolution.

As shown in Figure 5.4, the block diagram for convolution receives the gray scale image of size, Row x Column, as input to calculate the pixel direction and the edge strength, as explained in section 4.2.1. The 2*Column + 3 pixel of the input image is stored in the Register before starting the calculations, as shown in Figure 23. It takes 2*Column + 3 cycles to start the calculations. In next three clock cycles, pixel values (x0,y0), (x1,y0), (x2,y0), (x0,y1), (x1,y1), (x2,y1), (x0,y2), (x1,y2), (x2,y2) are used by Add and Normalize block to calculate two values for horizontal kernel (Ix1 and Ix2) and two values from vertical kernel (Iy1 and Iy2)(as discussed in chapter 4, section 4.2). There is a possibility that there might be an overflow in the add results. Thus, normalization is implemented to avoid the overflow. For an example of normalization, consider the equation (11a) used to calculate Ix1 and expand it to get equation (5.1):

$$Ix1 = I*Kx1.$$

$$Ix1 = I(x0,y0)*1 + I(x0,y1)*2 + I(x0,y2)*1 \tag{5.1}$$

Normalization is obtained by dividing equation (17) by 4. Four is obtained by adding all the elements in kernel Kx1. Hence the modified equation (18) is given by

$$Ix1/4 = I(x0,y0)*1/4 + I(x0,y1)*1/2 + I(x0,y2)*1/4 \tag{5.2}$$

The implementation of an Add and Normalize block for calculating Ix1 using equation (18) is shown in Figure 5.5.

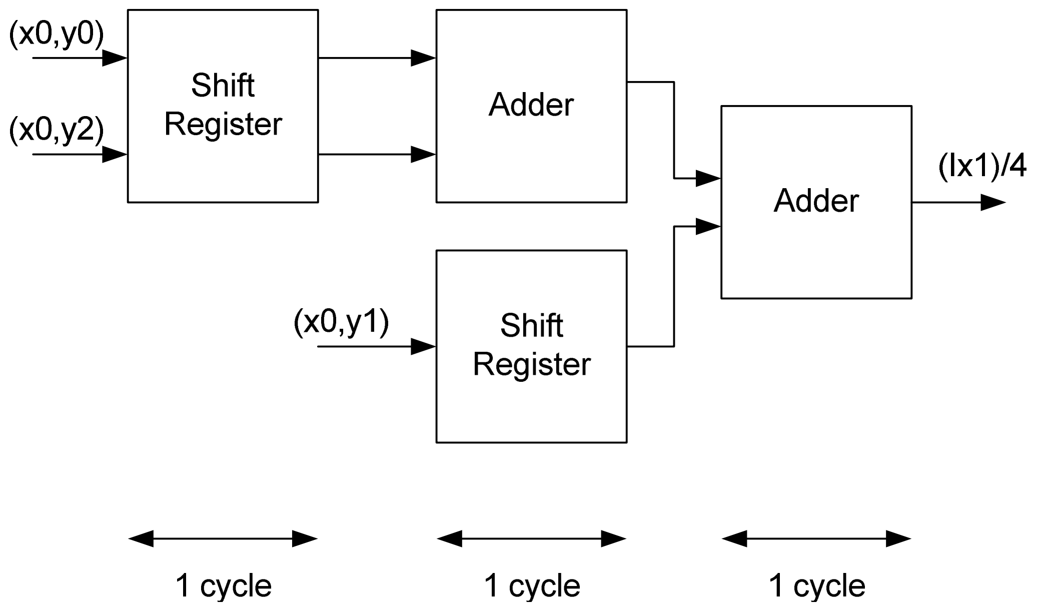


Figure 5.5. Implementation of add and normalize block and illustration of number of cycles taken to calculate.

As shown in Figure 5.5, pixel value (x0,y0) and (x0,y2) is taken as input by the shift register to obtain I(x0,y0)/4 and I(x0,y2)/4 values and then passed through the adder to add these two values. I(x0,y0)/2 is taken as input to the shift register to get I(x0,y1)/2. These two results are added by the adder to find Ix1. Similarly, Ix2, Iy1 and Iy2 are calculated. Notice this process takes 3 cycles to calculate Ix1, as shown in Figure 5.5.

“Adder 1” and “Adder 2” block in Figure 5.5 use the results from the Add and Normalize block to calculate the edge strength. Horizontal and vertical pixel direction is obtained by comparing the results from the Add and Normalize block. This result is compared to get the pixel direction as discussed in section 4.2.1, Table 4.1 (p. 17).

As shown in Figure 5.6a, pixel (x2,y2) in green, is the last pixel taken as input before starting the calculations. After calculating the pixel direction and edge strength for the pixel (x0,y0), pixel value (x0,y0) is never used again in the calculations. Hence, we replace (x0,y0) with (x0,y1) and replace (x0,y1) with (x0,y2) to efficiently use the memory, as shown in Figure 5.6b.

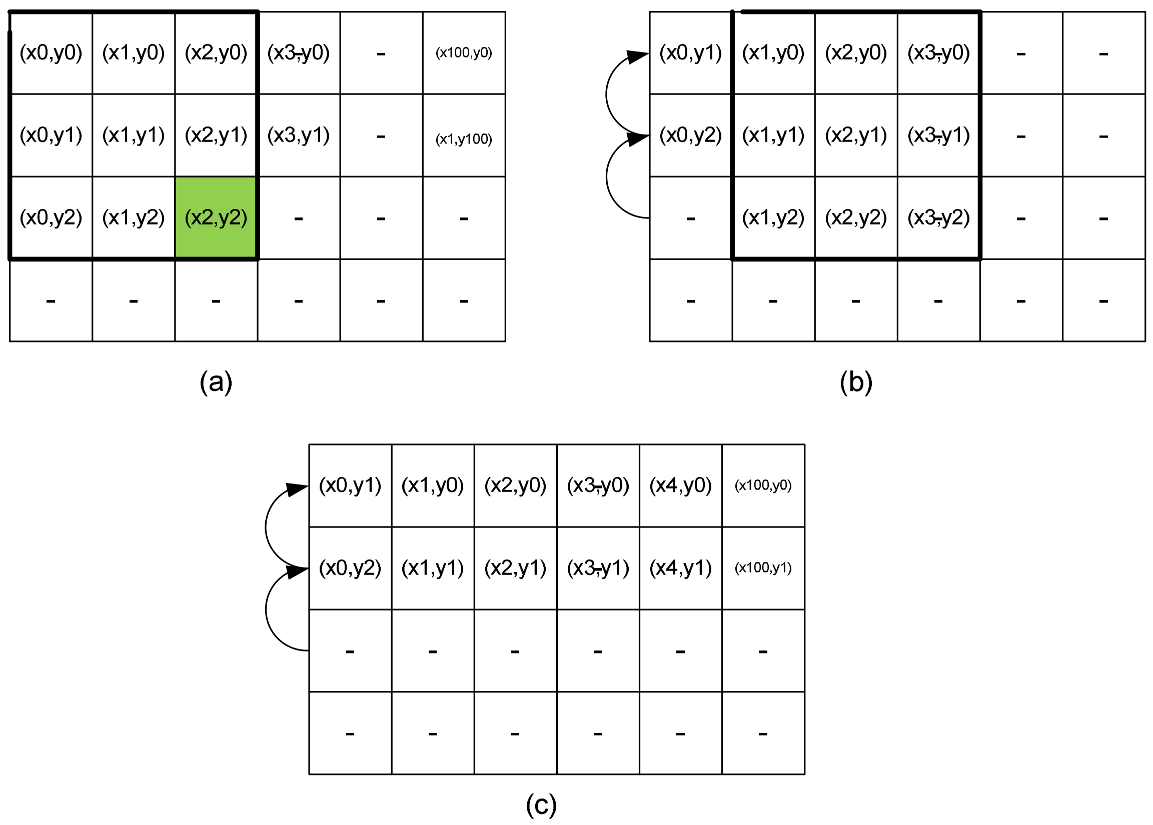

(x0,y0) (x1,y0) (x2,y0) (x3,y0) - (x100,y0)
(x0,y1) (x1,y1) (x2,y1) (x3,y1) - (x1,y100)
(x0,y2) (x1,y2) (x2,y2) - - -
(a)
(x0,y1) (x1,y0) (x2,y0) (x3,y0) - -
(x0,y2) (x1,y1) (x2,y1) (x3,y1) - -
- (x1,y2) (x2,y2) (x3,y2) - -
(b)
(x0,y1) (x1,y0) (x2,y0) (x3,y0) (x4,y0) (x100,y0)
(x0,y2) (x1,y1) (x2,y1) (x3,y1) (x4,y1) (x100,y1)
(c)

Figure 5.6. (a) When pixel value of (x2,y2) is received, calculation of pixel direction and edge strength for pixel (x1,y1) is started. (b) In the next cycle, when (x3,y2) pixel value is received, pixel value at (x0,y1) is moved to (x0,y0) and pixel value at (x0,y2) is moved to (x0,y1). (c) Results after scanning the whole row.

This replacing process is continued to the end of the row, i.e., till (xcolumn,y2) is reached. At this point, the whole of the first row is replaced by the second row and the second row is replaced by the third row, as shown in Figure 5.6c . Next, pixel direction and edge direction calculation are not started till the first 3 values of row 4 are received. This process is continued until the whole image is scanned and the pixel direction and the edge strength value for every pixel are calculated.

The pixel direction and edge strength obtained for every pixel is sent to the non-maximum suppression block every cycle. These values are stored in registers in the non-maximum suppression block to evaluate these values and retain only those values which form a thin edge for a boundary of an object. The next section discusses the implementation of NMS.

5.3 NON-MAXIMUM SUPPRESSION

The purpose of NMS is to get thin edges of one pixel width. As shown in Figure 5.7, pixel direction and edge strength results of the convolution block are taken as input to this block. Calculation is started when 2*column+3 pixel directions and edge magnitudes are received.

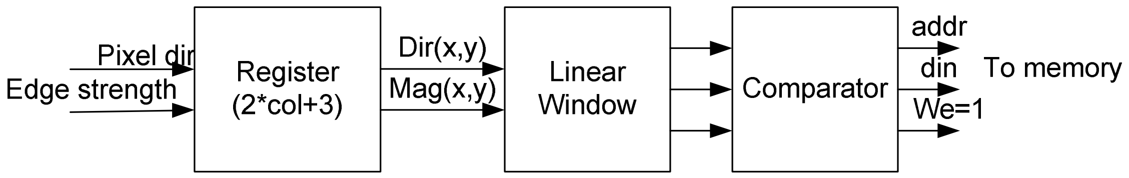


Figure 5.7. Block diagram of non-maximum suppression.

Once 2*column+3 pixel direction and edge strength are received, a linear window of size 3 pixels is considered corresponding to the pixel direction of the pixel (x,y). Hence the NMS block takes 2*column+3 cycles to start the calculations. The linear window block, as shown in the block diagram, sends pixel direction and edge strength of the three pixels in the linear window to the comparator. If the edge strength of the pixel (x,y) is less compared to the edge strength of the neighboring pixels in the linear window, then the pixel (x,y) is given zero value. If the edge strength of the pixel (x,y) is greater than any of the neighboring pixels in the linear window, then the output is the pixel direction of pixel (x,y). Hence this process takes only one cycle to derive at the output.

A detailed explanation of the comparator is shown in Figure 5.8. Let us consider the case when the pixel direction is 1, as shown in Figure 5.8. The linear window when the pixel direction is 1 consists of the pixels (x-1,y),(x,y) and (x+1,y). Pixel direction and edge strength are compared using the comparator within the linear window to get the output pixel

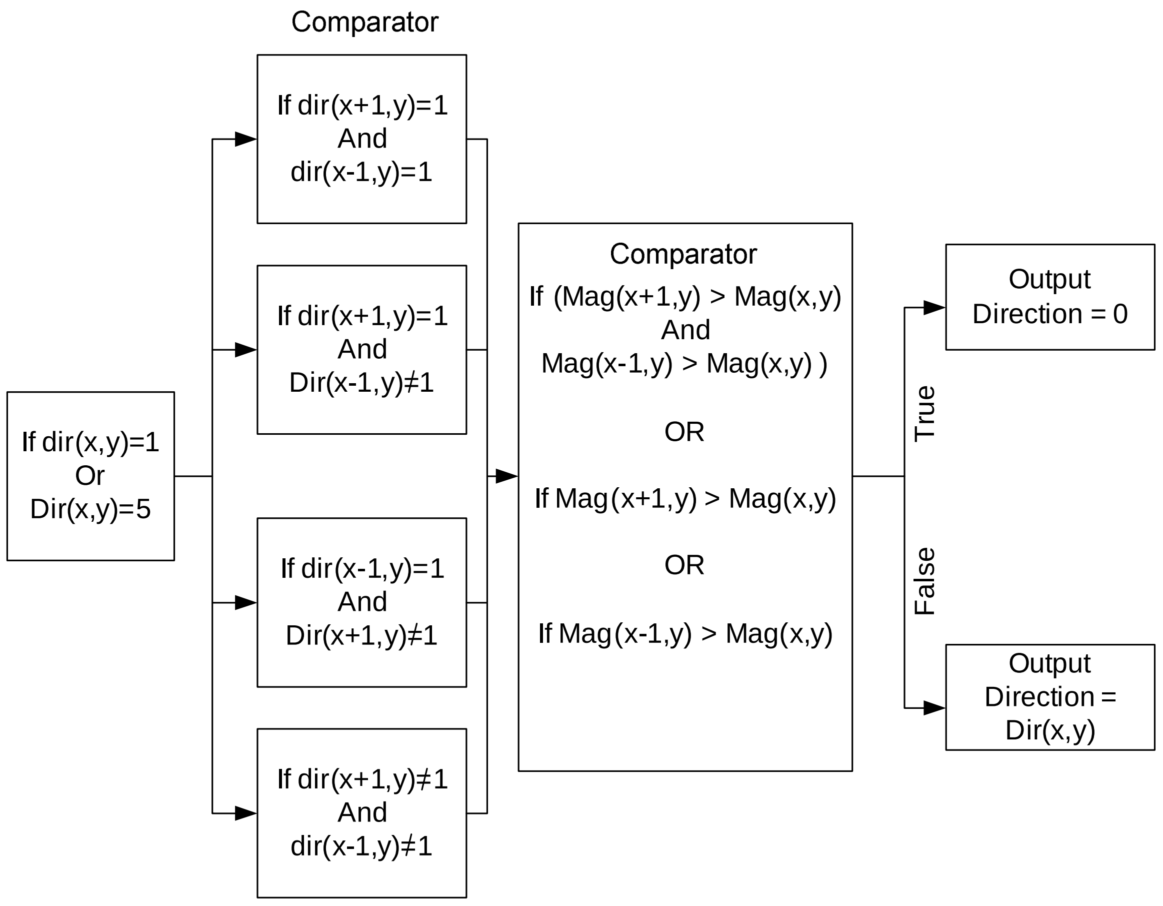

Comparator
If dir(x+1,y)=1 And dir(x-1,y)=1
If dir(x+1,y)=1 And Dir(x-1,y)≠1
If dir(x,y)=1 Or Dir(x,y)=5
If dir(x-1,y)=1 And Dir(x+1,y)≠1
If dir(x+1,y)≠1 And dir(x-1,y)≠1
Comparator
If (Mag(x+1,y) > Mag(x,y) And Mag(x-1,y) > Mag(x,y))
OR
If Mag(x+1,y) > Mag(x,y)
OR
If Mag(x-1,y) > Mag(x,y)
True
False
Output Direction = 0
Output Direction = Dir(x,y)

Figure 5.8. Block diagram of non-maximal suppression when the edge direction is 1.

direction for the pixel under consideration. Hence, the output of this block will be either zero pixel direction and zero edge strength or pixel direction and edge strength of the pixel under consideration.

This process is executed for all the pixels in the image to eliminate the pixel values which are not required. Results of NMS are stored in the memory. When all the pixels in the image are processed, a signal is sent to the Contour Tracing block to begin its process.

5.4 Control Block

The control block is a block which acts as a link between BRAM memory and the processing blocks, as shown in Figure 5.9. Depending on the requirements of the stage of processing, the control block decides on which processing block will have the access to BRAM memory. The control block acts as a multiplexer. As shown in Figure 5.9, when the

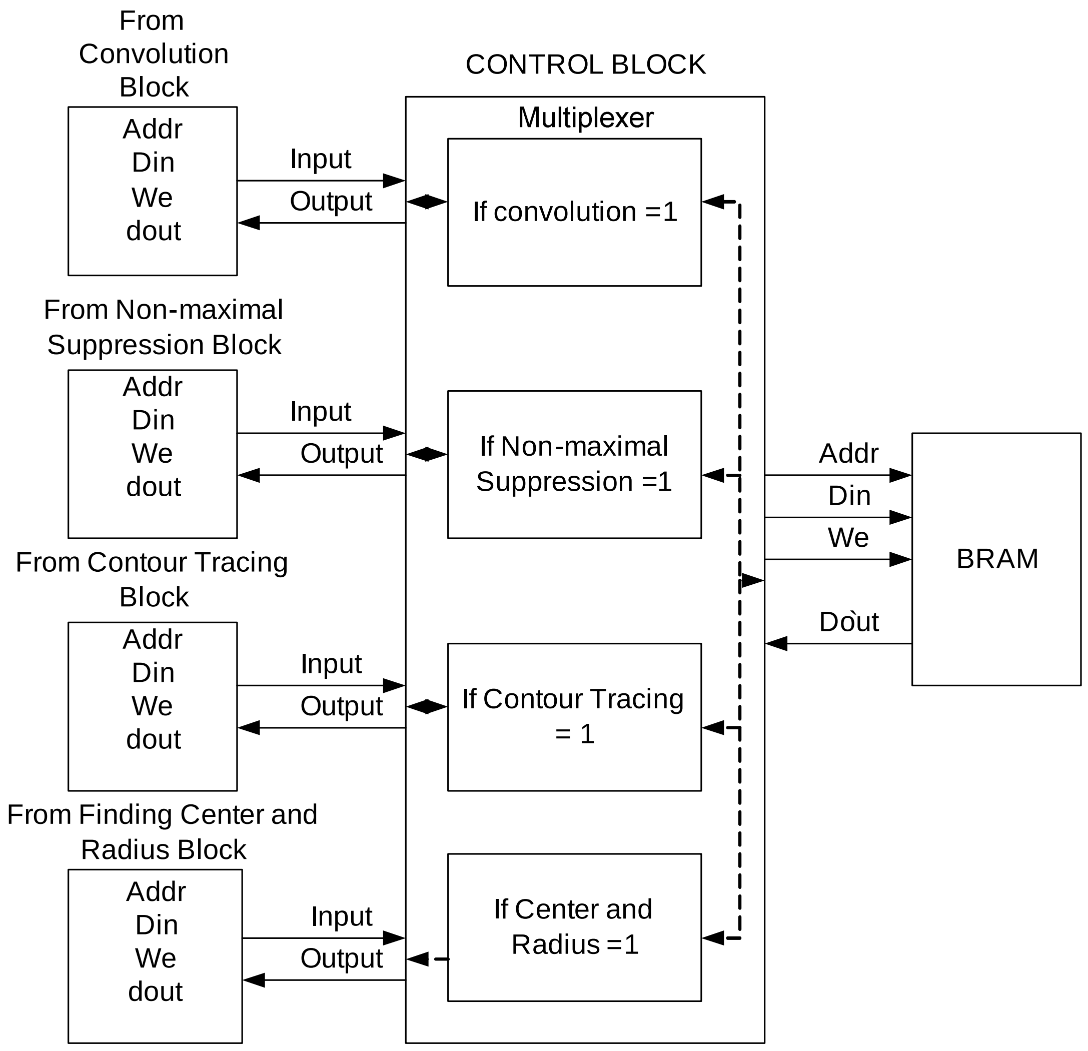


Figure 5.9. Block diagram of control block.

convolution signal is high, only the convolution block has access to BRAM memory and all other blocks wait until the processing of the convolution block is complete. Hence, the control block makes sure that only one of the processing blocks is accessing the memory at a given time, hence reducing the requirement of using another BRAM memory block for the "Processing block". As the control block acts as a buffer between the memory and the processing blocks, the presence of control block does increase the time taken to access the memory by one cycle. As pipeline [22] processing is implemented in the Convolution and Non-Maximal Suppression stages, the effect of increase of time by the control block is reduced.

Let us discuss the time taken to determine a thin edge image. Consider the image size as Row x Column. As discussed in section 5.2, time taken to start the calculation is 2*column + 3. Time taken to calculate the pixel direction and edge strength is given by 5 cycles, i.e., 3 cycles for Add and normalize block, as shown in Figure 5.7 (p. 36), and 2 cycles used by Adder 1 and Adder 2. Due to the pipelining process, all the pixel direction and edge strength calculation takes only one cycle. As discussed in section 5.3, time taken to start the calculations in NMS block is 2*column + 3 and for further calculations it takes one cycle per pixel. Hence, total time taken for calculating thin edge image from pixel direction and edge strength is Row x Column, which includes the 2*column + 3 cycles waiting time to start the calculation.

Thus, the total time taken to determine the thin edge image from gray scale image is given by:

$$\text{Time taken} = (2*\text{column} + 3) + 5 + (\text{Row x Column}) \text{ cycles} \tag{5.3}$$

If the input image is of size 100x100, then the time taken to calculate the thin edge image will be given by equation (5.3).

$$\begin{aligned}\text{Time taken} &= (2*100+3) + 5 + (100*100)\\ &= (10000 + 208) \text{ cycles.}\end{aligned}$$

Thus, we can conclude that the results of convolution and non-maximum suppression together can be achieved in the time given by equation (5.3), which is almost close to achieving the results of each pixel per cycle.

5.5 CONTOUR TRACING

In this section, we discuss the implementation of the Contour Tracing block. The purpose of this block is to eliminate all the spurious points and retain only the arcs which can be part of the potential circle. The Contour Tracing block uses the non-maximum suppression results stored in BRAM memory as input.

Let us focus first on implementing the process of finding an arc. As discussed in Chapter 3, section 4.2, we scan the results of non-maximum suppression to find the pixel with edge direction 2. Once the pixel with edge direction 2 is found, the prediction block predicts the next possible location of the edge pixel with edge direction 2, as shown in Figure 5.10.

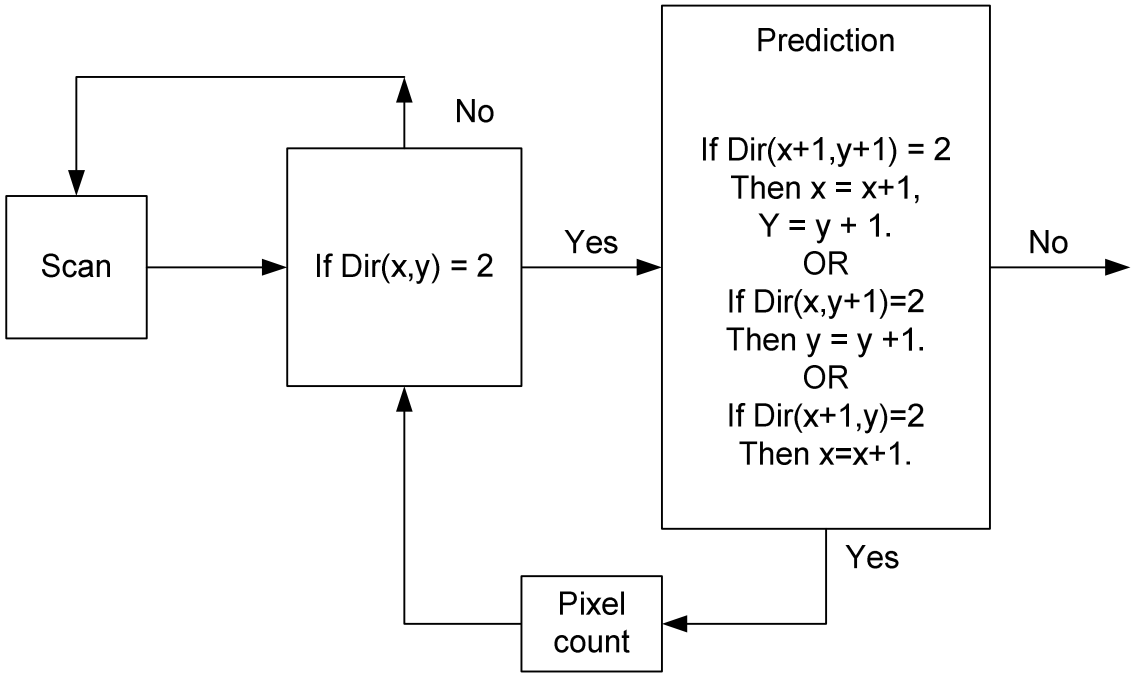


Figure 5.10. Example of contour tracing the arc to find an arc.

The prediction block is a comparator which compares the pixel direction of a pixel under consideration with other neighboring pixels, as discussed in section 4.3. The prediction block uses the pixel direction information to predict the next edge pixel location. Hence, the prediction block reduces the time required to contour trace the arc. For discussion, let us consider the pixel under consideration is of pixel direction 2. Every time a pixel with edge direction 2 is found while contour tracing, the pixel count is increased by 1 with updating the location to the latest pixel position with direction 2. This process is continued until the point there is no pixel with edge direction 2. Every edge pixel belonging to the arc is stored in the register. When the contour tracing for an arc is complete, if the pixel count is less than 10 we ignore the arc and start scanning the image from where it was stopped. The next arc found is overwritten on the register, thus eliminating the invalid arcs. If an arc length is greater than 10 pixels, then the arc is considered to be valid. All the edges that are contour traced are updated to zero pixel direction. Thus the same arc is not contour traced again.

The prediction block in Figure 5.11 predicts the edge direction of the next arc which should be connected to the valid arc found for it to be part of the potential circle. If the predicted arc is found, then the prediction block predicts the next arc to be found. If at any

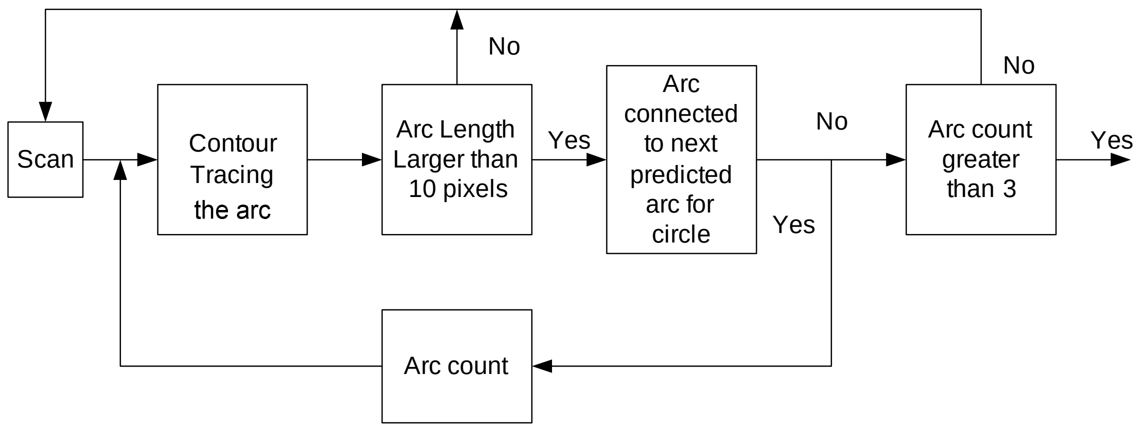

Scan
Contour Tracing the arc
Arc Length Larger than 10 pixels
No
Yes
Arc connected to next predicted arc for circle
No
Yes
Arc count greater than 3
No
Yes
Arc count

Figure 5.11. Block diagram for contour tracing.

point the predicted arc is not found, then the scanning starts from the point where it was stopped. The next valid arc found is overwritten on the previously stored registers, hence discarding the previously found arcs. This process continues till three major arcs are found. Thus, the three major arcs found are used by the next block to find the center and radius of the circle. Before going to the next block, let us discuss how the arcs are stored in the register and how it helps in making the calculation easier.

5.5.1 Memory Segmentation

Every edge pixel which can be a potential part of the arc is stored in the register. Before storing the x and y values of the edge pixel, the edge direction of the pixel is stored and the next address is kept empty to fill in the end address of the arc, as shown in Figure 5.12. Locations of the edge pixels belonging to an arc are stored in the next address onwards till the arc ends, which is represented by zero. The end address of the arc is now noted and stored in address position 1, as shown in Figure 5.12. This process is continued for the next major arc connected until three major arcs are found. Once the three major arcs are found, the next block to find the center and radius of the circle is invoked while the process of contour tracing is paused.

5.5.2 Time Taken by Contour Tracing Block to Scan in a Worst-Case Scenario

In this section, we discuss the time taken to detect a circle using the proposed algorithm. As we already know from section 4.4, the proposed algorithm traces the edges

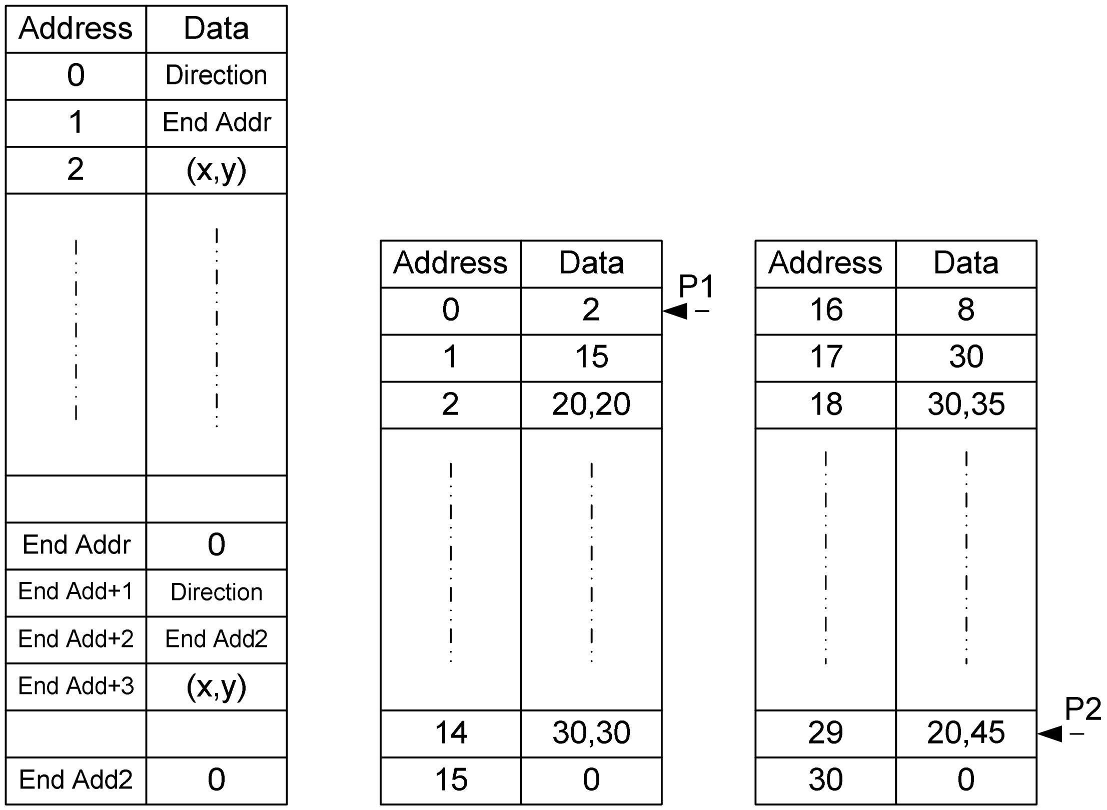


Figure 5.12. (a)Organization of the arcs stored in the register. (b) Example of storage of two major arcs with edge direction 2 and 8.

from the results of NMS. Thus, the time taken will depend on the number of edge pixels found. So in this section, we calculate the time taken to detect the circles in the worst-case scenario.

The worst-case scenario for the proposed algorithm will be the case when there are the maximum number of edges present for an image. For any image, the maximum possible number of edges will be the total number of pixels present in an image, i.e., Row x Column. For a worst-case scenario:

$$\text{Maximum possible edge pixels} = (\text{row x column}) \qquad (5.4)$$

From section 5.5 we know that while contour tracing the proposed algorithm, which scans and checks three neighboring pixels, takes 3 cycles per edge pixel. As we know from section 5.1, it takes 2 cycles to read the data from the memory and it takes 2 cycles to write the data in to the memory.

Knowing all this information, we can derive the equation for the time taken to contour trace the edge for a worst case scenario. The equation is as follows.

The number of cycles taken by contour tracing block to scan the image for the worst-case scenario

= 2 cycles (to read each pixel) x no. of pixels + 4 cycles (3 neighboring pixel scan + 1 extra cycle to read the first pixel) x no. of edge pixels

= (2+4) x (number of edge pixels)

Thus, from equation (5.4),

$$= (6 \times \text{row} \times \text{column}) \qquad (5.5)$$

Consider an image of size 100x100 for example to calculate the cycles for the worst case scenario. Then from equation (5.5),

No. of cycles taken = 6 x 100 x 100

= 60,000 cycles.

5.6 Finding Center and Radius of the Circle

In the previous section, three major arcs are found which can be a part of the potential circle. The purpose of the Finding Center and Radius of the Circle block is to apply the properties of the circle to these three major arcs and verify the results to confirm if these arcs form a circle.

As discussed in section 5.2, the first value in the array of arcs values gives us the edge direction. The second value gives us the end address of the arc. The end address + 1 gives us the starting address of the next arc, as shown in the Figure 5.12a (p. 42). Knowing the end address of the arc also gives us the starting address of the next major arc. Keeping this concept of finding the starting and ending addresses of the arc, let us move on to find the center and radius of the circle from these arcs.

To find the y value of the center combination of the arcs with edge directions 2 and 8 or 4 and 6 is used. Let us consider the arcs with edge directions 2 and 8. As shown in the Figure 5.13, x1 value of arc with edge direction 2 is compared with x2 value of arc with edge direction 8. If x1 < x2, x1 is increased till x1=x2; otherwise, if x1>x2 then x2 is decreased till x1=x2. At this point if a straight line is drawn between A(x1,y1) and B(x2,y2) then a vertical line is obtained. Using the property of the circle, i.e., the perpendicular bisector of the chord

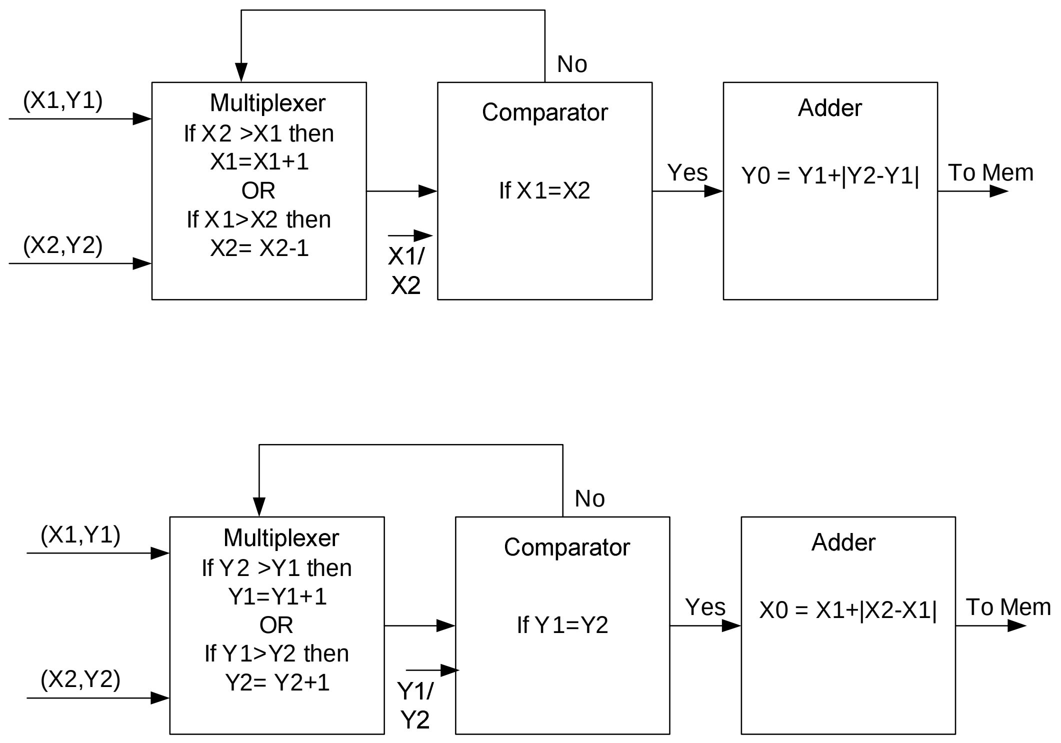


Figure 5.13. Block diagram of a system that locates the center and radius of a circle.

contains the center of the circle, we can conclude that the center of the line AB has the y0 value of the center, which is calculated by the adder block. This process is continued with all the points in the arcs 2 and 8 to find the array of y0 values.

Similarly, to find the x value of the center, a combination of the arcs with edge directions 2 and 4 or 8 and 6 is used. Let us consider the arcs with edge directions 2 and 4. As shown in Figure 5.13, y1 value of an arc with edge direction 2 is compared with y3 value of an arc with edge direction 8. If y1 < y3, y1 is increased till y1=y3 else if y1>y3, then y3 is increased till y1=y3. At this point if a straight line is drawn between A(x1,y1) and C(x3,y3), then a horizontal line is obtained. The center of the horizontal line AC has the x0 value of the center. This process is continued with all the points in the arcs 2 and 4 to obtain the array of x0 values.

In this array of x0 and y0, the number of times a value is repeated is calculated as shown in Figure 5.13. If the highest number of times a value repeated is greater than or equal to half of the length of the array of center, i.e., the value is repeated more than or equal to 50% of the length of the array of center, then the value repeated the highest number of times is considered the center. Thus, center (x0, y0) is found with more than 50% probability of (x0, y0) being the center. The center found can be a center of a vertical ellipse, also. To confirm that the center found is center of a circle, we will find the distance between the center found and the arc with edge directions 1 or 5 and also the distance between the center found and the arc with edge directions 3 or 7. If the distance found is same, then it is confirmed that the center and radius found are of a circle. Thus, the circle is detected with the location and radius of the circle.

Let us discuss, the time taken to calculate the center and radius of the circle for a worst-case scenario. Time taken to calculate the center and radius will always depend on the length of the arc. Let us consider the length of the arc as ‘L’. According to section 5.6, the operations performed to calculate the center and the radius are

- Scanning the arcs stored in the memory to find the pixels with same x or y values in the spatial domain. In the worst-case scenario, the whole arc is scanned for every edge pixel in the corresponding arc. Hence if the length of both the arcs is ‘L’ then the number of cycles it will take to scan is L x L.
- Calculating the center by subtracting the position value of two pixels, finding the mid-points and then adding to get the exact location, corresponding to the two related arcs. Executing these steps once will take 3 cycles. If the length of the arc is ‘L’ then to calculate the center for both x and y axis is given by = 2(for both x and y) x 3 (three cycles for calculation for each points) x L

Hence the total number of cycles taken for the worst case will be as given by steps 1 and 2.

$$= (L \times L) + (6 \times L) \tag{5.6}$$

CHAPTER 6

EXAMPLES OF DEPENDENCIES ON PARAMETERS AND CONCLUSION

Edge detection is basic to low-level image processing and good edges are necessary for higher level processing [10]. While the behavior of an edge detector may fall within tolerances in specific situations, edge detectors can have difficulty adapting to different situations. The quality of edge detection is highly dependent on lighting conditions, the presence of objects of similar intensities, the density of edges in the scene, and noise. In this section we will be discussing some of the parameters, like threshold and intensity of the pixel, which affect image processing, as well as how the Canny edge detector and proposed algorithm behave in relation to the changes in these parameters.

6.1 INVARIANT OF THRESHOLD

As discussed in Section 2.4, the Canny edge detection uses thresholding with hysteresis with two threshold values, T1 and T2 [2]. With this algorithm one or more thresholds are required, at different situations, for selecting which points belong to edges and which do not. Whenever a threshold is required, an issue to find an intensity threshold which will work for different images has not been solved satisfactorily.

Let us discuss the dependency of the Canny edge detector on Threshold. The Matlab Canny edge detector function has been used to derive the edges of objects in an image with defaults of 0.2, 0.5 and 0.9 as threshold parameters, as shown in Figure 6.1. As the threshold parameter increases, the number of edge points detected decreases. Hence, in canny edge detection as the threshold Th or Tl increases, the possibility of eliminating a potential edge increases. Similarly, as Th or Tl decreases, the possibility of considering non-edge pixel increases [8].

The proposed algorithm does not depend on intensity threshold. This algorithm uses pixel direction to detect edges, which is one of the principal properties of an edge [9]. As

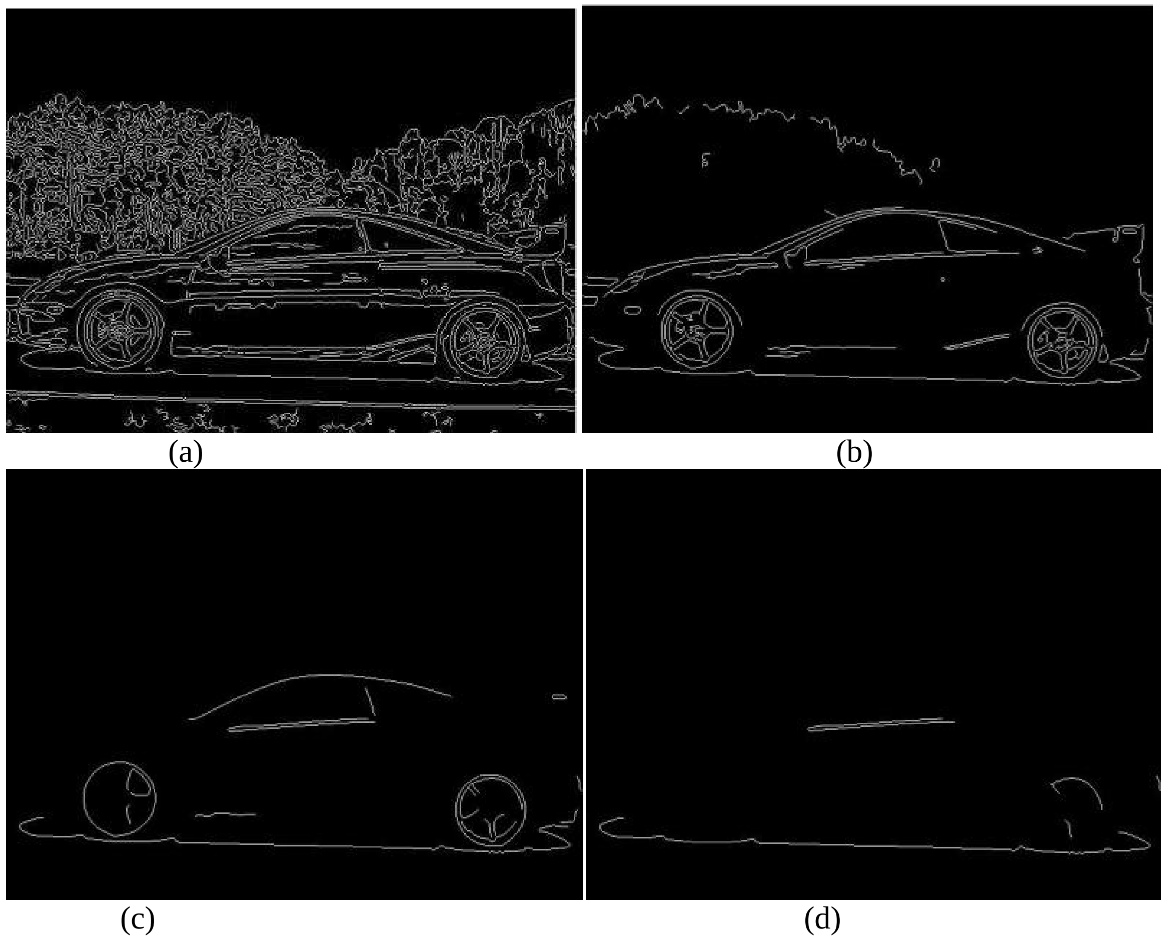


Figure 6.1. Matlab canny edge detector used to detect the edge of an image. Edge detected (a) with default value of threshold; (b) with 0.2 times the default threshold; (c) with 0.5 times the default value (d) with 0.9 times the threshold value.

only pixel directions are used to validate the edges of a circle, there is no threshold used to find an edge. Thus, the issue of finding the right threshold for every image is solved.

6.2 Sensitivity to Illumination

Many factors, like shade, texture, illumination, etc., in the image affect the results of the edge detection [8]. The whole goal of the edge detector is to discard the edges created as the result of shade, texture and illuminations, and give the edge segments for only the outer boundaries of the objects. This information is used by algorithms like Hough Transforms to detect different shapes in the image. Hence, it is important that enough information is given to the preceding transforms. In Figure 6.2, edges are detected using the Canny edge detector with a threshold parameter of 0.9, as shown in Figure 6.2c). Similarly, the edges are detected using our algorithm as shown in Figure 6.2e. The intensity of the original image is decreased

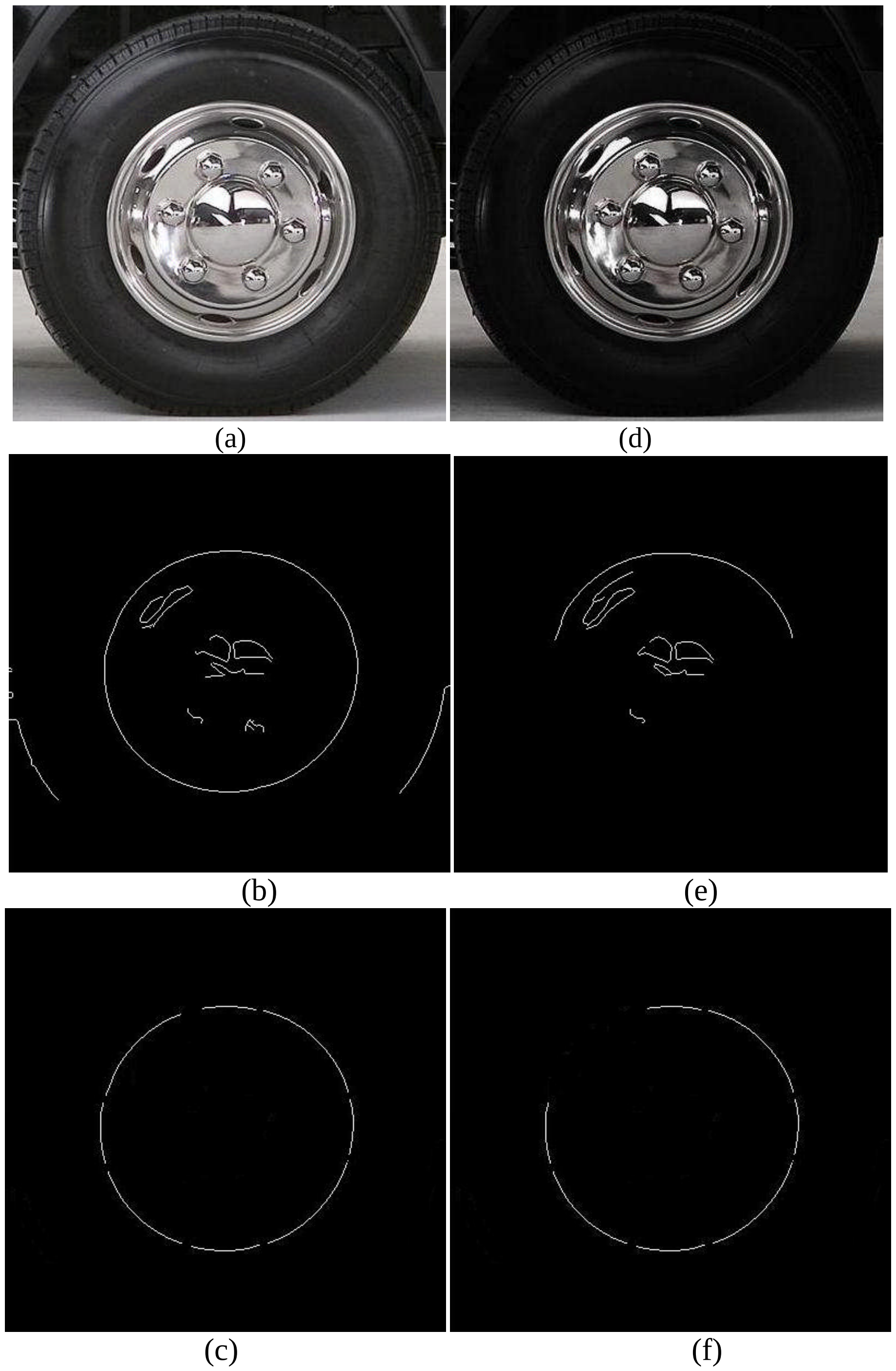


Figure 6.2. Illustration of dependency of the Canny edge detector and the proposed algorithm on change in intensity. (a) Original Image; (b) The Canny edge detection result with threshold parameter set to 0.9; (c) Proposed algorithm result; (d) Original image with reduced intensity by 20%; (e) Result of Canny edge detector with (d) as input with 0.9 as threshold parameter. Notice that the reduction in edges is detected. (f) Proposed algorithm result. Notice not much difference in results and has enough information to detect a circle.

to get the image in Figure 6.2b (p. 48). Edges are detected using the Canny edge detector with same threshold parameter as previously used, i.e., 0.9. Notice that the number of edges detected has decreased. When our algorithm is used we get a similar response as the response in Figure 6.2e (p. 48).

Illumination does affect our algorithm, but the sensitivity to illumination is less compared to the Canny edge detector because it uses the rate of change of pixel value, whereas we use the edge direction, which is derived by comparing the pixel values.

6.3 CALCULATION OF TIME TAKEN TO DETECT A CIRCLE

Detecting a circle within 33ms, i.e., detecting a circle from video data which is updating at the rate of 30 frames/s, will confirm that this algorithm can be used for real-time applications. Devices which perform image analysis to produce consecutive frames are defined to be in 3 sets of 24 fps, 25fps and 30fps [23]. The proposed algorithm is designed to be extended for a maximum value, i.e., 30fps and hence can detect circles for video data scanning at 30 frames per second.

As already discussed in sections 5.4.1, 5.5.2 and 5.6.1, the total time taken to detect the circle in a worst-case scenario is given by

Cycles taken to detect circles

= (2*column + 3) + 5 + (Row x Column) + (6 x row x column) + (LxL) + (6xL).

Where row is the number of rows in the image.

Column is the number of columns in the image.

L is the length of the arc.

Consider an image of size 100x100, for example, to calculate the time taken to detect circles. For this example consider L as 100

Cycles taken = (200 + 3) + 5 + (10000) + (25000) + (10000) + (6000)

= 51208 cycles.

If we consider the frequency of clock signal as 50MHz, then

Time taken is given by

= 51208/50M

= 0.001024 sec

= 1.024 ms

which is faster than 33ms, which is the standard considered to compare.

6.4 Memory Usage

In this section, the amount of memory used is discussed for the proposed algorithm and is compared with the memory usage of Circular Hough Transform. As discussed in section 5.2 (BRAM), the memory here is used only to store the image and then while calculating the center and the radius.

While storing the image the amount of memory used is given by

$$= \text{Row x Column x 12 bits / 8 bytes} \tag{6.1}$$

The length of the arc will determine the size of the memory used to store it. For 100x100 image the highest radius possible is 50 pixels. The circumference of the circle will determine the length of the arcs used. The circumference is given by:

$$\begin{aligned} &= 2\pi \text{ x radius.} \\ &= 2 \text{ x } 3 \text{ x radius. (rounding off the value of } \pi \text{ to 3)} \qquad (6.2) \\ &= 2 \text{ x } 3 \text{ x } 50 \\ &= 300 \text{ pixels.} \end{aligned}$$

Memory usage because of the arcs does not contribute significantly. Hence, the memory used is given by equation (6.1) and equation (6.2) for the image 100x100:

$$\begin{aligned} &= 100 \text{ x } 100 \text{ x } 12 / 8 + 300 \\ &= 15.3 \text{ kbyte.} \end{aligned}$$

Memory usage for Circular Hough Transform is given by [15]:

$$\begin{aligned} &= \text{Row x Column x number of edges.} \\ &= 100 \text{ x } 100 \text{ x } 100 \text{(for a worst case scenario)} \\ &= 1000000 \text{ x } 12/8 \\ &= 1.5 \text{ Mbyte.} \end{aligned}$$

Hence, it is clear from the above comparison that the memory usage in the proposed algorithm is efficient compared to the Circular Hough Transform.

6.5 Conclusion

A threshold invariant algorithm has been proposed that uses a modification of the Canny edge detector. The algorithm has been tested on a group of 30 images using Matlab and found to have reduced sensitivity to changes in image intensity compared with the Canny edge detector.

The proposed algorithm has been implemented in VHDL and synthesized on FPGA Vertex II pro. The design code performs the evaluation at a time interval less than 33 ms, thus the algorithm can be extended to extract circles in a video with frame rate of 30 frames per second. The size of the memory used in the proposed algorithm is less than the memory used in the Circular Hough Transform.

REFERENCES

[1] J. Canny, “A computational approach to edge detection,” *IEEE Trans. Pattern Anal. and Mach. Intell.*, vol. 8, pp. 679-714, Nov. 1986.

[2] L. S. Davis, "A survey of edge detection techniques," *Comput. Graph. and Image Process.*, vol. 4, pp. 248-260, Sept. 1975.

[3] W. Dong and Z. Shisheng, “Color image recognition method based on the Prewitt operator,” *Int. Conf. Computer Science and Software Engin.*, vol. 6, pp. 170-173, Dec. 2008.

[4] S. Lanser and W. Eckstein, “A modification of Deriche's approach to edge detection,” in *Proc. Int. Conf. Image, Speech, and Signal Anal.*, The Hague, Netherlands, 1992, pp. 633-637.

[5] M. Grazia Albanesi, “Time complexity evaluation of algorithm for the hough transform on mesh-connected computers,” in *Proc. 5th Annu. European Computer Conf.*, Bologna , Italy , 1991, pp. 253-257.

[6] P. Y. Hsiao, C. H. Chen, S. S. Chou, L. T. Li, and S. J. Chen, “A parameterizable digital-approximated 2D Gaussian smoothing filter for edge detection in noisy image,” in *IEEE Inter. Symp. Circuits and System*, Island of Kos, Greece, 2006, pp. 3189-3192.

[7] D. Venkateshwar Rao and M. Venkatesan, “An efficient reconfigurable architecture and implementation of edge detection algorithm using handle-C,” *Inter. Technology: Coding and Computing*, vol. 2, pp. 843-847, Apr. 2004.

[8] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Upper Saddle River, NJ: Prentice Hall, 2003.

[9] R. C. Gonzalez and R. E. Woods, *Digital Image Processing Second Edition[M]*. Upper Saddle River, NJ: Pearson Education Inc., 2002.

[10] C. Sun and P. Vallotton, “Fast linear feature detection using multiple directional non-maximum suppression,” in *Proc. Int. Conf. Pattern Recognition*, Hong Kong, China, 2006, pp. 288-291.

[11] G. M. Schuster and A. K. Katsaggelos, “Robust circle detection using a weighted mse estimator,” in *Proc. Inter. Conf. Image Processing*, Singapore, 2004, pp. 2111-2114.

[12] Z. Li, Y. Liu, J. Yun, and F. Huang, “A new edge detection method based on contrast enhancement,” in *Conf. Future BioMedical Inform. Eng.*, Sanya, China, 2009, pp. 164- 167.

[13] F. Yan, X. Shao, G. Li, Z. Sun, and Z. Yang, “Edge detection of tank level IR imaging based on the auto-adaptive double-threshold canny operator,” *Intell. Inform. Technology Applicat. Research Assoc.*, vol. 3, pp. 366 - 370, Dec. 2008.

[14] R. O. Duda and P. E. Hart, “Use of the hough transformation to detect lines and curves in pictures,” *Commun. the ACM*, vol. 15, pp. 11-15, Jan. 1972.

[15] M. Nixon and A. S. Aguado, *Feature Extraction & Image Processing*. Burlington, MA: Academic Press, 2008.

[16] E. R. Davis, *Machine Vision: Theory, Algorithms, Practicalities*. Burlington, MA: Academic Press, 1990.

[17] G. Q. Lu, H. G. Xu; and Y. B. Li, “Line detection based on chain code detection,” in *IEEE Int. Conf. Vehicular Safety*, Xian, China, 2005, pp. 98-103.

[18] H. Freeman, “Computer processing of line drawing images,” *Computing Surveys*, vol. 6, no. 1, pp. 57-97, Mar. 1974.

[19] M. Z. Zhang and H. R. Cao, “A new method of circle’s center and radius detection in image processing[C],” in *IEEE Int. Conf. Automation and Logistics*, Qingdao, China, 2008, pp. 2239-2242.

[20] H. R. Jacobs, *Geometry: Seeing, Doing, Understanding*. New York, NY: Macmillan, 2003.

[21] Xilinx. (2008). *LogiCORE Block memory generator v2.7 data shee*t [Online]. Available: http://web.engr.oregonstate.edu/~tavakola/ Data%20Sheets/blk algorithm, we use one of the properties of the chord of the circle to determine the center of the circle. The proposed method uses simple calculations of adding and subtracting to calculate the radius of the circle. _mem_gen_ds512.pdf

[22] M. J. Quinn, *Parallel Programming in C with MPI and Open MP*. New York, NY: McGraw-Hill Professional, 2004.

[23] Electronics and Electrical Engineering Laboratory, *Measurements for Competitiveness in Electronics*. Gaithersburg, MD: Diane Publications, 1993.