DEVELOPMENT OF A CLONAL SELECTION ALGORITHM BASED SYSTEM FOR AUTOMATIC DETECTION OF MULTIPLE SHAPES

By

Simon Obute OBUTE

M.Sc/Eng/2099/2011-2012

elcymon@gmail.com

Supervisory Team:

Prof. M. B. Mu'azu (Chairman)

Dr. S. Garba (Member)

A Dissertation Submitted to the Department of Electrical and Computer Engineering, Ahmadu Bello University, Zaria in Partial Fulfilment of the Requirements for the Award of Master of Science (M.Sc.) Degree in Control Engineering

September, 2017

DECLARATION

I, Simon Obute OBUTE, hereby declare that the work in this dissertation entitled Development of a Clonal Selection Algorithm Based System for Automatic Detection of Multiple Shapes has been carried out by me in the Department of Electrical and Computer Engineering. The information derived from literature has been duly acknowledged in the text and a list of references provided. No part of this dissertation was previously presented for another degree or diploma at this or any other institution.

Simon Obute OBUTE		
Name of Student	Signature	Date

CERTIFICATION

This dissertation entitled DEVELOPMENT OF A CLONAL SELECTION ALGORITHM BASED SYSTEM FOR DETECTION OF MULTIPLE SHAPES by Simon Obute OBUTE meets the regulations governing the award of degree of Master of Science (M.Sc.) in Control Engineering of the Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

Prof. M. B. Mu'azu		
Chairman, Supervisory Committee	Signature	Date
Dr. S. Garba		
Member, Supervisory Committee	Signature	Date
Dr. Y. Jibril		
Head of Department	Signature	Date
Prof. S. Z. Abubakar		
Dean School of Postgraduate Studies	Signature	Date

DEDICATION

To my parents and siblings.

ACKNOWLEDGEMENT

My deepest gratitude goes to God for His mercy, love, faithfulness and favour. Surely He never fails and makes a way where there seems to be no way.

To Prof. M. B. Mu'azu and Dr. S. Garba, my supervisors, I appreciate your patience, favour and guidance throughout the course of the work. I also appreciate the students and staff of Electrical and Computer Engineering that have at various stages critiqued, encouraged and advised me with the aim of bringing the best out of the research.

Many thanks to my parents Dr. Obute Garba and Mrs. Elizabeth Obute-Garba. They have been a great source of motivation and encouragement to me and have sacrificed so much to ensure I complete my program here at ABU. My siblings, Oloche, Obotu, Oyilonye, Adakoyi and Omoche have also inspired and encouraged me, all making sure I have a conducive environment and state of mind to complete my research work.

Thanks to the numerous friends like Oyeniyi, Grace, Yesufu, Samson and Rita among others who have encouraged me, supported me financially and in many other ways.

Many thanks to you all. I pray that God, who unfailingly rewards every good deed, will abundantly reward you. Amen.

Simon Obute OBUTE

June, 2016.

ABSTRACT

In this dissertation a Clonal Selection Algorithm (CSA) based system that detected multiple instances of circles, quadrilaterals and triangles in an image scene was developed. CSA models how B-cell antibodies of the immune system protect the body from invading antigens. The developed system eliminated the need for separate implementations of the CSA for detecting the different shapes in an image. The implementation of the system was done using MATLAB 2015a, the edges and corners in the image scene were first extracted using Canny edge and Harris corner detectors and the edge-only image was used as the antigen. A candidate solution was formed by random selection of three (3) edge points to form a circle, four (4) corner points to form a quadrilateral and three (3) corner points to form a triangle. The CSA iterated until either an optimal solution (fitness of 1.0) is attained or when a maximum of 100 iterations is reached. The antibodies with high fitness (above the set threshold of 0.9 on synthetic images and 0.5 on real images) were then analysed using a distinctness factor to detect all instances of the desired shapes and eliminate duplicate detections. A repository of five (5) synthetic images generated with MATLAB 2015a and six (6) real images captured with digital camera were used to test the performance of the algorithm. Simulation results showed sub pixel accuracy with Mean Absolute Error (MAE) between 0.44 and 0.52, Mean Squared Error (MSE) between 0.4722 and 0.5208 and Peak Signal to Noise Ratio (PSNR) between 56.9233 and 57.2381 on the synthetic test images. False Positive Rate (FPR) of 0% and False Negative Rate (FNR) of 3% were gotten on the synthetic images while the FPR and FNR on real images were 4.76% and 3.82% respectively. The implemented CSA had a mean error score of 0.14 for circle detection compared to 0.36 and 0.34 for Circle CSA and Learning Automata (LA) representing 61.11% and 58.82% improvements respectively.

TABLE OF CONTENTS

xx

xx

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABFOA	Adaptive Bacterial Foraging Optimization Algorithm
AIS	Artificial Immune System
ANN	Artificial Neural Networks
BFOA	Bacteria Foraging Optimization Algorithm
CSA	Clonal Selection Algorithm
DIRECT	DIviding RECTangle
EDPF	Edge Drawing Parameter Free
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
GA	Genetic Algorithm
HT	Hough Transform
JPEG	Joint Photographic Experts Group
KCC	K-means based Consensus Clustering
LA	Learning Automata
MAE	Mean Absolute Error
MATLAB	MATrix LABoratory
MCA	Midpoint Circle Algorithm
mCSA	Multiple shapes Clonal Selection Algorithm
MLA	Midpoint Line Algorithm
MSE	Mean Squared Error
NPV	Negative Predictive Value
PPV	Positive Predictive Value
PSNR	Peak Signal to Noise Ratio
RGB	Red Green Blue
RHT	Randomized Hough Transform
TN	True Negative

TP	True Positive
TPR	True Positive Rate

CHAPTER ONE

INTRODUCTION

1.1 Background

The detection of objects and shapes is an important research area in image analysis, attracting a large number of publications yearly. The successful detection and classification of shapes has been applied for automatic detection of road signs, inspection of manufactured products, aided vectorization of drawings, target detection and many other applications (Cuevas *et al.*, 2012a, Jiang, 2012, Akinlar and Topal, 2013, Li, 2014). Object or shape detection involves searching images for objects or shapes of interest in order to extract high-level information from the scene under analysis (Ramirez *et al.*, 2011). The images are usually fed into the system via cameras, after which the system or algorithm analyses the image to extract relevant information from it (i.e. the presence or absence of a desired object or shape).

An algorithm for detecting shapes/objects in a scene is considered efficient (Akinlar and Topal, 2013) if it:

1. is able to detect multiple instances of the object(s) of interest in an image;
2. is robust, detecting occluded or overlapping shapes, that is partially complete image of the desired shape;
3. discriminates between the various shapes present in the image;
4. is robust to noise, being able to detect the presence of the desired object(s) in the presence of noise;
5. is able to perform its task with little memory requirement, at least the relationship between image size or dimension and memory requirement should be linear and not exponential;
6. is fast, able to detect the presence or absence of object(s) of interest in real time if possible.

Object detection can broadly be classified into detection of non-geometric and geometric shapes.

a) Detection of objects with non-geometric shapes

This involves detection of objects whose physical shape cannot be analysed geometrically. Example of such objects include features like maps, mountains and trees. Detecting such shapes or objects can be infeasible using geometrical analysis. It is best detected using template matching, where an algorithm is trained to detect a template of the real object. The effectiveness of the algorithm is then tested by presenting test images of these objects to it. Another strategy is to break down the image of the object into smaller shapes or features that are characteristic to the particular object and training the algorithm to detect these smaller features. When presented with an image, the algorithm searches for these features in the image. Artificial Neural Networks (ANN) has been used for template matching of non-geometric shapes (Chaki and Parekh, 2011).

b) Detection of objects with geometric shapes

Man-made features like windows, doors, traffic signs, boxes, balls and walls have geometric shapes. This category deals with the detection of these shapes which can be easily analysed geometrically. The properties of these shapes can be fully described by geometric theorems, for example the distance from the centre of a circle to any point on its circumference is a constant known as the radius of that circle. Methods for detecting these shapes make use of these properties to check for their existence in a scene. Though template matching can also be used for detecting these shapes/objects (Gavrila, 1998), detection based on geometric properties of these shapes have found wide interests among researchers. A particular advantage of using the latter is that it eliminates the need for training the system/algorithm. Hough Transform (HT) based methods are among the earliest methods used for detection of geometric shapes. However, the memory requirement of the algorithm increases tremendously when the dimension of the shape is greater than two (for example circle dimension is three) (Dasgupta *et al.*, 2010). This has led to various methods to improve its performance and exploration of other approaches to shape detection. Some of other approaches to shape detections include: Random Hough Transform (RHT) (Jiang, 2012), Genetic Algorithm (GA) (Ramirez *et al.*, 2011), Arti-

ficial Immune System (AIS) (Cuevas *et al.*, 2012a), Learning Automata (LA) (Cuevas *et al.*, 2012b) and Clustering-Based algorithms (Scitovski and Marošević, 2015).

This research uses an Artificial Immune System (AIS) based algorithm known as the Clonal Selection Algorithm (CSA), to detect the presence of geometric shapes (circles, triangles and quadrilaterals) in an image. The immune system can detect several antigens with different characteristics and has memory of past encounters with antigens for more aggressive response when similar antigens are encountered in the future. These properties make the AIS suitable for solving multi-objective, pattern recognition and multi-modal problems (Ulutas and Kulturel-Konak, 2011) - detecting multiple shapes in this case.

1.2 Motivation

The state of the art in automatic detection of geometric shapes involves the development of algorithms capable of detecting a single shape type. The motivation for this research springs from the need to have a single system that is capable of detecting multiple shape types in an image. Such a system eliminates the need of implementing separate algorithms for detecting the different shape types; and makes the process of multiple shape types detection more efficient and less time consuming.

1.3 Problem Statement

After extensive search in literature on detection of shapes based on geometric properties, it has been observed that most researchers focused on the detection of single shapes or multiple instances of that shape in an image. This means, a user intending to apply these methods for detecting multiple shape types will have to use separate detection system per shape. This is inefficient and time consuming when applied to scenarios where multiple shapes need to be identified and classified.

1.4 Aim and Objectives

The aim of the research is the development of a Clonal Selection Algorithm (CSA) based system for detection of multiple shapes.

The objectives of the research are as follows:

1. To develop a CSA based system for detection of multiple shapes (circles, triangles and quadrilaterals) irrespective of their positions, sizes and orientations in an image;
2. To enhance the developed system in 1. to be able to detect multiple instances of these shapes during a single run of the CSA algorithm and be robust in the presence of noise;
3. To implement the system developed in 1. and 2.; validate by computing the MSE, PSNR, MAE, and error score on five (5) synthetic images developed using MATLAB; and computing the False Positive Rate (FPR) and False Negative Rate (FNR) on the synthetic images and six (6) real image scenes captured on digital camera.

1.5 Methodology

The steps taken to achieve the Clonal Selection Algorithm (CSA) for detecting multiple shapes are:

1. Prepare a repository of images:
   (a) Generate, using MATLAB, three (3) synthetic images of multiple circles, quadrilaterals and triangles in which the shapes are of varying dimensions, positions and orientation in the image scene and two (2) synthetic images containing a mixture of the desired and undesired shapes.
   (b) Save the locations of the pixels used to generate the shapes. These are used as ground truth for the images.
   (c) Save the generated images in Joint Photographic Experts Group (JPEG) format.

(d) Capture images of circular, triangular and quadrilateral objects using a digital camera and save them on the computer.

2. Develop CSA based systems for detection of circles, triangles and quadrilaterals:

   (a) Develop CSA that detects a circle in an image scene.

   (b) Develop CSA that detects a triangle in an image scene.

   (c) Develop CSA that detects a quadrilateral in an image scene.

3. Enhance the developed CSA systems to detect multiple instances of desired shapes in a single CSA algorithm:

   (a) Develop a system of analysing the CSA memory to extract other detected circles, that is other antibodies with fitness higher than a set threshold, and compute the MSE and PSNR when detecting synthetic circles of 1(a).

   (b) Apply the system developed in 3(a) for multiple triangles in an image and compute the MSE and PSNR when detecting synthetic triangles of 1(a).

   (c) Apply the system developed in 3(a) for detecting multiple quadrilaterals in an image and compute the MSE and PSNR when detecting synthetic quadrilaterals of 1(a).

   (d) Combine the antibody pools of circle, triangle and quadrilateral detectors.

   (e) Develop a CSA that utilizes the antibody pool of 3(d) for detecting antigens (desired shapes) in an image scene.

   (f) Test the developed CSA of 3(e) with synthetic images of 1(a) by computing its MSE, PSNR, MAE and error score on these images.

4. Use the CSA implementation of 3 for detecting the desired shapes in real images using the captured images in 1(d).

5. Compute the False Positive Rate (FPR) and False Negative Rate (FNR) on detection of the whole test images (both synthetic and real images).

1.6 Dissertation Organization

The background and introduction to the project has been provided in chapter one. Chapter two provides the relevant theories incorporated into the research for multiple shapes detection and a review of literature in the field. The detailed design decisions made in implementing the CSA system for multiple shapes detection is provided in chapter three. Results and analysis of the results is provided in chapter four while chapter five provides concluding remarks and recommendations for future research.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This Chapter introduces the fundamental concepts of the project and reviews similar works done in the detection of geometric shapes in an image scene.

2.2 Review of Fundamental Concepts

This section provides an overview of concepts fundamental to this study.

2.2.1 Definition of shapes to be detected

The research focuses on the detection of three (3) geometric shapes: circles, triangles and quadrilaterals. For the purpose of this research, these shapes can be defined as follows:

a) Circle

A circle is the locus of a point which moves such that it remains at a fixed distance (called the radius) from a fixed point (called the centre) (Pender *et al.*, 2011).

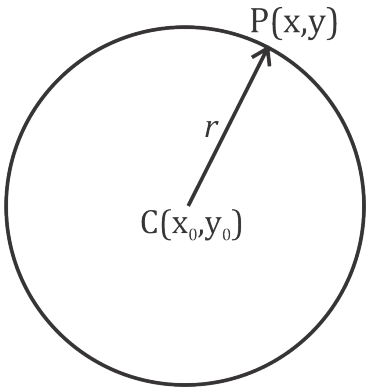


Figure 2.1: Circle with Centre at $C(x_0, y_0)$ and Radius r

Given the circle in Figure 2.1, the radius, r, is defined as the Euclidean distance between the centre $C(x_0, y_0)$ and any point on its circumference, say $P(x, y)$.

$$CP = r \tag{2.1}$$

But $CP = \sqrt{(x - x_0)^2 + (y - y_0)^2}$

$$\therefore \qquad \sqrt{(x - x_0)^2 + (y - y_0)^2} = r \tag{2.2}$$

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{2.3}$$

Equation (2.3) is the equation for a circle with centre at $C(x_0, y_0)$ and radius r. Given three points P_1, P_2 and P_3, with coordinates (x_1, y_1), (x_2, y_2) and (x_3, y_3) respectively, lying on the circumference of a circle, it is possible to compute the centre coordinates (x_0, y_0) and radius r of the circle using (Cuevas *et al.*, 2012a):

$$x_0 = \frac{\begin{vmatrix} x_2^2 + y_2^2 - (x_1^2 + y_1^2) & 2(y_2 - y_1) \\ x_3^2 + y_3^2 - (x_1^2 + y_1^2) & 2(y_3 - y_1) \end{vmatrix}}{4((x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1))} \tag{2.4}$$

$$y_0 = \frac{\begin{vmatrix} 2(y_2 - y_1) & x_2^2 + y_2^2 - (x_1^2 + y_1^2) \\ 2(y_3 - y_1) & x_3^2 + y_3^2 - (x_1^2 + y_1^2) \end{vmatrix}}{4((x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1))} \tag{2.5}$$

and

$$r = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2} \tag{2.6}$$

b) Triangle

A triangle is a figure formed by connecting three noncollinear points A, B and C with three different line segments ($\overline{AB}$, $\overline{BC}$ and $\overline{CA}$) each of which has two of these points as end points (Leff, 1997). A triangle is shown in Figure 2.2. The points A, B and C are the vertices of the triangle and the three line segments are its sides.

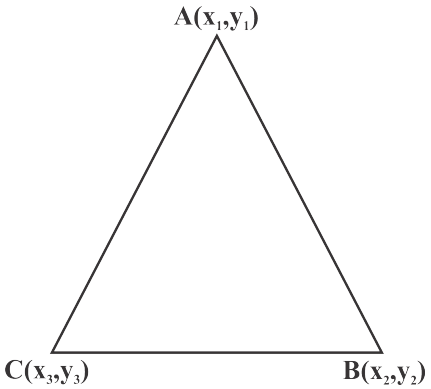


Figure 2.2: Triangle with Vertices at $A(x_1, y_1)$, $B(x_2, y_2)$ and $C(x_3, y_3)$

c) Quadrilaterals

Four points A, B, C and D such that no three points are collinear form a quadrilateral which is defined as the union of four line segments $\overline{AB}$, $\overline{BC}$, $\overline{CD}$ and $\overline{DA}$. The four segments are the sides of the quadrilateral and the points A, B, C and D are the vertices of the quadrilateral (Kay, 2011, Venema, 2013). Figure 2.3 shows a quadrilateral.

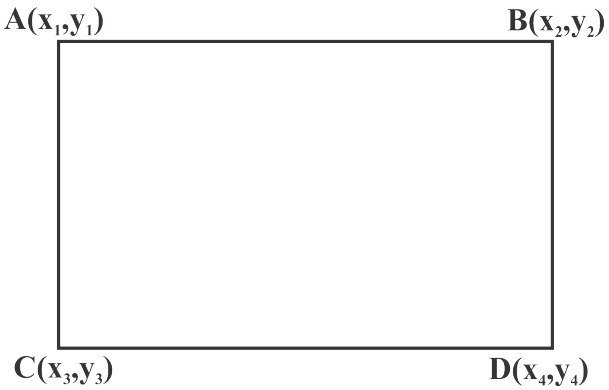


Figure 2.3: Quadrilateral with Vertices at $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ and $D(x_4, y_4)$

The proposed AIS based detection is going to be able to detect the presence of one or more of these shapes in any image, irrespective of their orientation.

2.2.2 Midpoint Line Algorithm (MLA)

The midpoint line algorithm, developed by Bresenham, is a method used to draw lines on raster images. It scan converts lines with positive gradient less than 1 using only incremental integer calculations. Pixel positions along a line path are determined by sampling at unit intervals of x. The process starts from the left (x_0, y_0) of the line and plots the y-positions for successive

x-positions by selecting the y-position closest to the scan-line. The process is summarized thus (Hearn and Baker, 1997)

1. Input the two (2) line end-points, storing the left end-point in (x_0, y_0).
2. Plot the point (x_0, y_0).
3. Compute the constants Δx, Δy, $2\Delta y$ and $(2\Delta y - 2\Delta x)$ and get the first value for the decision parameter as:

$$p_0 = 2\Delta y - \Delta x \tag{2.7}$$

4. At each x_k along the line, starting at $k = 0$, test if $p_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and:

$$p_{k+1} = p_k + 2\Delta y \tag{2.8}$$

   Otherwise, the next to plot is $(x_k + 1, y_k + 1)$ and:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x \tag{2.9}$$

5. Repeat step 4. $(\Delta x - 1)$ times.

2.2.3 Midpoint Circle Algorithm (MCA)

The MCA calculates pixel locations along a circle's circumference starting at $(x_0, y_0 + r)$ for first octant using integer additions and subtractions. The other pixel positions for the remaining seven (7) octants are computed by determining their corresponding symmetry points. The steps are summarized thus (Hearn and Baker, 1997):

1. Input radius r and circle center (x_0, y_0) and obtain the first point on the circumference as $(x_0, y_0 + r)$.
2. Calculate the initial value of the decision parameter as $p_0 = 1 - r$.
3. At each x_k position, starting at $k = 0$, test if $p_0 < 0$, the next point along the circle is $(x_k + 1, y_k)$ and:

$$p_{k+1} = p_k + 2x_{k+1} + 1 \tag{2.10}$$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1} \tag{2.11}$$

4. Determine symmetry points in the other seven (7) octants.

5. Move each calculated pixel position (x, y) onto the circular path centered at (x_0, y_0) to plot the coordinate values:

$$x = x + x_0 \tag{2.12}$$

$$y = y + y_0 \tag{2.13}$$

6. Repeat step 3 through 5 until $x \geq y$.

2.2.4 Canny edge detector

Canny Edge detector is one of the standard edge detection algorithms. It has five (5) major algorithmic steps(Canny, 1986):

1. The image is smoothed or blurred using Gaussian filter to remove noise.

2. The intensity of gradients are computed. Edges appear where the gradients of the image has large magnitudes.

3. Suppression of gradients with low magnitudes.

4. Double thresholding is done to determine potential edges.

5. The edges are then tracked by hysteresis, eliminating all edges not connected to probable edge points.

2.2.5 Harris corner detector

Corners can easily be recognized by looking at intensity values within a small window in the entire image. When shifting the window in any direction yields a large change in appearance,

then a corner exists within that region. Harris corner detector gives a computational approach for determining corner points based on this windowing process by applying the change intensity for the shift $[u, v]$ (Harris and Stephens, 1988):

$$E(u,v) = \sum_{x,y} w(x,y) \left[I(x+u, y+v) - I(x,y) \right]^2 \tag{2.14}$$

Where $w(x, y)$ is the window function, $I(x+u, y+v)$ is the shifted intensity and $I(x, y)$ is the intensity.

2.2.6 Gaussian filtering of images

Gaussian filter is used in image processing to reduce image noise. It is the result of transforming each pixel in the image with a Gaussian function. When applied to 2-dimensional image, it produces a surface of concentric circle contours having a Gaussian distribution from the centre point. The Gaussian function $G(x, y)$ is (Neycenssac, 1993):

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.15}$$

Where x, y and σ are the horizontal distance from origin, vertical distance from origin and standard deviation of the Gaussian distribution respectively.

2.2.7 Artificial Immune System (AIS)

Artificial Immune System (AIS) is a term used to cover all the efforts to develop computational models inspired by biological (vertebrate) immune system (Dasgupta, 1999). The immune system has many computationally interesting properties that make it suitable for solving important aspects of computation. These properties include parallel and distributed adaptive system, robustness, pattern recognition, feature extraction and learning memory (Greensmith *et al.*, 2010, Dasgupta *et al.*, 2011, Cuevas *et al.*, 2012a). Unlike other biological inspired algorithms which have an archetypal algorithm, the AIS has several algorithms which model different aspects of the immune system. The three most popular AIS algorithms are: negative selection, immune network and clonal selection algorithm (Greensmith *et al.*, 2010).

2.2.7.1 *Negative selection*

Negative selection algorithm is modeled off the T-cell maturing process that happens in the thymus, where T-cells that recognize self cells are eliminated before the rest are deployed into the immune system to recognize and attack intruding pathogens (Ji and Dasgupta, 2007). It is based on the pseudo-random generation of antibodies (T-cells) and training them by presenting self patterns to them. Antibodies that detect patterns during training are removed from the repertoire of cells. T-cells that survive the training process are then elevated to mature cells or antibodies and are used to detect nonself patterns or antigens (Greensmith *et al.*, 2010).

2.2.7.2 *Immune network (Idiotypic network)*

Immune network is based on the proposal that B-cells are capable of achieving immunological memory by the existence of a mutually reinforcing network. They both stimulate and suppress connected cells to regulate the over stimulation of B-cells in order to maintain a stable memory (Dasgupta *et al.*, 2011). The theory suggests that antibodies continue communicating even in the absence of antigens, which produces continual change of concentrations. The network continually adapts itself, maintaining a steady state that reflects the global results of interacting with the environment (Greensmith *et al.*, 2010). Although the theory has no immunological backing, it has gained popularity within the AIS due to its ability to produce flexible selection mechanisms, especially in robot navigation (De Castro and Von Zuben, 2000, Luh and Liu, 2008, Challoo *et al.*, 2010).

2.2.7.3 *Clonal Selection Algorithm (CSA)*

Clonal Selection Algorithm (CSA) model the behaviour of B-cells to match and kill an antigen (Garrett, 2005). B-cells produce antibodies of a specific configuration, and their diversity is stimulated upon encounter with a foreign antigen, where the resulting B-cell clones vary the receptor configuration in order to perform a biological local search to find the best fitting receptor (Greensmith *et al.*, 2010). CSA performs the repeated cycle of match, clone, mutate

and replace until a stopping criterion (optimal solution, convergence or maximum number of iterations) is achieved. It is the most widely applied AIS algorithm, with function optimization and pattern recognition being the two (2) major applications for CSA (Dasgupta, 1999, Secker *et al.*, 2003, Freschi *et al.*, 2009, Ulutas and Kulturel-Konak, 2011). It has also been argued to produce robust solutions (Greensmith *et al.*, 2010).

This research implements the CSA for solving the multiple shapes detection problem because of its suitability for pattern recognition. It is robust, can store information of other suboptimal solutions in its memory and convenient for concurrent detection of multiple shapes (antigens).

Following the notations and description used in Cuevas *et al.* (2012a) and Ulutas and Kulturel-Konak (2011), boldfaced capital letters represent matrices while boldfaced small letters represent vectors, what follows is the definition of some key terms in the AIS paradigm.

(i) Antigen: This is the problem to be optimized and its constraints (circle, triangle and quadrilateral detection);

(ii) Antibody: the candidate solutions of the problem (circle, triangle and quadrilateral candidates);

(iii) Affinity: the objective function measurement for an antibody (circle, triangle and quadrilateral matching);

Suppose an antibody is encoded as a vector $\mathbf{a}$, set $\mathbf{I}$ is called the antibody space meaning $\mathbf{a} \in \mathbf{I}$. The antibody population space is thus defined as (Cuevas *et al.*, 2012a):

$$\mathbf{I}^m = \{\mathbf{A} : \mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m), \mathbf{a}_k \in \mathbf{I}, 1 \leq k \leq m\} \tag{2.16}$$

where the positive integer m is the size of the antibody population, $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\}$ is an m-dimensional group of antibodies and $\mathbf{a}$ being a spot within the antibody space $\mathbf{I}$.

The steps taken in CSA as can be seen in Figure 2.4 are:

1. *Initialization*

   A population, $\mathbf{P}_T$, of antibodies is initialized by the generation of pseudo-random candidate solutions. An antibody ($\mathbf{a}_m$) represents the encoding of a candidate solution of the

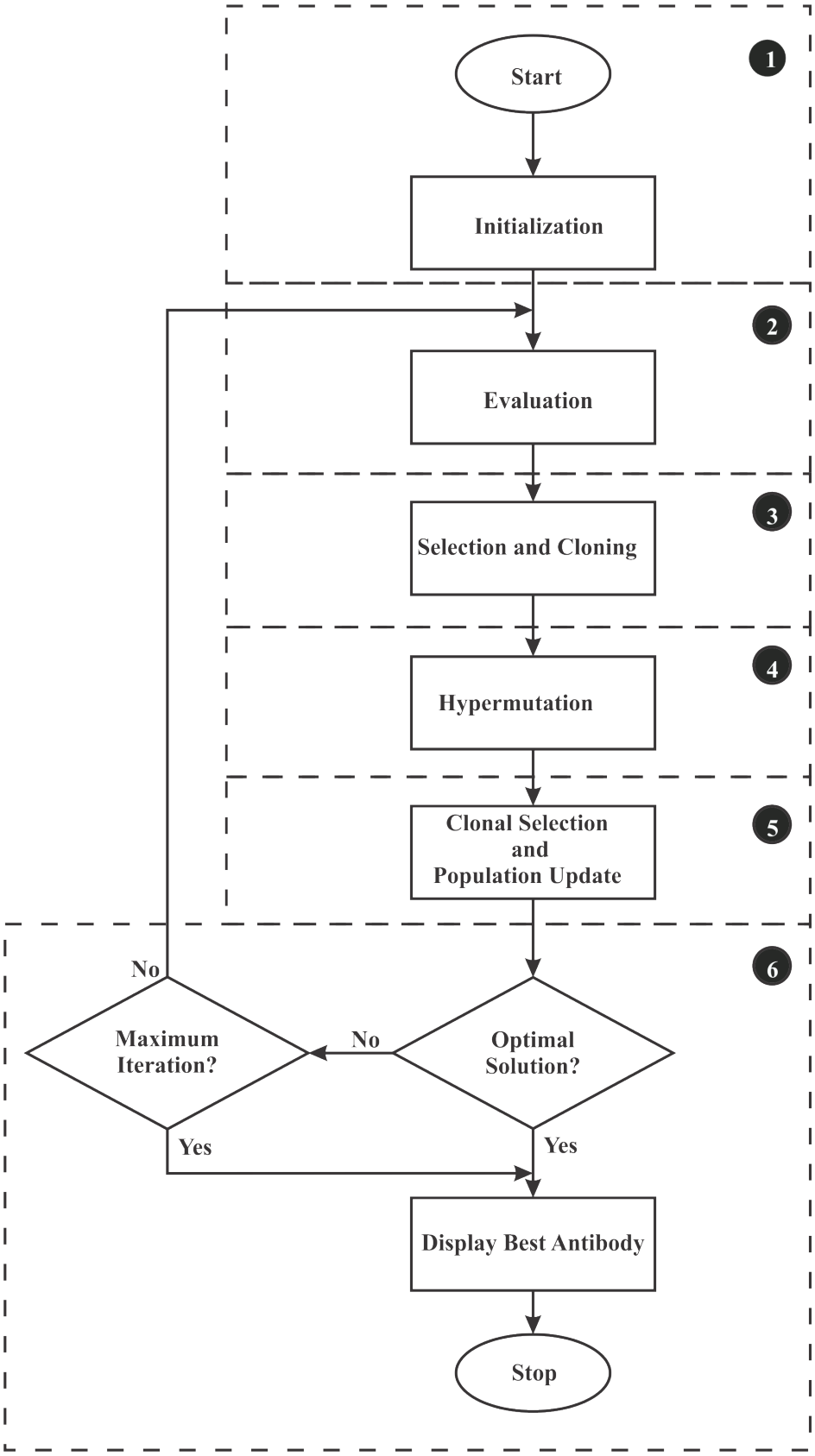


Figure 2.4: The Clonal Selection Algorithm Basic Flow Chart (De Castro and Von Zuben, 2000)

problem. They are represented as binary strings or real numbers depending on the nature of the problem and choice of the programmer (Cuevas *et al.*, 2012a).

$$\mathbf{P}_T = \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\} \tag{2.17}$$

2. *Evaluation*

   The affinity (match) of each antibody (candidate solution) with the antigen (image) is computed. Examples of affinity measure include Hamming, Euclidean and Manhattan distances between antibodies and antigens (De Castro and Von Zuben, 1999).

3. *Selection and cloning*

   n best antibodies of the population ($\mathbf{P}_T$) are selected to form a new population ($\mathbf{A}(k)$). A temporary population, $\mathbf{Y}(k)$, of clones of each individual in $\mathbf{A}(k)$ is made in proportion to their affinity with respect to the antigen (Cuevas *et al.*, 2012a).

$$\mathbf{Y}(k) = T_p^c(\mathbf{A}(k)) = [T_p^c(\mathbf{a}_1(k)), T_p^c(\mathbf{a}_2(k)), ..., T_p^c(\mathbf{a}_n(k))] \tag{2.18}$$

4. *Hypermutation*

   Each antibody clone in $\mathbf{Y}(k)$ is mutated in inverse proportion to its affinity to the antigen to form a new population $\mathbf{Z}(k)$. The higher the affinity, the less likely the antibody will undergo mutation.

5. *Clonal selection and population update*

   The best individuals from $\mathbf{Z}(k)$ and $\mathbf{A}(k)$ are selected to compose a new memory set $\mathbf{M} = \mathbf{A}(k+1)$. The selection process is done in such a way that an antibody is replaced by its mutated clone only when the mutation results in a higher affinity antibody. Random novel antibodies ($\mathbf{P}_r$) are added to the new memory cells ($\mathbf{M}$) to add diversity and build a new $\mathbf{P}_T$ (Cuevas *et al.*, 2012a):

$$\mathbf{P}_T = \mathbf{M} + \mathbf{P}_r \tag{2.19}$$

6. *Stopping criteria*

   The algorithm returns to step 2 if no stopping criterion is met. The stopping criteria used include finding the optimal solution, convergence of the algorithm or maximum number

of iterations. If a stopping criterion is met, the algorithm gives as output the antibody with the highest affinity which represents the best solution found by the algorithm.

2.2.8 Performance evaluation

This section discusses the performance metrics to be used in evaluating the performance of the CSA.

2.2.8.1 *Mean Squared Error (MSE) and Peak Signal to Noise Ratio (PSNR)*

Several statistical tools are available for evaluating the effectiveness of an algorithm. One of such is the computation of the Mean Squared Error (MSE). It provides a quantitative score that describes the degree of similarity/fidelity between the original data and the result gotten by the algorithm (Wang and Bovik, 2009). Given an original data (ground truth), x_i, and result from the algorithm of, y_i, the MSE is computed thus (Wang and Bovik, 2009):

$$MSE(\mathbf{x},\mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2 \tag{2.20}$$

A small value of MSE indicates high similarity between the original signal and the algorithm's result. This in turn signifies a high-performing algorithm. In the literature of image processing, MSE is often converted into a Peak Signal to Noise Ratio (PSNR) measure (Korhonen and You, 2012).

$$PSNR = 10\log_{10}\frac{L^2}{MSE} \tag{2.21}$$

PSNR is measured in decibels and used as a measure of visual quality. The greater the PSNR, the better the visual quality (Korhonen and You, 2012).

2.2.8.2 *False positive and false negative rates*

The performance of an algorithm (especially when the outcome is the detection of the presence/absence of a feature as in a shape detector) can also be measured by computing the sensitivity, specificity, Positive Predictive Value (PPV), Negative Predictive Value (NPV), False

Positive Rate (FPR) and False Negative Rate (FNR) on the test data. These values are computed based on the False Positive (FP), False Negative (FN), True Positive (TP) and True Negative (TN) results from testing the algorithm. With respect to detection of desired shape(s) on test data, these terms can be defined as:

1. *False Positive (FP)*: is the number of shapes detected when the desired shapes are absent.
2. *False Negative (FN)*: is the number of desired shapes that the algorithm fails to detect.
3. *True Positive (TP)*: is the number of desired shapes that have been detected.
4. *True Negative (TN)*: is the number of undesired shapes correctly classed as undesired by the algorithm.
5. *Sensitivity*: also called True Positive Rate (TPR), is a measure of how well the algorithm correctly identifies the desired shape (Hoffrage *et al.*, 2000).

$$Sensitivity = \frac{TP}{TP + FN} \tag{2.22}$$

6. *Specificity*: is the measure of how well the algorithm correctly identifies the absence of the desired shape (Hoffrage *et al.*, 2000).

$$Specificity = \frac{TN}{TN + FP} \tag{2.23}$$

7. *Positive Predictive Value (PPV)*: also known as precision, is the probability that a positive prediction is correct (Hoffrage *et al.*, 2000).

$$Precision = \frac{TP}{TP + FP} \tag{2.24}$$

8. *Negative Predictive Value (NPV)*: is the probability that a negative prediction is correct (Hoffrage *et al.*, 2000).

$$NPV = \frac{TN}{TN + FN} \tag{2.25}$$

9. *False Positive Rate (FPR)*: also called type I error or false alarm rate, is the measure of how much the algorithm fails to detect the absence of the desired shape (Hoffrage *et al.*, 2000).

$$FPR = 1 - Specificity = \frac{FP}{FP + TN} \tag{2.26}$$

10. *False Negative Rate (FNR)*: measures the failure rate of the algorithm to detect the presence of the desired shape (Hoffrage *et al.*, 2000).

$$FNR = 1 - Sensitivity = \frac{FN}{TP + FN} \tag{2.27}$$

More information on these terms can be found in Hoffrage *et al.* (2000), McDonald (2009) and Witten *et al.* (2011). These parameters have been used as performance metrics for research in fields of astronomy (Fressin *et al.*, 2013), medicine (Matern *et al.*, 2007, Wolfer *et al.*, 2008) and engineering and computing (Aman *et al.*, 2009, Cheng-Bin, 2009, Vennila *et al.*, 2014) with design objective being the minimization of FPR and/or FNR, for example Matern *et al.* (2007).

2.2.8.3 *Mean Absolute Error (MAE)*

This is the average of the absolute error. The MAE represents the average magnitude of the total error of the detection/prediction. A value of MAE less than 1 in this research work represents a sub pixel accuracy in detecting the desired shape, while values greater than 1 indicates a higher dissimilarity between the expected result and that returned by the algorithm. MAE is computed thus (Willmott and Matsuura, 2005):

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |e_i| \tag{2.28}$$

Where $e_i = x_i - y_i$ is the error or mismatch between the expected value (x_i) and the observed value (y_i), and n is the number of test points.

2.2.8.4 *Error score*

A final performance measure, for reasons of comparison with results from literature, is to compute the error score (E_s). The E_s was used as a measure of the accuracy of detections for the binary coded CSA in Cuevas *et al.* (2012a) and LA in Cuevas *et al.* (2012b). It is computed thus for circles (Cuevas *et al.*, 2012a):

$$E_{sC} = 0.1(|x_A - x_B| + |y_A - y_B|) + 0.05(|r_A - r_B|) \tag{2.29}$$

Where the expected (ground truth) radius and centre coordinates are r_A and (x_A, y_A) respectively, while the radius and centre coordinates of circle detected by the algorithm are r_B and (x_B, y_B) respectively. Extending the computation of E_s to quadrilaterals (E_{sQ}) and triangles (E_{sT}), only the coordinates of the vertices are used as shown in Equations 2.30 and 2.30 respectively.

$$E_{sdq} = 0.1(|x_{1A} - x_{1B}| + |y_{1A} - y_{1B}| + |x_{2A} - x_{2B}| + |y_{2A} - y_{2B}| + |x_{3A} - x_{3B}| + |y_{3A} - y_{3B}| + |x_{4A} - x_{4B}| + |y_{4A} - y_{4B}|) \tag{2.30}$$

$$E_{sdt} = 0.1(|x_{1A} - x_{1B}| + |y_{1A} - y_{1B}| + |x_{2A} - x_{2B}| + |y_{2A} - y_{2B}| + |x_{3A} - x_{3B}| + |y_{3A} - y_{3B}|) \tag{2.31}$$

2.3 Review of Similar Works

Various algorithms for shape detection have been proposed in literature. Some publications on the detection of circular and quadrilateral shapes are reviewed as follows:

Dasgupta *et al.* (2010) developed an Adaptive Bacterial Foraging Optimization Algorithm (ABFOA) for automatic detection of circles on both synthetic and natural images with varying range of complexity and noise level. Each bacterium in the swarm was a vector representing the centre coordinates and radius of a candidate circle to be detected on the image. The algorithm was able to detect single and multiple circles in an image by following the four steps in BFOA - chemitaxis, swarming, reproduction and elimination. An adaptive scheme for the chemotactic step was implemented to reduce the elimination problem of BFOA when a bacterium approached the optimum solution. Simulation results were compared with BFOA, Random Hough Transform (RHT) and Genetic Algorithm (GA) for circle detection. The proposed ABFOA method was able to detect circles faster, with less errors. However, ABFOA was not able to detect multiple circles in a single run of the algorithm. To detect multiple circles, the algorithm had to be run several times depending on the number of circles, and already detected

circles had to be masked before each run. Also, ABFOA was suitable for detecting only circular shapes.

In **Ramirez *et al.* (2011)**, a Genetic Algorithm (GA) approach for detecting quadrilateral shapes was proposed. They encoded four edge points from the image as the vertices of a candidate quadrilateral. The edge points were extracted from the image using Prewitt edge operator. A chromosome or individual in the GA population represented a candidate quadrilateral to be detected on the image. To avoid degenerate shapes, they included a collinearity check to avoid selecting three (3) points resting near a single line. They also used a GA sharing based approach, which ensured that there were no similar individuals in the population to avoid several individuals representing a single quadrilateral. Their algorithm was able to detect quadrilaterals under perspective transformation, quadrilaterals in images corrupted by Gaussian noise, quadrilaterals present in real images, multiple and hand-drawn quadrilaterals. However, their approach involved calculating the Manhattan distance between a tested pixel on the candidate quadrilateral and an actual edge on the image, which can be computationally intensive with increase in number of pixels to be tested, population of individuals and number of generations or iterations. However, the authors did not consider the detection of circular and triangular shapes.

Cuevas *et al.* (2012a) proposed a method for the automatic detection of multiple circular shapes over complicated and noisy images based on the Clonal Selection Algorithm (CSA) of the Artificial Immune System (AIS). A candidate circle (antibody) was represented as a 22-bits binary string formed by random selection of three non-collinear edge points in the edge image of the scene; and the matching function (affinity measure) was used to measure the existence of a candidate circle on the edge map (antigen). The CSA evolution of the candidate circles (antibody population) was guided by this matching function such that the antibody with the best candidate circle could fit into an actual circle present in the image. The CSA-memory was then analysed at the end of the evolution or optimization process to find other local minima which represented other circles present in the image. The proposed method was therefore able to detect multiple circles, including overlapping circles, within a single execution of the algorithm. Comparisons with other circle detection algorithms based on GA and BFOA on

synthetic and natural images demonstrated the superior performance of the proposed approach. The authors however restricted the implementation to detections of only circles.

Cuevas *et al.* (2012b) proposed a Learning Automata (LA) algorithm for multiple circle detection. Though LA converges to a single optimum value, further analysis of the probability distribution of the algorithm can reveal other local minima, which could represent more circles present in the image. This property made the LA implementation detect multiple circles during a single run of the algorithm. As has been done by other authors, the edges in the image were first extracted to give an edge-only image. The coordinates of the edge pixels were then stored in a vector. The LA algorithm made use of 5% of the total edge pixels to generate actions (candidate circles) - a single action was formed using three randomly selected non-collinear edge pixels, and a matching function for the reinforcement signal. The learning rule used was the linear reward/inaction scheme. The LA operated by selecting an action probabilistically from the set of actions. After assessing the performance of the action, a reinforcement signal was evaluated based on its performance. The internal probability distribution was then updated - actions with desirable performance were reinforced, under-performing actions were left unchanged. The cycle was repeated until a probability of 1 or maximum iterations was reached. The action with the highest probability represented the best circle detected, while further analysis of lesser probabilities above the threshold represented other circles present in the image. However, they did not extend the algorithm to detect multiple shapes.

The Hough Transform (HT) has been an attractive method for circle detection because it is insensitive to noise and easy to realize in parallel computing. However, the computation time and storage requirement makes it an inefficient method for circle detection. Randomized Hough Transform (RHT) was developed to address these problems. It is however inefficient for detecting multiple circles. **Jiang (2012)** proposed an improved RHT which was efficient for detecting multiple circles. He optimized the methods for determining sample points and finding candidate circles. To improve the random sampling in RHT, the assumption made was that edge points (edge pixels) on circle circumference have more neighbourhood points. Therefore, sampling of points (three points) was done according to the number of neighbourhood points - the more neighbourhood points around an edge pixel, the higher the probability of it being

sampled. To improve detection of candidate circles, only points that fell in the region formed by the vertical circumscribed and inscribed quadrilaterals were used in the evidence-collection stage. All points in the image were stored in an edge set. If after the run of the algorithm, a candidate circle was affirmed as a true circle, all points lying on its circumference were deleted from the edge set. This process made it possible for the algorithm to detect other circles present in the image during the next run. Results from the proposed method proved outperformed the RHT. However, to detect multiple circles, the algorithm had to be run once per circle, and the neighbourhood points calculated after each circle detection.

Akinlar and Topal (2013) proposed a real-time circle detector with a false detection control, EDCircles, which was able to detect circles and near-circular ellipses. The first step was to extract edges in the image using Edge Drawing Parameter Free (EDPF) algorithm. All closed edge segments were processed by fitting them into circles or ellipses to form candidate solutions. If both processes failed for any closed edge segment, it was further processed along with other non-closed edge segments. The edge segments were turned into line segments using EDLines. The computed lines were then converted into arcs, which were then combined to generate other candidate circles and ellipses. These candidate circles and ellipses were then validated by the Helmholtz principle, which eliminated false detections and left only valid circles and near-circular ellipses. The performance of EDCircles was assessed using both synthetic and natural images containing circular and elliptic objects. EDCircles, however, is not capable of detecting triangular and quadrilateral shapes.

Lu and Yu (2013) proposed an AIS-based approach for multiple circle detection. They took advantage of the robust nature of the AIS algorithm for detecting non-ideal circles (beads with cancer cells). The antigens were encoded as edge segments, these were formed by traversing edge pixels connected to each other using 8-point connectivity. A segment or antigen was therefore a chain index of edge pixels directly or indirectly connected to each other. An antibody was formed from the antigen (segment) by selecting three random edge points from the antigen and forming a circle that passed through these points. The affinity between the antibody and antigen was the hamming distance between them. Real value representation of antibodies was used, and when a circle was detected, the set of pixels lying on its circumference were

deleted, and the antibody that detected the circle was added to the memory cells. To prevent multiple memory cells from representing a single antigen, new memory cells were compared with existing memory cells, and the antibody with the higher affinity was retained as a memory cell. According to the authors, deleting detected edge pixels simulates the activity of antibodies destroying antigens. To demonstrate the effectiveness of their proposed approach for detecting multiple non-ideal circles, they used a dataset of 85 microscopic images with thousands of PLZ4 beads to test their algorithm. However, deleting edge pixels making on the circumference of the detected circle can result in removal of pixels shared by the detected circle and other circles.

In **Li (2014)**, a geometric framework for detecting rectangular shapes irrespective of orientation was proposed. The proposed method focused on tackling three main issues in shape detection. They were: outliers - image points lying outside or inside the shape boundary; open shape - set of points that form an incomplete shape; and fragmentation - sets of points from which a shape cannot be deduced. Their approach started by introducing the channel scale space of RGB images, aimed at enhancing the chance of detecting good rectangle candidates. Then specific algorithms were developed to address the three difficulties individually. The algorithm to remove outliers in a candidate rectangle used spatial likelihood. The open shape problem was solved by determining the four vertices of the rectangle using either Bounding-Box or Diagonal-First method. The fragmentation issue was then tackled by algorithm to check if candidate shapes were complimentary or not. Complimentary candidate shapes were synthesized. False positive from the above steps were eliminated by the introduction of interestness measure of a rectangular shape by the integration of imbalanced points. The candidate shape with the highest interestness was output as the detected rectangle. The algorithm was used for the detection of license plates and road sign images. A comparison with other works based on analysis of geometry information revealed the method had better detection, but at a slower rate. Also, the author only considered the detection of quadrilateral shapes.

In **De Marco *et al.* (2015)**, a randomized circle detection with isophotes curvature analysis was proposed. Isophotes are curves connecting pixels in the image with equal intensity. They have been demonstrated to have same shape independent of rotation and varying lighting conditions,

these reliable/consistent features made isophotes better detector attributes than intensities or gradients. The proposed approach started by discarding isophotes with too high or too low curvature after which isophotes with same curvature were grouped into subsets. Points sampled to form candidate circles were restricted to isophotes in same subgroups. Kernel density based estimation voting process was performed for each candidate circle after which each detected circle parameters were refined with an error linear compensation algorithm in order to provide better fitting with the recognized circle. The approach was tested on several natural images with multiple circular shapes and several levels of noise, and the results showed that the approach greatly reduced the number of edge pixels needed for circle detection. The algorithm also overcame the problem of multiple detection of a single circle. Its performance was superior when compared with those of EDCircles, classical Hough based approach and randomized circle detection (GRCD-R). However, it was slower than EDCircles and incapable of detecting non-circular shapes.

A centre-based clustering algorithm for multiple circle detection was proposed by **Scitovski and Marošević (2015)**. The approach made use of a number of data points which should belong to k circles (partitions). The algorithm iteratively found the optimal k-partitions for the clusters, while using circle fitting algorithms - DIvidingRECTangle (DIRECT) and K-means based Consensus Clustering (KCC) - to form clusters round the detected centres. Davies-Bouldin index, Calinski-Harabasz index and Simplified Silhouette Width Criterion, methods for measuring distance between circles, were used for determining an appropriate number of clusters necessary for a partition. Comparison of the cluster-based approach with the Hough Transform for circle detection on synthetic images revealed its superior performance - it had higher detection rate and required less data points than Hough Transform based detection. According to results presented by the authors, the approach gave sub-pixel accuracy and can be extended for detection of multiple ellipses in images. The approach, however, did not consider the detection of triangular and quadrilateral shapes.

From the reviewed literature, it is obvious that most authors focused on the development of algorithms for detecting a single shape in images. This research, however, implements a multiple shape detection algorithm based on the Clonal Selection Algorithm (CSA). The CSA is

a decentralized and robust algorithm that has been successfully applied to pattern recognition, multi-modal and multi-objective problems with superior results (Ulutas and Kulturel-Konak, 2011). These make CSA an excellent choice for solving the problem of multiple shapes detection.

CHAPTER THREE

MATERIALS AND METHODS

3.1 Introduction

The chapter provides description of the development of the CSA based system capable of detecting multiple circles, quadrilaterals and triangles in an image. The developed system has five (5) major parts as shown in Figure 3.1. The first step is to form a repository of synthetic and real test images. This is followed by the design of the preprocessing steps for the images. The third step is the CSA design while the fourth step analyses the CSA antibody pool to extract all distinct detections. Finally, the results are superimposed on the test image to give a visual representation of the system's output.

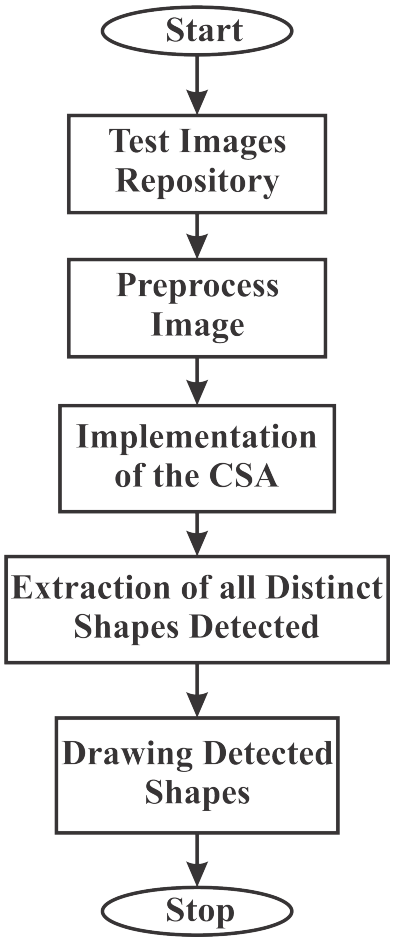


Figure 3.1: Flow Chart of the Five Major Parts of the Developed System

3.2 Test Images Repository

The repository is made up of five (5) synthetic images generated using Matlab and six (6) real images captured with the aid of a digital camera. The subsections that follow describe the process of acquiring these images.

3.2.1 Generating synthetic images with MATLAB

Two functions, ShapePoints (Appendix A) and ShapeMaker (Appendix B) implement the generation of five (5) image scenes

3.2.1.1 *Shape points*

ShapePoints function saves all the points needed for drawing/generating the required shapes in the five (5) synthetic images namely: Ptc, Ptq, Ptt, PtMxd1 and PtMxd2. For a particular image, the points and type of shapes are stored as $n \times 2$ cell, where n is the number of shapes. Four shape types are defined in the function: circle (c), triangle (t), quadrilateral (q) and hexagon (p). All points (Pt) needed to draw Shapes in an image is:

$$\text{Pt} = \begin{Bmatrix} \text{'c'} & \text{Pointsc} \\ \text{'q'} & \text{Pointsq} \\ \text{'t'} & \text{Pointst} \\ \text{'p'} & \text{Pointsp} \end{Bmatrix} \tag{3.1}$$

Where Pointsc, Pointsq, Pointst and Pointsp are matrices of points needed to fully define (draw) the circle, quadrilateral, triangle and polygon respectively. Table 3.1 shows the number and type of shapes contained in the developed synthetic images.

Table 3.1: Shapes Contained in each Synthetic Image

	Ptc	Ptq	Ptt	PtMxd1	PtMxd2
Circle	6	0	0	2	2
Quadrilateral	0	6	0	2	2
Triangle	0	0	6	2	2
Polygon	0	0	0	0	1

1. Circle

   As mentioned in section 2.2.1, a circle can be fully defined by the coordinates of it's centre, (x_0, y_0), and radius, r. Therefore, Pointsc is an $m \times 3$ matrix where m is the number of circles present.

   For an image with no circles, Pointsc is an empty matrix.

$$\text{Pointsc} = [\,] \tag{3.2}$$

   For an image with m circles,

$$\text{Pointsc} = \begin{bmatrix} x_{01} & y_{01} & r_1 \\ x_{02} & y_{02} & r_2 \\ \vdots & \vdots & \vdots \\ x_{0m} & y_{0m} & r_m \end{bmatrix} \tag{3.3}$$

   Table 3.1 shows the number of circles defined in the five (5) synthetic images.

2. Quadrilateral

   From section 2.2.1, the coordinates of the four vertices of a quadrilateral are needed in order to successfully draw the quadrilateral. Pointsq encodes a quadrilateral by representing the start and end coordinates of each of the four lines that bound the quadrilateral as four rows of a matrix.

   Therefore, for one quadrilateral,

$$\text{Pointsq} = \begin{bmatrix} AB \\ BC \\ CD \\ DA \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & x_2 & y_2 \\ x_2 & y_2 & x_3 & y_3 \\ x_3 & y_3 & x_4 & y_4 \\ x_4 & y_4 & x_1 & y_1 \end{bmatrix} \tag{3.4}$$

   Where the vertices A, B, C and D have coordinates (x_1, y_1), (x_2, y_2), (x_3, y_3) and (x_4, y_4) respectively.

   For m quadrilaterals, Pointsq will be $4m \times 4$ matrix. The number of quadrilaterals ShapePoints function defined in each of the five (5) synthetic images are as shown in Table 3.1.

3. Triangle

   The same concept used to define quadrilaterals in Pointsq is used for defining triangles in Pointst. Triangles with vertices A, B and C with coordinates (x_1, y_1), (x_2, y_2) and (x_3, y_3) respectively is a 3×4 matrix in Pointst.

$$\text{Pointst} = \begin{bmatrix} AB \\ BC \\ CA \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & x_2 & y_2 \\ x_2 & y_2 & x_3 & y_3 \\ x_3 & y_3 & x_1 & y_1 \end{bmatrix} \tag{3.5}$$

   For m triangles, Pointst is $3m \times 4$ matrix. Table 3.1 outlines the number of triangles present in the five (5) synthetic images.

3.2.1.2 *Shape maker*

The major roles of ShapeMaker function are to draw the shapes on a white background, save the created image as JPG format and extract the ground truth. The algorithm used to draw the circle is MCA (source code shown in appendix N), while edges of the quadrilaterals, triangles and polygons are drawn by successive calls to the MLA (implemented in appendix O) - one call per edge.

1. Drawing the shapes in image scene

   The ShapeMaker function iterates through all the shapes contained in Pt (Equation (3.1)) and uses MLA (for straight lines) or MCA (for curved lines) to get coordinates of all edge points that make up the shapes. These are then plotted as black spots on the image scene.

   The MCA is invoked with the centre coordinates, (x_0, y_0), and radius, r, of the circle as input arguments. The MCA then returns the xy-coordinates that make up the circumference of the circle. One call of MCA is made per circle, therefore to draw n circles, n calls are made.

   The MLA, however, is called when drawing a line that constitutes an edge of a quadrilateral, triangle or polygon. Input arguments are the xy-coordinates of the starting and

ending points of the line. One call per line segment is made. This means 3 calls of MLA for triangle, 4 calls for quadrilateral and n calls for polygon of n sides.

2. Saving the created image scene

   Once the image scene has been generated, the ShapeMaker function saves the image in 'Images' folder as a JPG file. This is achieved using the inbuilt 'imwrite' function in MATLAB. For example, to save an image, Pt, with the name 'Image1' in JPG format in the 'Images' folder/directory, use:

   imwrite('Images/Image1.JPG',Pt);

3. Extracting ground truth

   The ground truth represents the exact coordinates of the edge pixels used to define the shapes. These are the centre coordinates and radius for circles and locations of vertices for triangles and quadrilaterals.

3.2.2 Acquisition of real images

Real images are acquired with the aid of a Nikon coolpix S9500 digital camera with resolution set to 4896×3672 pixels. Six images were taken: One for real circular shapes; one for triangles; one for quadrilaterals and three images containing the three shapes were captured. The shapes are cut from cardboard papers, and placed on white cardboard which serves as the background.

3.3 Preprocessing Images

The images to test the CSA algorithm passes through a number of stages collectively termed image preprocessing. The image is first imported into MATLAB. The image then passes through two independent processes to extract the edge points to form edge-only image and edge map, and to extract the corner points to create a corner-only image and corner map. PrepImage function, whose source code is shown in appendix P contains the implementation of the preprocessing stage.

3.3.1 Creating edge image and edge map

The flow chart of Figure 3.2 illustrates the steps in creating an edge image. The imported image is first converted to grey scale image. Edges on a grey scale image can be detected using the Canny edge detector by calling the 'edge' function in MATLAB. This returns an edge-only image (EdgeImage variable in the source code) as a logical(binary) matrix where 0 represents absence of an edge point, and 1 represent the presence of an edge point.

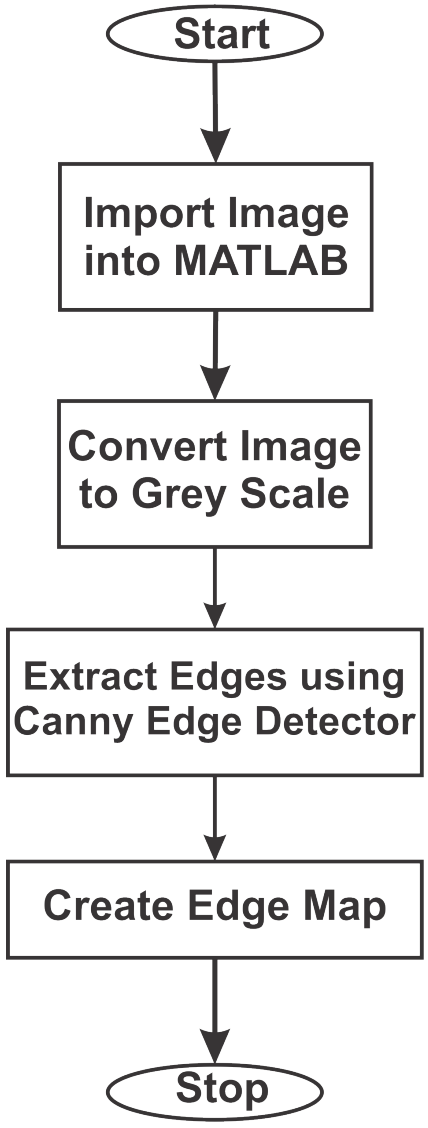


Figure 3.2: Flow Chart for Generating the Edge-only image from an Input Image

The edge map (EdgeMap variable in the source code) is an $n \times 2$ matrix where n is the number of edge points and the elements in each row represent the xy-coordinate of the n-th edge point. It is formed by testing all the pixels in the logical image and saving the (column,row), that is

(x,y), locations where a value of 1 is encountered in the EdgeMap matrix.

3.3.2 Creating corner image and corner map

PrepImage function also contains the implementation for creating a corner-only image (CornerImage variable) and corresponding corner map (CornerMap) for the image. Figure 3.3 is the flow chart for generating the corner-only image from an input image.

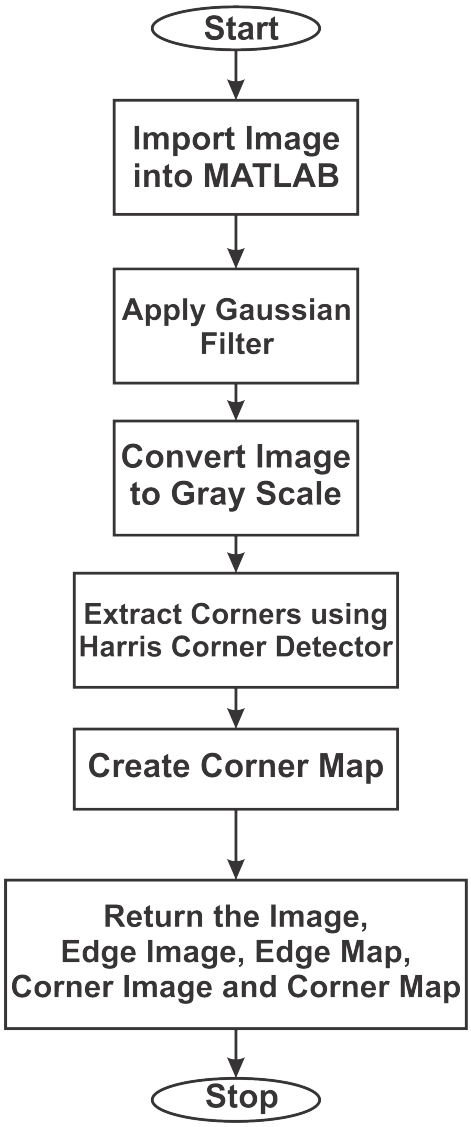


Figure 3.3: Flow Chart for Generating the Corner-only Image from an Input Image

The image is first imported into MATLAB environment and then converted to grey scale. The next step is to pass the image through a Gaussian filter to reduce the noise. This is followed by applying the Harris corner detector, which is implemented as a 'corner' inbuilt function in

MATLAB. The 'corner' takes as input the grey image, and returns the xy-coordinates (column and row locations) of all corners points detected using the Harris corner detector. The xy-coordinates so returned is the corner map. Corner-only image is formed by creating a logical image with same dimension as the grey image and setting all its values to 0 except for locations where corners are present, which are set to 1.

3.4 Implementation of the CSA

This section describes the design decisions made during the implementation of the CSA algorithm for detecting circles, quadrilaterals and triangles.

3.4.1 Antibody representation

An antibody (candidate shape/solution) is represented as a $1 \times n$ vector. $n = 8, 11$ or 13 for circle, triangle or quadrilateral respectively. Candidate circle is formed by randomly selecting three (3) non-collinear edge points (P_1, P_2 and P_3) from the edge map, computing its centre coordinates ((x_0, y_0)), radius (r), and fitness (F). These values are then encoded as a candidate circle antibody as:

$$\begin{bmatrix} P_1 & P_2 & P_3 & 0 & F & r & x_0 & y_0 \end{bmatrix} \tag{3.6}$$

A candidate triangle is formed by randomly selecting three (3) non-collinear corners P_1, P_2 and P_3 from the corner map, extracting the xy-coordinates, (x_1, y_1), (x_2, y_2) and (x_3, y_3) respectively of these points and computing the fitness, F. The triangle antibody is represented as:

$$\begin{bmatrix} P_1 & P_2 & P_3 & 0 & F & x_1 & y_1 & x_2 & y_2 & x_3 & y_3 \end{bmatrix} \tag{3.7}$$

Similarly, a candidate quadrilateral is formed by random selection of four (4) non-collinear corners P_1, P_2, P_3 and P_4 from the corner map, extracting the xy-coordinates, (x_1, y_1), (x_2, y_2) , (x_3, y_3) and (x_4, y_4) respectively of these points and computing the fitness, F. A candidate quadrilateral is represented thus:

$$\begin{bmatrix} P_1 & P_2 & P_3 & P_4 & 0 & F & x_1 & y_1 & x_2 & y_2 & x_3 & y_3 & x_4 & y_4 \end{bmatrix} \tag{3.8}$$

3.4.2 Collinearity check

The collinearity check (appendix H) has been incorporated to prevent selecting three (3) points that can be joined by a straight or nearly straight line. Three (3) collinear points result in invalid/degenerate shapes. A threshold, d_{min} has been set so that the perpendicular distance of a third point from the line joining any two (2) points is equal or greater than the threshold, d_{min}. Consider three points shown in Figure 3.4, the perpendicular distance, d_1, of point P_1 from the straight line joining points P_2 and P_3 can be computed using Equation 3.9 (Weisstein, 2002b):

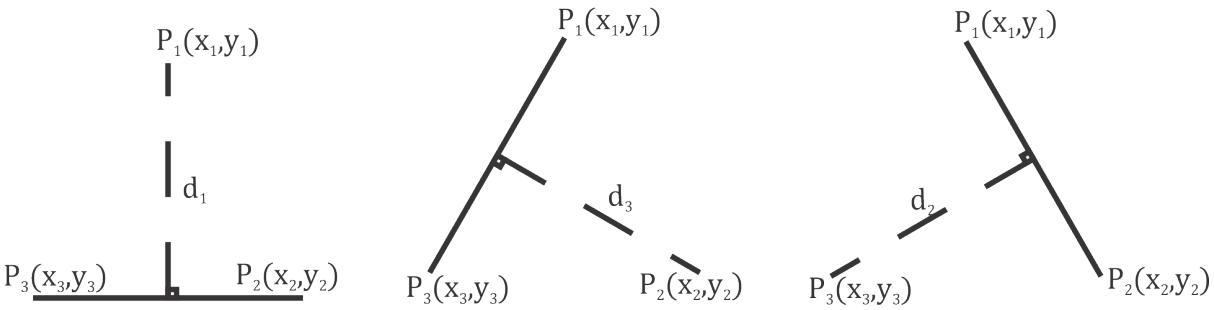


Figure 3.4: Perpendicular Distance between a Point and a Line

$$d_1 = \frac{|([x_3, y_3, z_3] - [x_2, y_2, z_2]) \times ([x_2, y_2, z_2] - [x_1, y_1, z_1])|}{|[x_3, y_3, z_3] - [x_2, y_2, z_2]|} \tag{3.9}$$

For the three (3) random points P_1, P_2 and P_3 to be accepted as valid non-collinear points, d_1, d_2 and d_3 must all be greater than the threshold, d_{min}. This system of collinearity bound simultaneously imposes a minimum dimension for the shape. d_2 and d_3 are computed in similar fashion. Note that the z component is 0 since the focus is on 2-dimensional image scenes.

With d_{min} set to 25 pixels, the minimum acceptable dimension of triangles, circles and quadrilaterals are described next.

3.4.2.1 *Minimum dimension for triangles*

From Figure 3.5, for A, B and C to be accepted as vertices of a valid/acceptable triangle candidate, d_1, d_2, and d_3 must be greater than or equal to d_{min}. Minimum dimension results when

$d_1 = d_2 = d_3 = d_{min}$. This will form an equilateral triangle where $|AB| = |BC| = |CA| = a$. There is an already established relationship between a and d_{min} for equilateral triangles (Gantert, 2008).

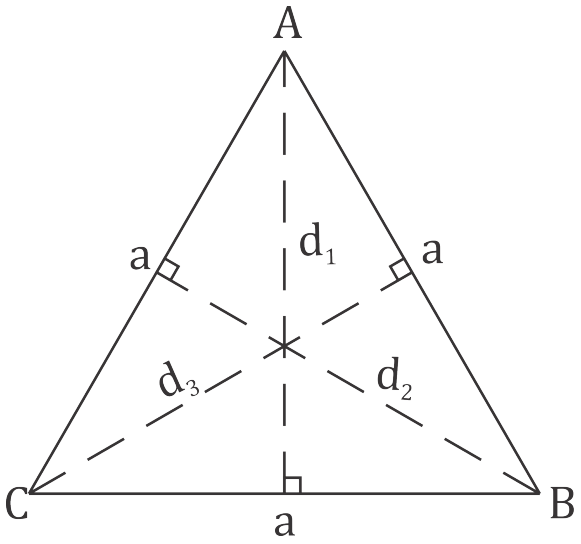


Figure 3.5: Minimum Triangle Dimension

$$a = \frac{2}{\sqrt{3}} d_{min} \tag{3.10}$$

For $d_{min} = 25$ pixels,

$$a = \frac{2}{\sqrt{3}} \times d_{min} = 28.87 \approx 29 \text{ pixels}$$

The smallest acceptable triangle occupies 29×25 pixels. This is 0.29% of 500×500 pixels synthetic image scene.

3.4.2.2 *Minimum dimension of circle*

For a circle, the three (3) valid points, when $d_1 = d_2 = d_3 = d_{min}$, form a circumscribed equilateral triangle (Figure 3.6).

The relationship between circle's radius and altitude of inscribed triangle is (Gantert, 2008):

$$\begin{aligned} r =& \frac{2\, d_{min}}{3} \\ r =& \frac{2 \times 25}{3} = 16.67 \approx 17 \text{pixels} \end{aligned} \tag{3.11}$$

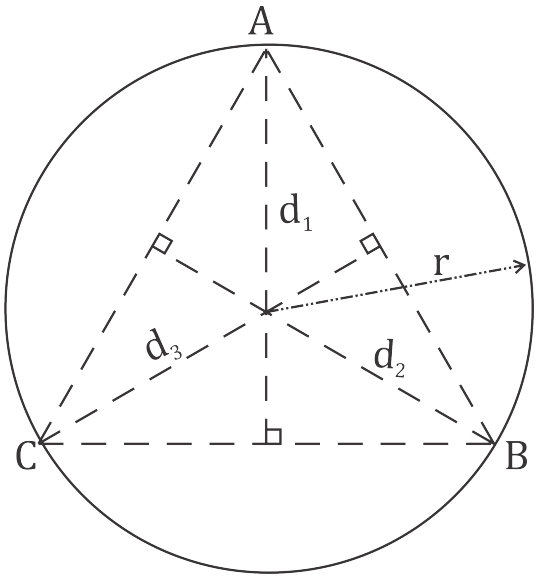


Figure 3.6: Minimum Circle Dimension

The circle occupies 34×34 pixels (0.46% of 500×500 pixels synthetic image).

3.4.2.3 *Minimum dimension for quadrilaterals*

For a quadrilateral, the perpendicular distance between any two (2) vertices and a line joining the remaining two (2) vertices must be equal to or greater than threshold, d_{min}. The quadrilateral formed when $d_1 = d_2 = d_3 = d_4 = d_{min}$ is a square with each side $|AB| = |BC| = |CD| = |DA| = d_{min}$, as shown in Figure 3.7.

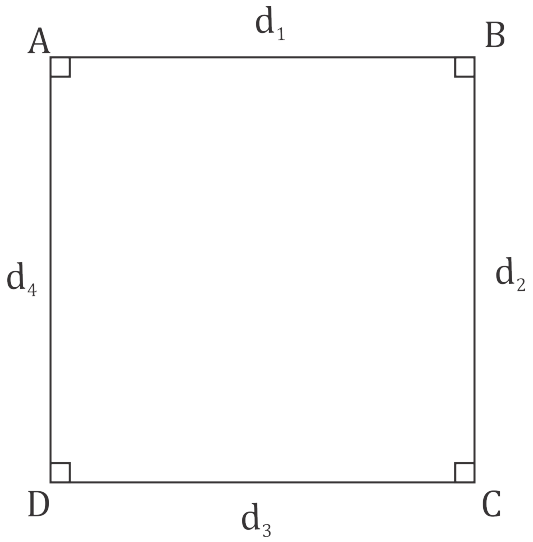


Figure 3.7: Minimum Quadrilateral Dimension

The square occupies 25×25 pixels, making up only 0.25% of the pixels in a 500×500 pixels

image scene.

3.4.3 Vertex arrangement for quadrilateral

Unlike the triangle where there is no need for special rearrangement of its vertices, a wrong arrangement of vertices (or line segment connecting two vertices) can result in invalid shapes. A simple algorithm has been implemented in order to correctly rearrange the vertices that make up the quadrilateral. The algorithm checks the intersection of diagonals that make up the quadrilateral. If the diagonals of a particular arrangement of vertices intersect within the quadrilateral formed (Figure 3.8), then the arrangement is accepted. Otherwise, the vertices are rearranged. Appendix M shows the implementation.

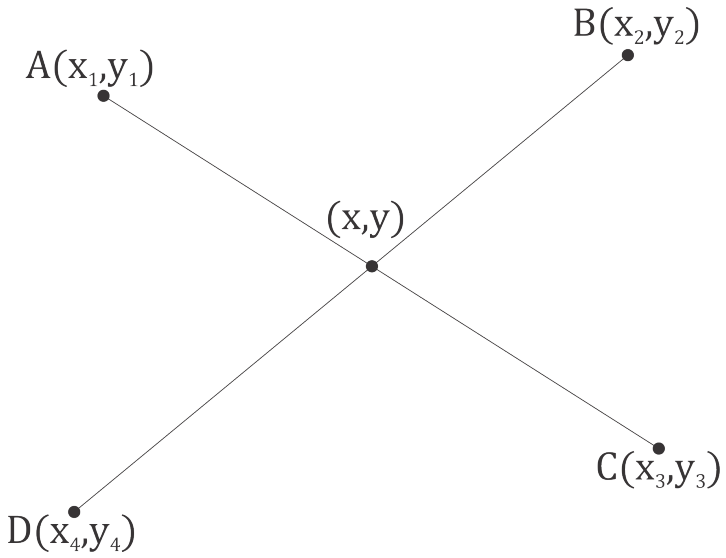


Figure 3.8: Intersection of Two Lines

The xy-coordinates of the point of intersection of two lines (Figure 3.8) is (Weisstein, 2002a):

$$x = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & x_1 - x_2 \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & x_3 - x_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 \\ x_3 - x_4 & y_3 - y_4 \end{vmatrix}} \tag{3.12}$$

$$y = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & y_1 - y_2 \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & y_3 - y_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 \\ x_3 - x_4 & y_3 - y_4 \end{vmatrix}} \tag{3.13}$$

3.4.4 Fitness calculation

The fitness calculation measures the match/similarity between the candidate shape and image scene. The similarity is measured between candidate shapes and edge-only-image of the image scene. This is true for all the shapes - circles, quadrilaterals and triangles.

$$F = \frac{\sum_{i=1}^{N} f_i}{N} \tag{3.14}$$

Where N is the number of edge pixels on the candidate shape, and $f_i = 1$ when edge is present in the edge-only-image and $f_i = 0$ when edge is absent.

3.4.5 Implementation of the CSA algorithmic steps

Appendix F shows the implementation of a single CSA system that is capable of detecting multiple circles, quadrilaterals and triangles. The following subsections describe the design

concepts employed for each CSA step shown in Figure 2.4.

3.4.5.1 *Initialization*

The antibody pool is initialized as a 3 by 2 cell matrix. The first column of Pt describes the type of shape (antigen) and the second column represents the candidate solutions (antibodies) suitable for detecting the shape. For example in Equation 3.15, 'c' for circle candidate solutions AB_c, 'q' for quadrilateral candidate solutions AB_q and 't' for triangle candidate solutions AB_t. This reflects the behaviour of the human immune system where an antibody is best suited for detecting only a subset of antigens. That is, there is no one antibody suitable for detecting all types of antigens, but the immune system is made up of varieties of antibodies whose receptors can detect and kill different kinds of antigens.

$$\mathrm{Pt} = \begin{Bmatrix} \text{'c'} & \mathrm{AB\_c} \\ \text{'q'} & \mathrm{AB\_q} \\ \text{'t'} & \mathrm{AB\_t} \end{Bmatrix} \tag{3.15}$$

Consistent with the work of (Cuevas *et al.*, 2012a), given that there are NAb = 120 candidate solutions for each type of shape:

AB_c is an NAb by 8 matrix where each row represents a circle antibody and each column is as described in Equation (3.6)

AB_q is an NAb by 13 matrix where each row is a quadrilateral antibody and each column is as described in Equation (3.8)

AB_t is an NAb by 11 matrix where each row is a triangle antibody and each column is as described in Equation (3.7)

The antibodies for each shape type are stored in a matrix AB with each row representing an antibody.

$$\text{AB} = \begin{bmatrix} Ab_1 \\ Ab_2 \\ Ab_3 \\ \vdots \\ Ab_m \end{bmatrix} \tag{3.16}$$

Where $Ab_i = 1 \times n$ matrix and $i = 1, 2, \ldots, m$. n is the number of columns associated with the candidate shape. $n = 8,\ 11$ or 13 for circle, triangle or quadrilateral respectively as described in section 3.4.1, while m is the population of antibodies.

3.4.5.2 *Evaluation*

The fitness of all antibodies (candidate shapes) are computed as described in section 3.4.4. The computed fitness is then stored as the fifth element on the vector representing the antibody.

The antibodies are then sorted in descending order of fitness using the 'sortrows' inbuilt MATLAB function.

```
AB = sortrows(AB,-5);
```

The best antibodies for each shape type are then chosen as the global best antibody, GBest for each shape. This is achieved in MATLAB by:

```
GBest = AB(1,:);
```

3.4.5.3 *Selection and cloning*

The 100 best antibodies in AB are selected as memory antibodies, Ak. The MATLAB snippet code to do this is:

```
Ak = AB(1:100, :);
```

Cloning of the antibodies in Ak is done in direct proportion to the fitness of the antibody (Cuevas *et al.*, 2012a).

$$q_i = \text{round}\left(N_c \times \frac{F_i}{\sum_{i=1}^{n} F_i}\right) \quad i = 1, 2, 3, \ldots, n \tag{3.17}$$

$N_c = 100$ is the desired number of clones, F_i is the fitness of the i-th antibody, q_i is an integer representing the number of clones for the i-th antibody and round(x) returns x to the nearest integer.

3.4.5.4 *Hypermutation*

All clones are then mutated according to mutation probability α given by (Cuevas *et al.*, 2012a):

$$\alpha_i = e^{(-\rho F_i)} \tag{3.18}$$

ρ is the fixed step. Cuevas *et al.* (2012a) found by study that $\rho = 5$ gives good results.

Mutation is done by replacing each edge/corner pixel used to create the antibody if some randomly generated number (between 0 and 1) is less than mutation probability. With reference to Equations (3.6), (3.7) and (3.8), P_1, P_2, P_3 and/or P_4 are replaced by randomly selecting edge points from the edge map when mutating circles and randomly selecting corner points from the corner map for triangles and quadrilaterals.

3.4.5.5 *Clonal selection and population update*

When the hypermutation of antibody clone(s) of an antibody results in a superior antibody (antibody with higher fitness), that antibody is replaced in Ak. Otherwise, the non-mutated antibody is retained. This is done for all antibodies present in Ak.

Twenty (20) new antibodies, Abnew, are then created to replace the worst antibodies in AB. The new antibody pool is formed by vertical concatenation of the Ak and Abnew.

The evaluation step (section 3.4.5.2) is then performed on the newly created AB.

3.4.5.6 *Stopping criteria*

The CSA algorithm stops and prints out GBest when optimal solution has been found for all shapes. Otherwise, sections 3.4.5.3 to 3.4.5.6 are repeated until a preset maximum iterations of 100 is attained. An optimal solution (detection) occurs when an antibody attains a fitness, $F = 1$, which indicates 100% match.

3.5 Extraction of all Distinct Shapes Detected

Further analysis of the antibody pool, Pt, is done to extract the presence of other instances of circles, triangles and quadrilaterals present in the input image scene. This is done in two (2) steps - eliminating antibodies with low fitness and computing the distinctness of all antibodies with high fitness. Figure 3.9 shows the flow chart of the steps taken to extract all the distinct shapes the CSA algorithm detected. The implementations for these steps are contained in AllDistinctShapes function (Appendix G)

3.5.1 Fitness threshold

To ensure only antibodies (candidate solutions) with high fitness (match) are considered as positive detections, a threshold of 0.9 (90% match) is set. For a shape type with a set of antibodies, AB, a new antibody pool for the shape, known as memory antibodies, MAb, with fitness above threshold of 0.9 can be created with the following MATLAB code snippet:

MAb = AB(AB(:,5) $>$0.9);

3.5.2 Distinctness factor

All duplicates in MAb are eliminated by measuring the similarity of antibodies within MAb. All the distinct shapes extracted from MAb are then stored in a new variable FMAb, which is a list of all the distinct shapes detected.

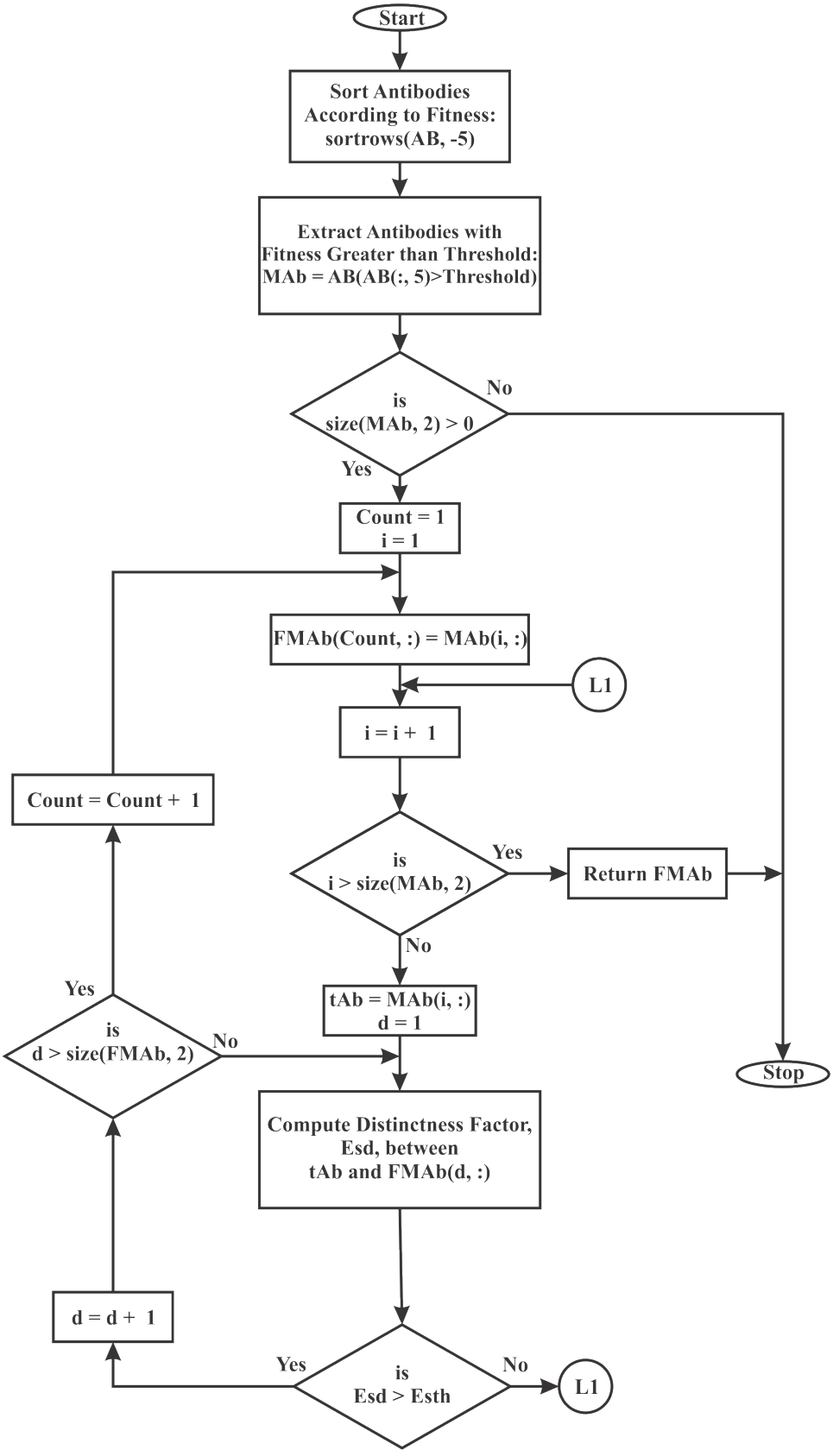


Figure 3.9: Basic Flow Chart of Extracting Multiple Shapes. Count = index of where a distinct shape (candidate solution) should be stored in FMAb, i = index of candidate solution in MAb and d = index of candidate solution in FMAb

The similarity measure used in Cuevas *et al.* (2012a) has been implemented for similarity measure in this project. Distinctness factor between two (2) instances of a shape A and B is computed thus (Cuevas *et al.*, 2012a):

1. Circle

$$E_{sdc} = |x_A - x_B| + |y_A - y_B| + |r_A - r_B| \tag{3.19}$$

Where (x_A, y_A), r_A and (x_B, y_B), r_B are the centres and radii of circle A and B respectively.

2. Triangle

The absolute sum of differences in corresponding vertices is computed.

$$E_{sdt} = |x_{1A} - x_{1B}| + |y_{1A} - y_{1B}| + |x_{2A} - x_{2B}| + |y_{2A} - y_{2B}| + |x_{3A} - x_{3B}| + |y_{3A} - y_{3B}| \tag{3.20}$$

3. Quadrilateral

Similarly, distinctness factor is:

$$E_{sdq} = |x_{1A} - x_{1B}| + |y_{1A} - y_{1B}| + |x_{2A} - x_{2B}| + |y_{2A} - y_{2B}| + |x_{3A} - x_{3B}| + |y_{3A} - y_{3B}| + |x_{4A} - x_{4B}| + |y_{4A} - y_{4B}| \tag{3.21}$$

For the two (2) instances of the shape to be considered different, the distinctness factor E_{sd} has to be greater than the distinctness threshold E_{sth} computed using (Cuevas *et al.*, 2012a).

$$E_{sth} = \frac{r_{max} - r_{min}}{s} \tag{3.22}$$

s is the sensitivity factor. For two (2) shapes that are very similar, a high value of s will have them accepted as two separate shapes while a low/small value of s will regard the two (2) shapes as duplicates. A value of $s = 10$ has been used. r_{max} depends on the size of the image. The minimum between the rows and columns of a 2-dimensional image is used to set the value of r_{max}. r_{min} is the minimum feasible shape dimension as computed in section 3.4.2 for the three shapes. For synthetic image dimension of 500×500 pixels, E_{sth} is computed thus:

1. Circle, $r_{min} = 34$ pixels, which is the minimum accepted diameter for a circle.

$$E_{sth} = \frac{500 - 34}{10} = 46.6 \simeq 47 \text{ pixels} \tag{3.23}$$

2. Triangle, $r_{min} = 29$ pixels

$$E_{sth} = \frac{500 - 29}{10} = 47.1 \simeq 48 \text{ pixels} \tag{3.24}$$

3. Quadrilateral, $r_{min} = 25$ pixels

$$E_{sth} = \frac{500 - 25}{10} = 47.5 \simeq 48 \text{ pixels} \tag{3.25}$$

3.5.3 Drawing detected shapes

Finally, all the distinct instances of the shape(s) detected are drawn on the input image. This will serve as the visual representation of what the algorithm detects. Detected circles are drawn using the MCA while triangles and quadrilaterals are drawn using MLA. The implementation for drawing the shapes is done in DrawShape function which takes as input arguments the detected shapes and the image scene (appendix J).

CHAPTER FOUR

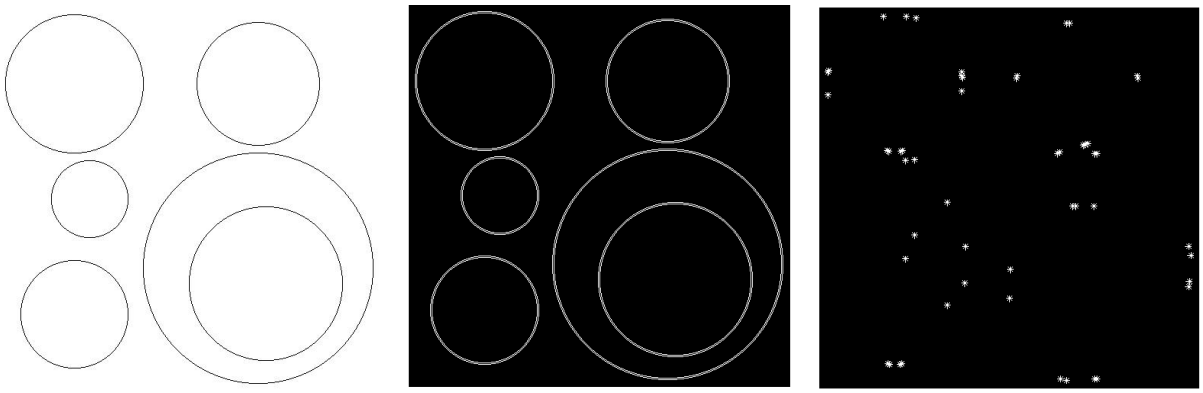
RESULTS AND DISCUSSION

4.1 Introduction

This chapter presents the simulation results and the relevance of such results are discussed. The detections by the algorithm have been plotted on the test images to give a visual perception of the algorithm's output. The Mean Absolute Error (MAE), Mean Squared Error (MSE) and error scores are averaged over 20 runs of the algorithm, PSNR are computed from the MSE values using equation (2.21). FP, FN, TP and TN are summed for 20 runs of the algorithm. A single run of the algorithm is what it has been able to detect after 100 iterations.

4.2 Detecting Desired Shapes on Synthetic Images

The five synthetic test images developed using MATLAB along with their edge-only and corner-only images are shown in Figure 4.1 for multiple circles, Figure 4.2 for multiple quadrilaterals, Figure 4.3 for multiple triangles and Figures 4.4 and 4.5 for mixed shapes.



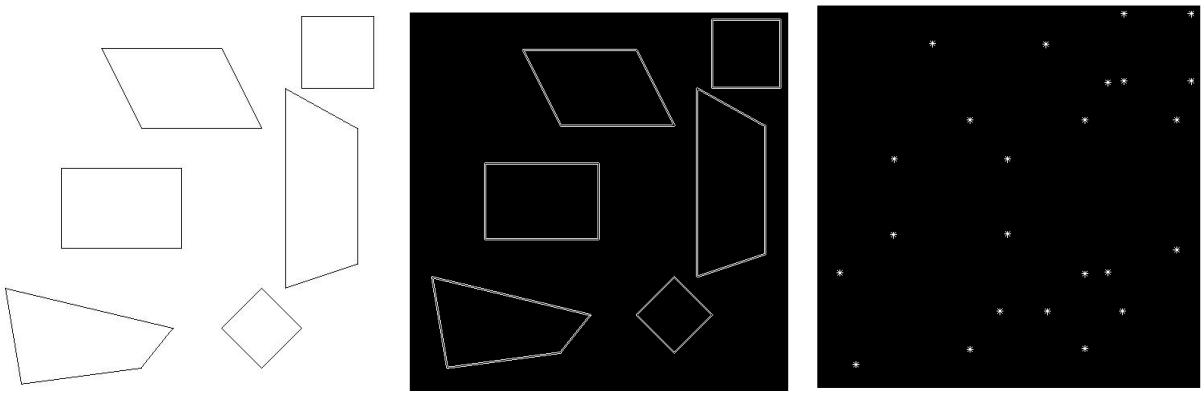
(a) Ptc Image

(b) Ptc Edge Image, Edges = 6104

(c) Ptc Corner Image, Corners = 52

Figure 4.1: Synthetic Image of Multiple Circles

The CSA based systems for detecting circles, triangles and quadrilaterals were tested on these synthetic images and their results presented.

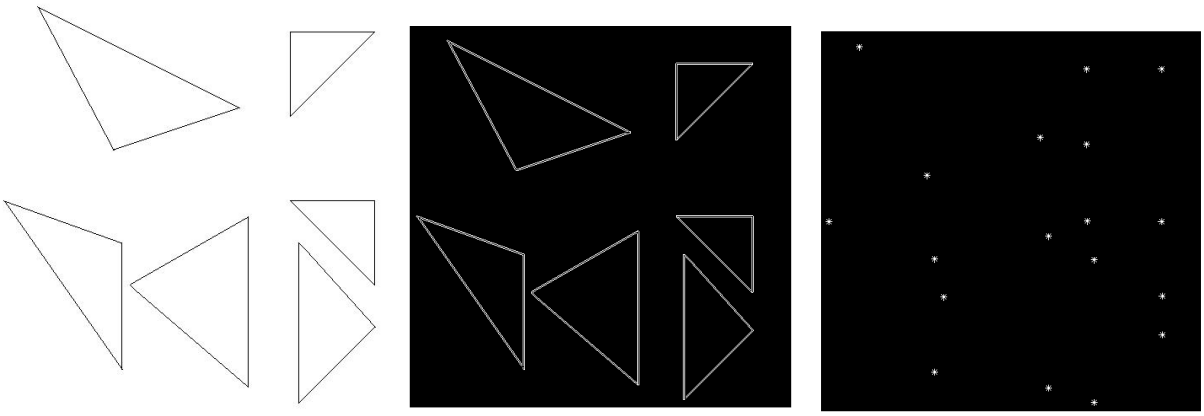


(a) Ptq Image

(b) Ptq Edge Image, Edges = 5347

(c) Ptq Corner Image, Corners = 24

Figure 4.2: Synthetic Image of Multiple Quadrilaterals

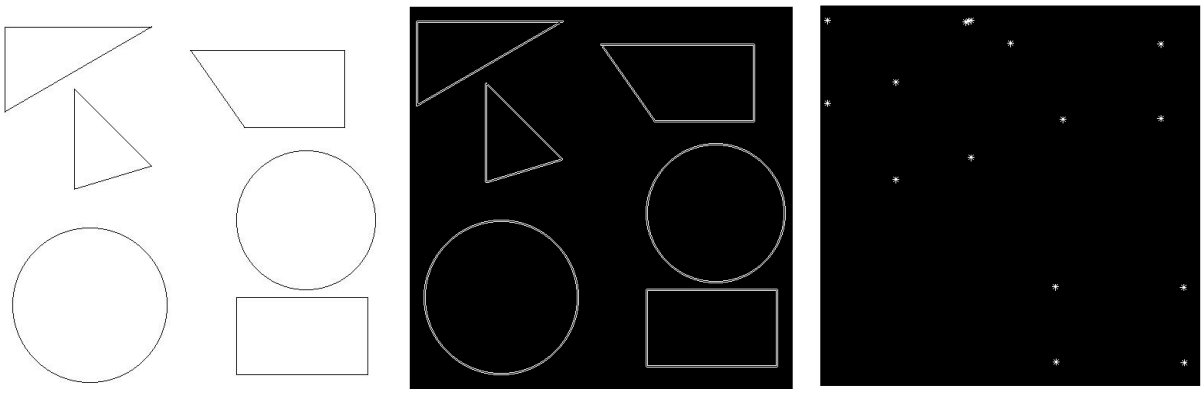


(a) Ptt Image

(b) Ptt Edge Image, Edges = 4975

(c) Ptt Corner Image, Corners = 18

Figure 4.3: Synthetic Image of Multiple Triangles

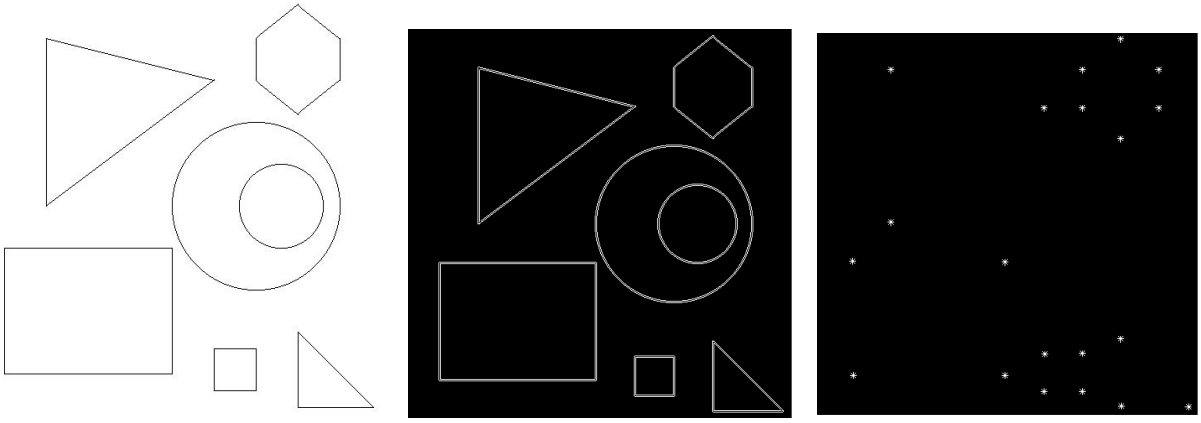


(a) PtMxd1 Image

(b) PtMxd1 Edge Image, Edges = 5890

(c) PtMxd1 Corner Image, Corners = 16

Figure 4.4: Synthetic Image of Multiple Shapes 1



(a) PtMxd2 Image

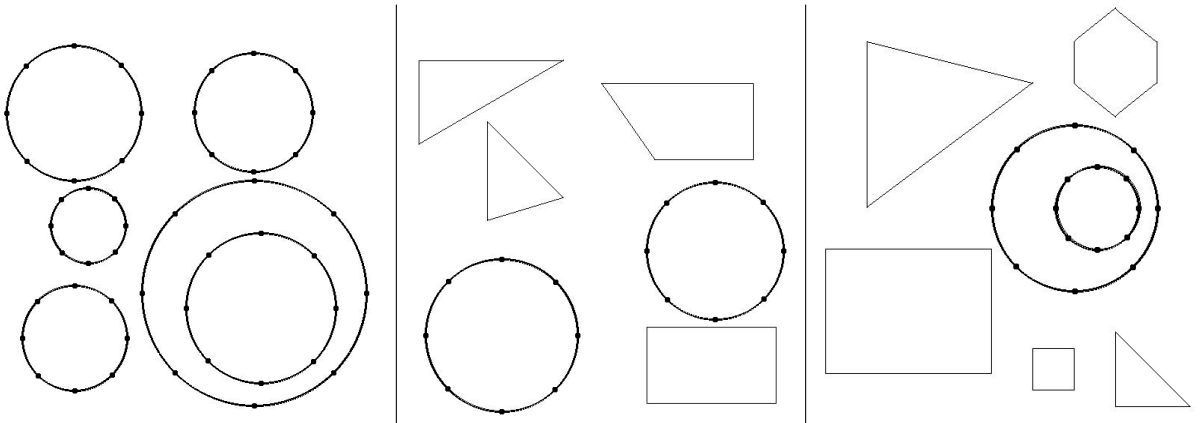
(b) PtMxd2 Edge Image, Edges = 5811

(c) PtMxd2 Corner Image, Corners = 20

Figure 4.5: Synthetic Image of Multiple Shapes 2

4.2.1 Detecting circles

Figure 4.6 shows the detections by the CSA based circle detector on three different synthetic images. The CSA system is able to detect all circles in the image scenes even in the presence of other shapes, discriminating between the different shapes and extracting only the desired circular shapes.



(a) Detecting Circles on Ptc

(b) Detecting Circles on PtMxd1

(c) Detecting Circles on PtMxd2

Figure 4.6: Detecting Circles on Synthetic Images

The results obtained after 20 runs of the algorithm are shown in Table 4.1. The MAE, MSE and PSNR represent the average values, while the FP, FN, TP and TN values were summed up for the 20 runs.

Table 4.1: Detecting Circles on Synthetic Images

Image	MAE	MSE	PSNR	FP	FN	TP	TN
Ptc	0.46	0.5	56.9897	0	17	103	0
PtMxd1	0.45	0.5	56.9897	0	0	40	80
PtMxd2	0.43	0.4259	57.6863	0	2	38	100
Total	-	-	-	0	19	181	180

From Table 4.1, the FPR and FNR can be computed thus:

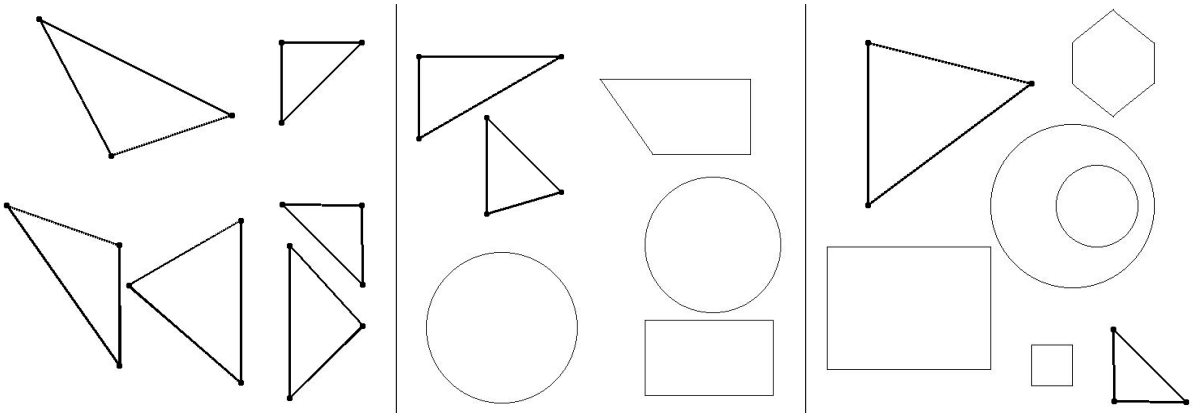
$$\begin{aligned} FPR =& \frac{FP}{FP+TN} = \frac{0}{0+180} = 0 \\ FPR =& 0\% \end{aligned} \tag{4.1}$$

$$\begin{aligned} FNR =& \frac{FN}{TP+FN} = \frac{19}{181+19} = \frac{19}{200} = 0.095 \\ FNR =& 9.5\% \end{aligned} \tag{4.2}$$

FPR of 0% indicates that in all tests, the CSA did not wrongly classify a non-circle as a circle. However, the algorithm failed to detect all circles all the time. This is indicated by the FNR of 9.5%, meaning there is a probability of missing 1 circle out of every 10 circles present in an image. Table 4.1 also contains information about accuracy of detected circles. MAE and MSE values gotten indicate a sub-pixel accuracy detections, with the PSNR computed from the MSE.

4.2.2 Detecting triangles

Figure 4.7 shows the CSA based triangle detector tested on three different synthetic images. The CSA system is able to detect all triangles in the image scenes and just as in the case for detecting only circles, non-triangular shapes were not misclassified as triangles.



(a) Detecting Triangles on Ptt

(b) Detecting Triangles on PtMxd1

(c) Detecting Triangles on PtMxd2

Figure 4.7: Detecting Triangles on Synthetic Images

The results obtained after 20 runs of the algorithm are shown in Table 4.2. The MAE, MSE and PSNR represent the average values, while the FP, FN, TP and TN values were summed up for the 20 runs.

Table 4.2: Detecting Triangles on Synthetic Images

Image	MAE	MSE	PSNR	FP	FN	TP	TN
Ptt	0.44	0.5	56.9897	0	0	120	0
PtMxd1	0.42	0.4167	57.7812	0	0	40	80
PtMxd2	0.42	0.4167	57.7812	0	0	40	100
Total	-	-	-	0	0	200	180

From Table 4.2, the FPR and FNR can be computed thus:

$$FPR = \frac{FP}{FP + TN} = \frac{0}{0 + 180} = 0$$
$$FPR = 0\% \tag{4.3}$$

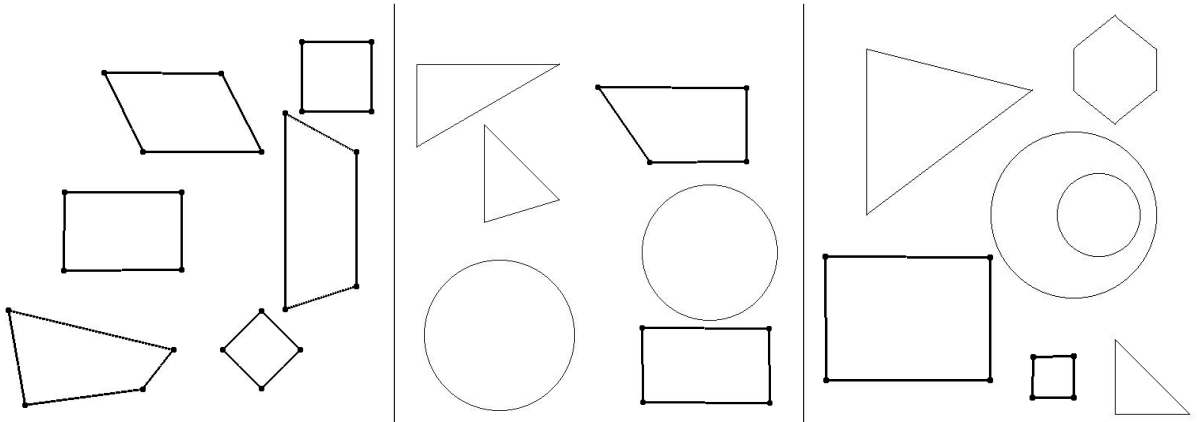
$$FNR = \frac{FN}{TP + FN} = \frac{0}{200 + 0} = 0$$
$$FNR = 0\% \tag{4.4}$$

The triangles detected by the CSA were to a sub-pixel accuracy as indicated by MAE and MSE of less than 1.0 in all cases. There was perfect detections of triangles, as no triangles were missed during the tests. Thus, the FNR is 0% and FPR of 0% indicates that there were no false

alarms (no case of returning a non-triangle as a triangle).

4.2.3 Detecting quadrilaterals

Figure 4.8 shows the CSA based quadrilateral detector tested on three different synthetic images. Showing that irrespective of the positions of the quadrilateral shapes, the CSA correctly identified the quadrilaterals in the test images. The system also distinguished the desired shapes (quadrilaterals in this case) from other shapes present in the test images.



(a) Detecting Quadrilaterals on Ptc

(b) Detecting Quadrilaterals on PtMxd1

(c) Detecting Quadrilaterals on PtMxd2

Figure 4.8: Detecting Quadrilaterals on Synthetic Images

The results obtained after 20 runs of the algorithm are shown in Table 4.3. The MAE, MSE and PSNR represent the average values, while the FP, FN, TP and TN values were summed up for the 20 runs.

Table 4.3: Detecting Quadrilaterals on Synthetic Images

Image	MAE	MSE	PSNR	FP	FN	TP	TN
Ptq	0.52	0.5208	56.8127	0	24	96	0
PtMxd1	0.56	0.5625	56.4782	0	2	38	80
PtMxd2	0.5	0.5	56.9897	0	0	40	100
Total	-	-	-	0	26	174	180

From Table 4.3, the FPR and FNR can be computed thus:

$$FPR = \frac{FP}{FP + TN} = \frac{0}{0 + 180} = 0$$
$$FPR = 0\% \tag{4.5}$$

$$FNR = \frac{FN}{TP + FN} = \frac{26}{174 + 26} = 0.13$$
$$FNR = 13\% \tag{4.6}$$

The CSA showed consistency in reporting 0% FPR as observed when tested on circles and triangles. However, there was a higher FNR of 13%. That is, there is a likelihood of missing 13 quadrilaterals out of every 100 quadrilaterals. In terms of accuracy of detections, the quadrilaterals detected had sub-pixel accuracy with maximum MAE of 0.56 and MSE of 0.5625 as can be readily observed from table 4.3.

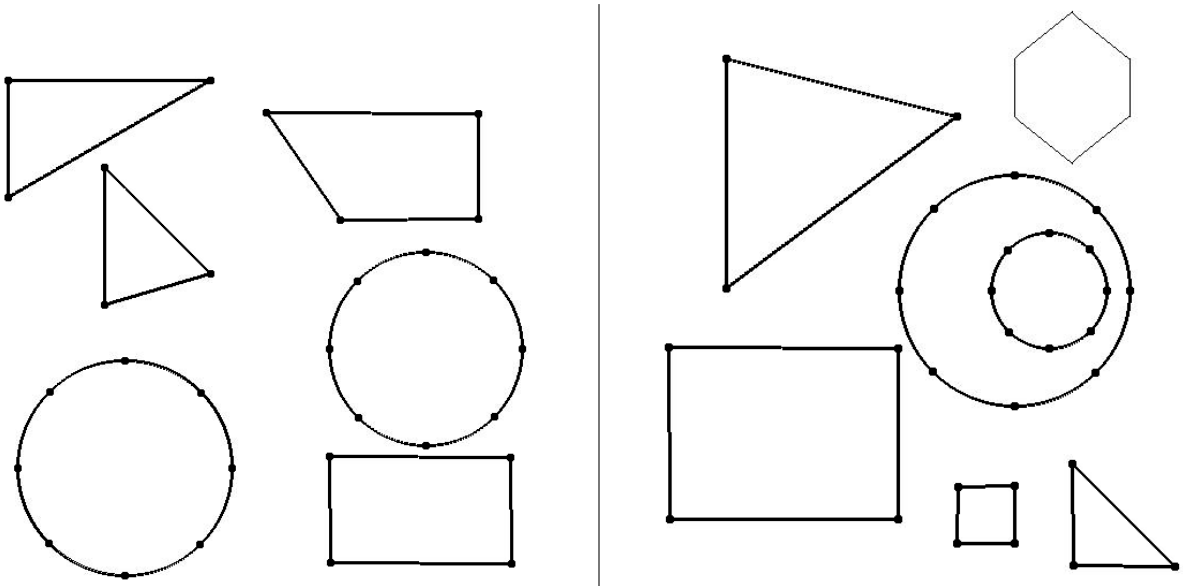
4.2.4 Detecting multiple shapes

The final CSA based system for detecting multiple shapes was then implemented and tested on the synthetic test images in order to test its ability for detecting multiple desired shapes within an image. The CSA system was able to correctly identify all desired shapes in the image scenes - be it multiple instances of a shape or mixture of desired and undesired shapes. This is shown in Figure 4.9, where the task was to detect all quadrilaterals, circles and triangles in the test images.

The system was tested on the five (5) synthetic images and the results are as presented in Table 4.4 for 20 runs of the algorithm on each image. The MAE, MSE and PSNR represent the average values, while the FP, FN, TP and TN values were summed up for the 20 runs.

Table 4.4: Detecting Multiple Shapes on Synthetic Images

Image	MAE	MSE	PSNR	FP	FN	TP	TN
Ptc	0.46	0.4921	57.0589	0	6	114	0
Ptt	0.44	0.5	56.9897	0	0	120	0
Ptq	0.52	0.5208	56.8127	0	11	109	0
PtMxd1	0.48	0.5077	56.9233	0	1	119	0
PtMxd2	0.46	0.4722	57.2381	0	0	120	20
Total	-	-	-	0	18	582	20



(a) Detecting Multiple Shapes on PtMxd1

(b) Detecting Multiple Shapes on PtMxd2

Figure 4.9: Detecting Multiple Shapes on Synthetic images

From Table 4.4, the FPR and FNR can be computed thus:

$$FPR = \frac{FP}{FP + TN} = \frac{0}{0 + 20} = 0$$
$$FPR = 0\% \tag{4.7}$$

$$FNR = \frac{FN}{TP + FN} = \frac{18}{582 + 18} = 0.03$$
$$FNR = 3\% \tag{4.8}$$

The FPR is still 0%, while in the task for detecting mixed shapes, the FNR of 3% has been recorded. All the shapes detected were to a sub-pixel accuracy, with all MAE and MSE in table 4.4 being less than 1.0. Therefore, there was no loss in performance of the algorithm when it was modified to detect multiple shapes.

In order to compare the accuracy of the detections with those reported in Cuevas *et al.* (2012a),called Circle CSA, and Cuevas *et al.* (2012b), called LA, the error scores (minimum, maximum and mean E_s) of the proposed multiple shapes detection CSA, called mCSA, for circles, quadrilaterals, triangles are presented in Table 4.5. The error score on detecting mixture of the three shapes has also been presented.

Table 4.5: Minimum, Maximum and Mean E_s for Circle CSA, LA and mCSA on Synthetic Images

	Minimum E_s	Maximum E_s	Mean E_s
Circle CSA	0.31	0.40	0.36
LA	0.28	0.41	0.34
mCSA (Circle)	0.05	0.30	0.14
mCSA (Triangle)	0.25	0.27	0.26
mCSA (Quadrilateral)	0.40	0.45	0.43
mCSA (AllShapes)	0.05	0.45	0.27

Table 4.5 reports the E_s for the developed CSA capable of detecting multiple shapes (mCSA), and how it compares with the E_s reported in Cuevas *et al.* (2012a) and Cuevas *et al.* (2012b) for circle detection. The mCSA has been able to report the lowest mean $E_{sC} = 0.14$ when detecting circles, and it can thus be concluded that the developed mCSA is the most accurate of the three (3) implementations compared on circle detections. The mean E_s for mCSA when detecting quadrilaterals, $E_{sQ} = 0.43$; triangles, $E_{sT} = 0.26$; and the three (3) shapes combined, $E_s = 0.27$. Figure 4.10 gives a bar chart representation of the error scores for Circle CSA, LA and multiple shapes CSA when detecting circles, showing their respective minimum, maximum and mean E_s.

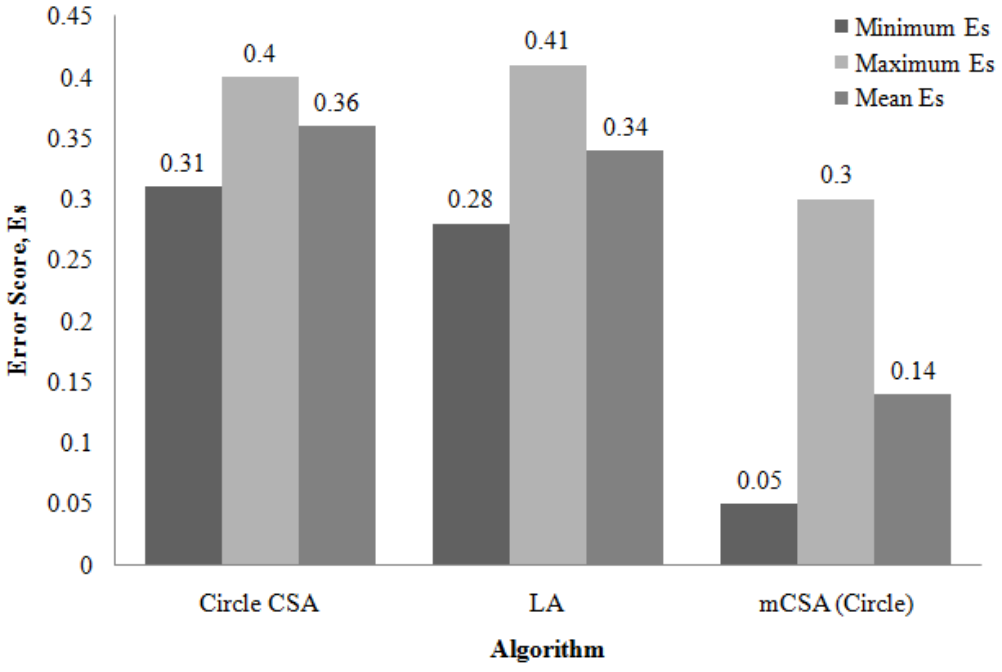


Figure 4.10: Minimum, Maximum and Mean Error Scores for Circle CSA, LA and mCSA (Circle)

The percentage improvement or decrease in error scores for circle detections by the mCSA against those reported in literature are:

1. Improvement on Circle CSA developed by Cuevas *et al.* (2012a)

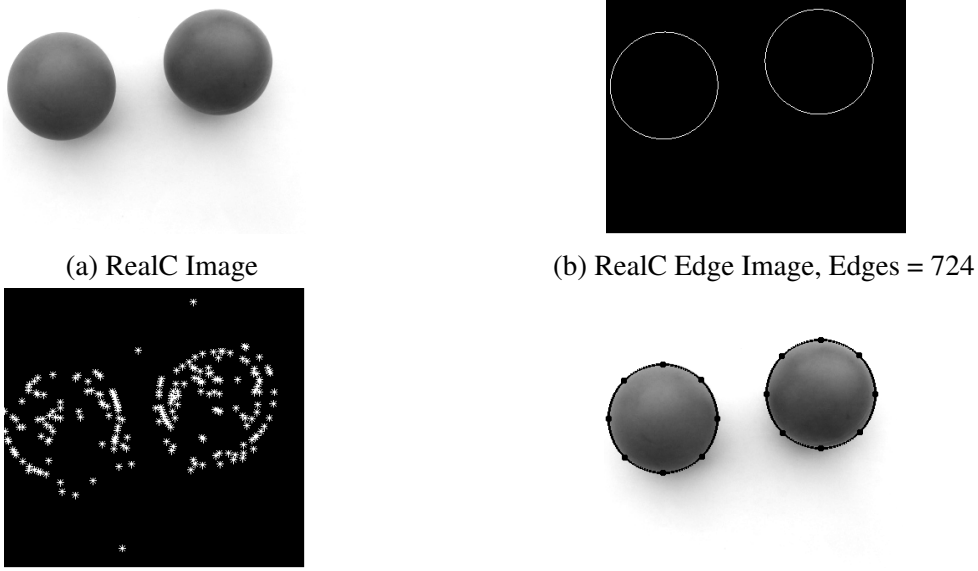
$$\begin{aligned}\text{Percentage improvement} &= \frac{0.36 - 0.14}{0.36} \times 100\% \\ &= 61.11\%\end{aligned}$$

2. Improvement on LA developed by Cuevas *et al.* (2012b)

$$\begin{aligned}\text{Percentage improvement} &= \frac{0.34 - 0.14}{0.34} \times 100\% \\ &= 58.82\%\end{aligned}$$

4.3 Detecting Shapes on Real Images

The CSA based system for detecting multiple shapes was finally tested on six (6) real images containing the desired shapes. Figures 4.11 to 4.16 show the images, edge-only images, corner-only images and detections by the proposed CSA system. The proposed CSA system correctly detected the shapes in all the tested images.



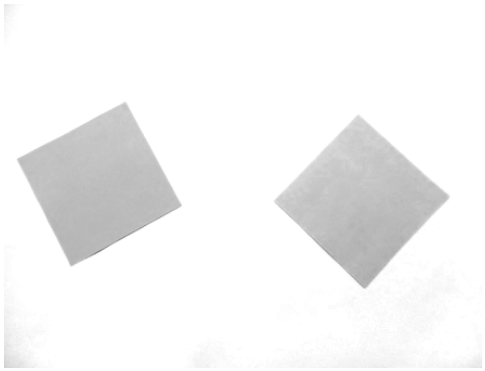
(a) RealC Image

(b) RealC Edge Image, Edges = 724

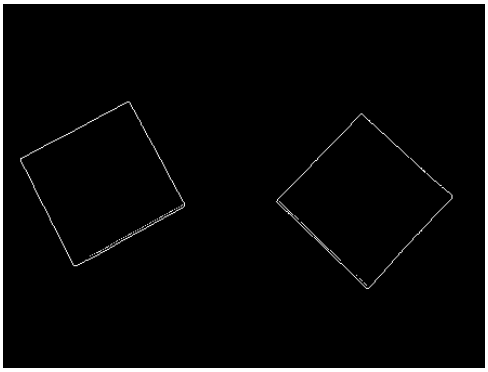
(c) RealC Corner Image, Corners = 200

(d) RealC Detections

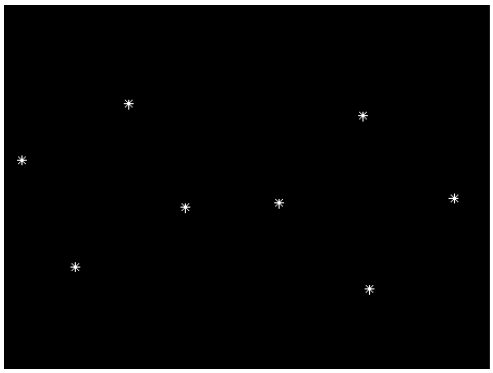
Figure 4.11: Real Image of Circular Shaped Objects, RealC



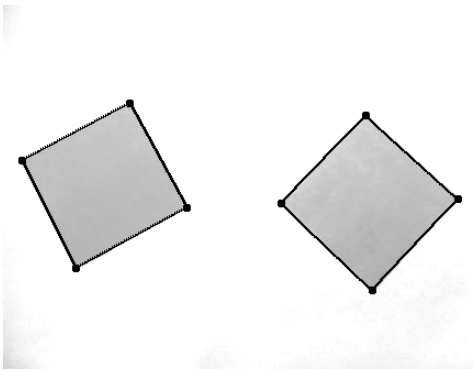
(a) RealQ Image



(b) RealQ Edge Image, Edges = 853

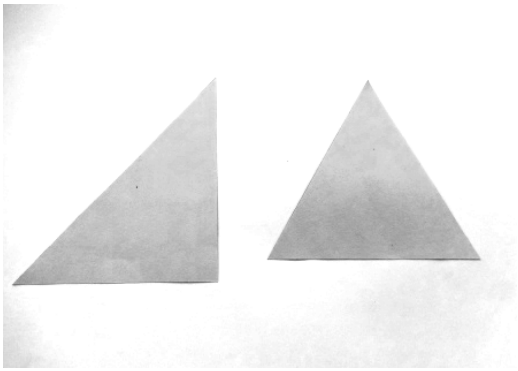


(c) RealQ Corner Image, Corners = 8

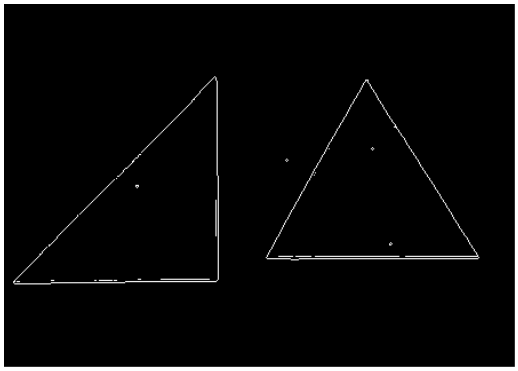


(d) RealQ Detections

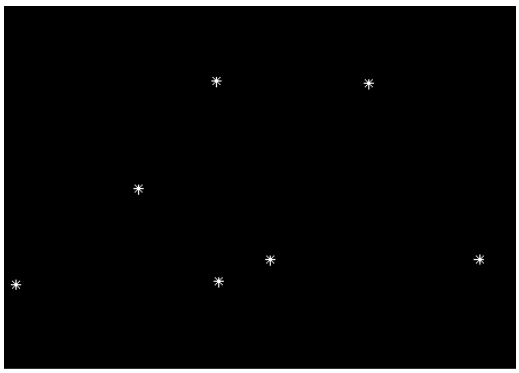
Figure 4.12: Real Images of Quadrilateral Shapes, RealQ



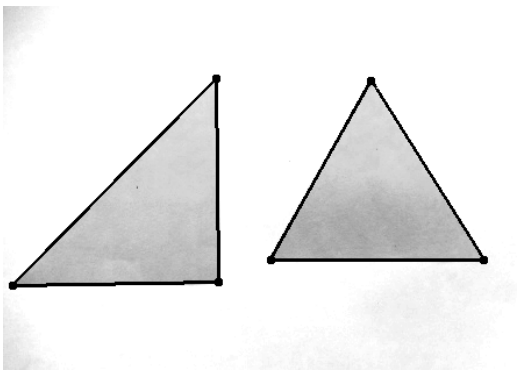
(a) RealT Image



(b) RealT Edge Image, Edges = 1330

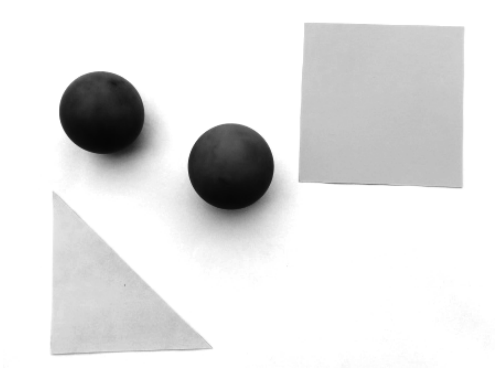


(c) RealT Corner Image, Corners = 7

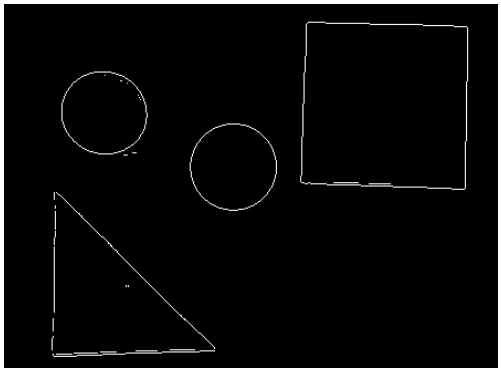


(d) RealT Detections

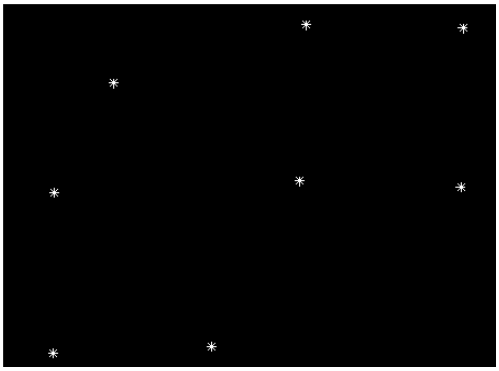
Figure 4.13: Real Images of Triangular Shapes, RealT



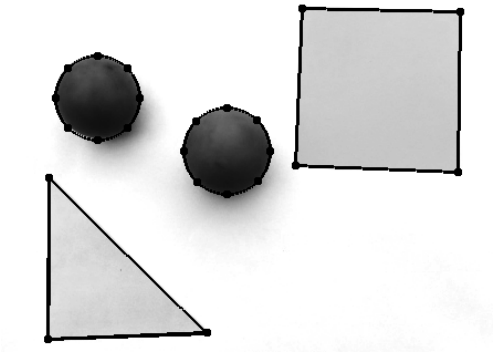
(a) RealMxd1 Image



(b) RealMxd1 Edge Image, Edges = 1601

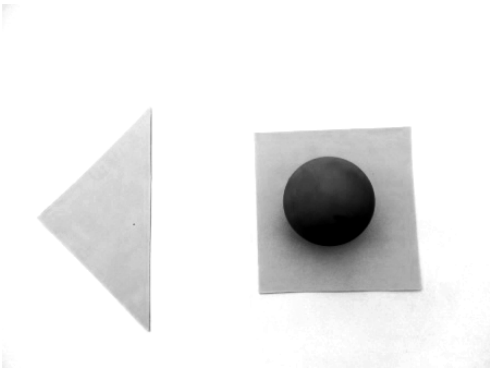


(c) RealMxd1 Corner Image, Corners = 8

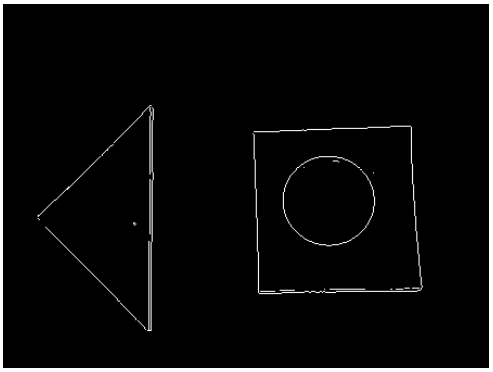


(d) RealMxd1 Detections

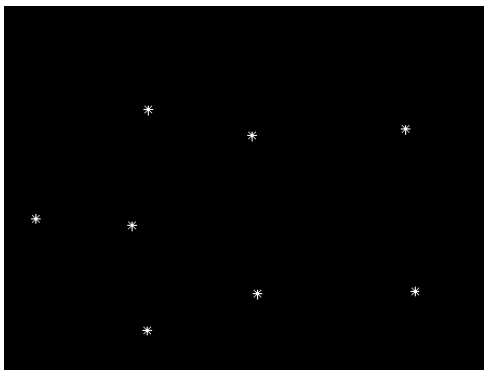
Figure 4.14: Multiple Shapes Image, RealMxd1



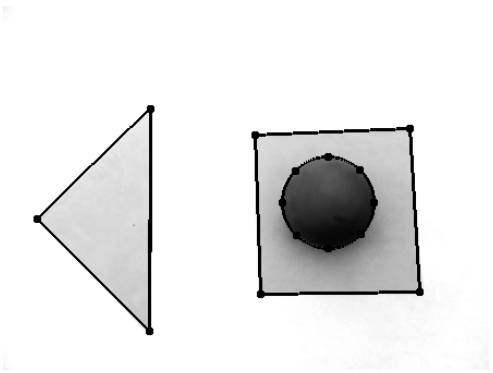
(a) RealMxd2 Image



(b) RealMxd2 Edge Image, Edges = 1532

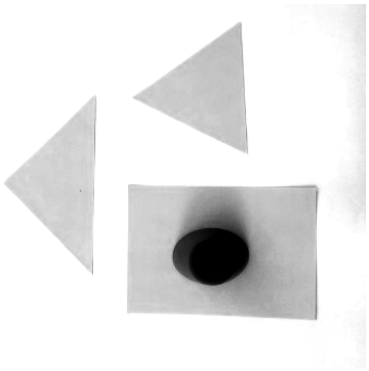


(c) RealMxd2 Corner Image, Corners = 8

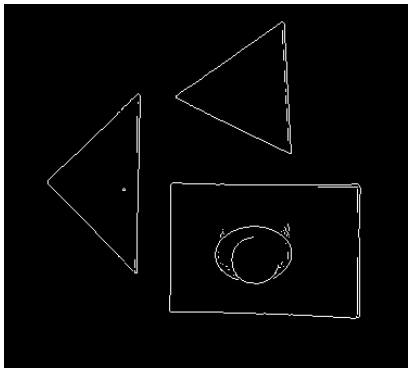


(d) RealMxd2 Detections

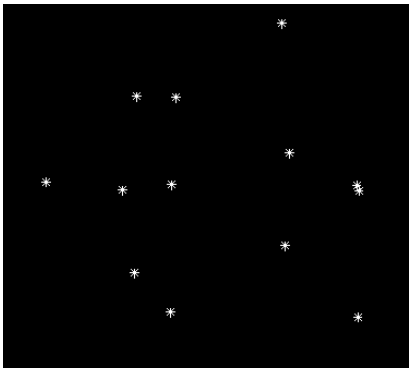
Figure 4.15: Multiple Shapes Image 2, RealMxd2



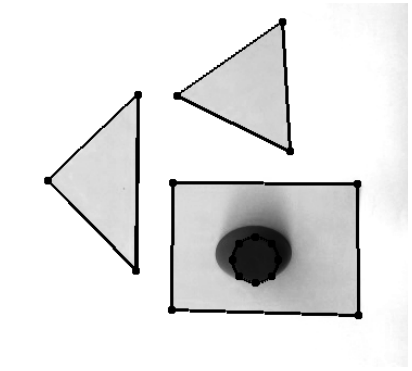
(a) RealMxd3 Image



(b) RealMxd3 Edge Image, Edges = 1769



(c) RealMxd3 Corner Image, Corners = 13



(d) RealMxd3 Detections

Figure 4.16: Multiple Shapes Image 3, RealMxd3

Table 4.6 presents the result for detections on real images when the algorithm tested 20 times on each image.

Table 4.6: Detecting Multiple Shapes on Real Images

Image	FP	FN	TP	TN
RealC	0	7	33	0
RealQ	0	0	40	0
RealT	0	0	40	0
RealMxd1	0	0	80	0
RealMxd2	0	0	60	0
RealMxd3	1	6	74	20
Total	1	13	327	20

Similarly, the FPR and FNR can be computed thus:

$$FPR = \frac{FP}{FP + TN} = \frac{1}{1 + 20} = 0.0476$$
$$FPR = 4.76\% \tag{4.9}$$

$$FNR = \frac{FN}{TP + FN} = \frac{13}{327 + 13} = 0.0382$$
$$FNR = 3.82\% \tag{4.10}$$

With FPR = 4.76% and FNR = 3.82%, as computed from the results shown in Table 4.6, there is therefore a probability of 0.0476 for the mCSA to falsely classify a shape as one of the desired shapes on real images. The probability of mCSA to miss a desired shape in real images is 0.0382. The mCSA algorithm is able to detect these shapes even though they represent imperfect triangles, quadrilaterals and circles as can be seen from the edge images of Figures 4.11 to 4.16. The mCSA is able to achieve this because it does not require a perfect match to ascertain the presence of the shape.

Figure 4.17 gives a bar chart comparison of the FNR and FPR of the proposed CSA system, mCSA, for detecting the desired shapes on synthetic and real images. Lower values of FNR and FPR are reported for the synthetic images because these represent perfect samples of the desired shapes. The imperfections in the captured real images has resulted in increased FPR and FNR. However, despite the odds, the proposed mCSA system was still able to correctly identify circles, quadrilaterals and triangles in the test images.

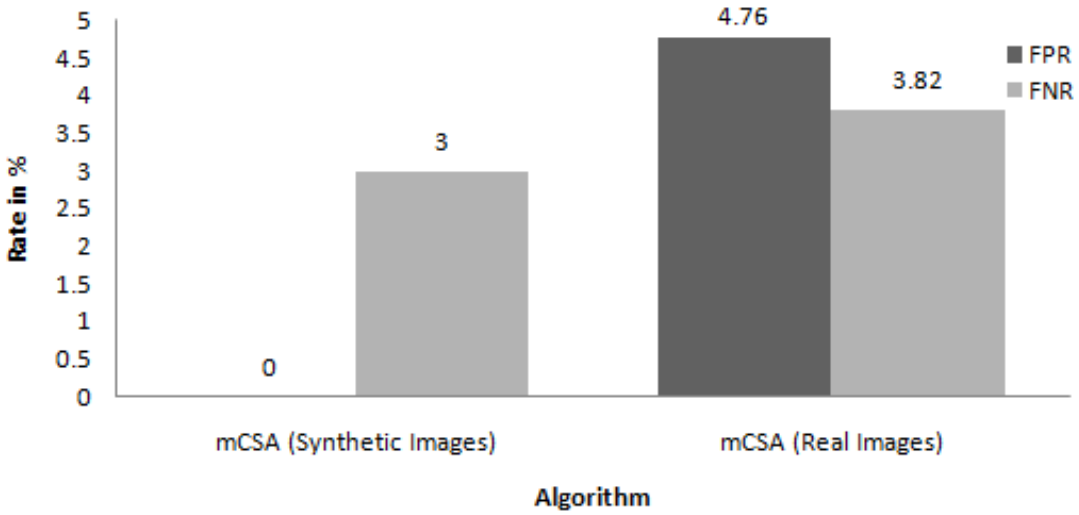


Figure 4.17: False Positive and False Negative Rates for Detecting Multiple Shapes on Real and Synthetic Images

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

This chapter gives the summary of the research, problems encountered, significant contributions conclusions drawn, and recommendations for future directions of the research.

5.2 Summary

This dissertation has reported the development and implementation of CSA based system for detecting multiple circles, quadrilaterals and triangles in an image. The image scene is first preprocessed by extracting all edge and corner points. The next step involved the generation of candidate circles from the edge points, while candidate quadrilaterals and triangles were generated from the corner points. This is followed by optimization of the candidate solutions using the CSA and extraction of candidate solutions with high fitness with the image scene. Finally, these solutions were processed using a distinctness factor to eliminate duplicate detections after which all detected shapes were returned to the user. The CSA system was tested on five (5) synthetic and six (6) real images and the MSE, PSNR, FPR and FNR for these tests were reported. Also reported were the error scores of the implemented multiple shapes detector and how it compared with values reported in two reviewed literature.

5.3 Problems Encountered

The two major problems encountered during the course of the research are:

1. Inability to get a repository of images from authors of published work to compare the developed system with.
2. Inability to access a standard repository of geometric shapes to use to test the algorithm.

5.4 Significant Contributions

From results presented, the major contributions of this research are:

1. The successful development of the CSA algorithm to detect three (3) shapes, unlike other implementations that were capable of detecting only single shape type.
2. The attainment of mean error score of 0.14 for circle detection, representing 61.11% and 58.82% improvement when compared to Circle CSA (with error score of 0.36) and LA (with error score of 0.34) respectively.

5.5 Conclusion

A CSA system has been developed using MATLAB 2015a for detecting multiple circles, quadrilaterals and triangles present in an image scene. The algorithm exploits the basic geometric properties of these shapes to actualize the set objectives of correct detection of all instances of the desired shapes irrespective of their locations, orientations and sizes in both real and synthetic images. The CSA system has been able to achieve these objectives within a single run of the algorithm. Performance analysis carried out on the system reported MSE ranging from 0.4722 to 0.5208 and PSNR ranging from 56.9233 to 57.2381 on synthetic images (Table 4.4) and gave sub pixel accuracy of detected shapes as observed from the MAE of less than 1. FPR of 0% and FNR of 3% was reported on the detections of multiple shapes on synthetic images while on real images, there were higher FPR and FNR of 4.76% and 3.82% respectively. The mean error score of 0.14 on circle detections is the lowest in comparison with the results of Circle CSA (0.36) and LA (0.34) reported in literature, representing an improvement of 61.11% and 58.82% respectively.

5.6 Recommendations for Future Work

Further work can be done to:

1. Investigate ways of reducing the FPR and FNR of the system, especially on real images.

2. Extend the algorithm to detect desired shapes in video scenes.
3. Extend the algorithm to detect more shapes other than the ones studied.
4. Implement the system in a concurrent programming approach rather than the procedural approach adopted in this project.

REFERENCES

Akinlar, C. and Topal, C. Edcircles: a real-time circle detector with a false detection control. *Pattern Recognition*, 46(3):725–740, 2013.

Aman, J., Yao, J., and Summers, R. M. Reducing the false positive rate of computer aided detection for ct colonography using content based image retrieval. In *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 915–918, June 2009. doi: 10.1109/ISBI.2009.5193202.

Canny, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851.

Chaki, J. and Parekh, R. Plant leaf recognition using shape based features and neural network classifiers. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 2(10), 2011.

Challoo, R., Rao, P., Ozcelik, S., Challoo, L., and Li, S. Navigation control and path mapping of a mobile robot using artificial immune systems. *International Journal of Robotics and Automation*, 1(1):1, 2010.

Cheng-Bin, L. A new intrusion prediction method based on feature extraction. In *Computer Science and Engineering, 2009. WCSE '09. Second International Workshop on*, volume 1, pages 7–10, Oct 2009. doi: 10.1109/WCSE.2009.610.

Cuevas, E., Osuna-Enciso, V., Wario, F., Zaldívar, D., and Pérez-Cisneros, M. Automatic multiple circle detection based on artificial immune systems. *Expert Systems with Applications*, 39(1):713–722, 2012a.

Cuevas, E., Wario, F., Osuna-Enciso, V., Zaldivar, D., and Perez-Cisneros, M. Fast algorithm for multiple-circle detection on images using learning automata. *IET Image Processing*, 6 (8):1124–1135, 2012b.

Dasgupta, D. An overview of artificial immune systems and their applications. In *Artificial immune systems and their applications*, pages 3–21. Springer, 1999.

Dasgupta, D., Yu, S., and Nino, F. Recent advances in artificial immune systems: models and applications. *Applied Soft Computing*, 11(2):1574–1587, 2011.

Dasgupta, S., Das, S., Biswas, A., and Abraham, A. Automatic circle detection on digital images with an adaptive bacterial foraging algorithm. *Soft Computing*, 14(11):1151–1164, 2010.

De Castro, L. N. and Von Zuben, F. J. The clonal selection algorithm with engineering applications. In *Proceedings of GECCO*, volume 2000, pages 36–39, 2000.

De Castro, L. N. and Von Zuben, F. J. Artificial immune systems: Part i–basic theory and applications. *Universidade Estadual de Campinas, Dezembro de, Tech. Rep*, 210, 1999.

De Marco, T., Cazzato, D., Leo, M., and Distante, C. Randomized circle detection with isophotes curvature analysis. *Pattern Recognition*, 48(2):411–421, 2015.

Freschi, F., Coello, C. A. C., and Repetto, M. Multiobjective optimization and artificial immune systems: a review. *Handbook of Research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies*, 4:1–21, 2009.

Fressin, F., Torres, G., Charbonneau, D., Bryson, S. T., Christiansen, J., Dressing, C. D., Jenkins, J. M., Walkowicz, L. M., and Batalha, N. M. The false positive rate of kepler and the occurrence of planets. *The Astrophysical Journal*, 766(2):81, 2013. URL `http://stacks.iop.org/0004-637X/766/i=2/a=81`.

Gantert, A. *Amsco's Geometry*. AMSCO School Publications, Incorporated, 2008. ISBN 9781567655964. URL `https://books.google.com.ng/books?id=9jm-PAAACAAJ`.

Garrett, S. M. How do we evaluate artificial immune systems? *Evolutionary computation*, 13 (2):145–177, 2005.

Gavrila, D. M. Multi-feature hierarchical template matching using distance transforms. In

Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on, volume 1, pages 439–444. IEEE, 1998.

Greensmith, J., Whitbrook, A., and Aickelin, U. Artificial immune systems. In *Handbook of Metaheuristics*, pages 421–448. Springer, 2010.

Harris, C. and Stephens, M. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988.

Hearn, D. and Baker, M. P. *Computer Graphics (2Nd Ed.): C Version*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997. ISBN 0-13-530924-7.

Hoffrage, U., Lindsey, S., Hertwig, R., and Gigerenzer, G. Communicating statistical information. *Science*, 290:2261–2262, 2000.

Ji, Z. and Dasgupta, D. Revisiting negative selection algorithms. *Evol. Comput.*, 15 (2):223–251, June 2007. ISSN 1063-6560. doi: 10.1162/evco.2007.15.2.223. URL http://dx.doi.org/10.1162/evco.2007.15.2.223.

Jiang, L. Efficient randomized hough transform for circle detection using novel probability sampling and feature points. *Optik-International Journal for Light and Electron Optics*, 123 (20):1834–1840, 2012.

Kay, D. *College Geometry: A Unified Development*. Textbooks in Mathematics. Taylor & Francis, 2011. ISBN 9781439819111. URL https://books.google.com.ng/books?id=lzlJv921uOkC.

Korhonen, J. and You, J. Peak signal-to-noise ratio revisited: Is simple beautiful? In *Quality of Multimedia Experience (QoMEX), 2012 Fourth International Workshop on*, pages 37–38. IEEE, 2012.

Leff, L. *Geometry the Easy Way*. Barron's E-Z. Barron's Educational Series, 1997. ISBN 9780764101106. URL https://books.google.com.ng/books?id=brXqptpHUMQC.

Li, Q. A geometric framework for rectangular shape detection. *Image Processing, IEEE Transactions on*, 23(9):4139–4149, 2014.

Lu, D. and Yu, X.-H. Multi-circle detection for bladder cancer diagnosis based on artificial immune systems. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–7. IEEE, 2013.

Luh, G.-C. and Liu, W.-W. An immunological approach to mobile robot reactive navigation. *Applied Soft Computing*, 8(1):30–45, 2008.

Matern, D., Tortorelli, S., Oglesbee, D., Gavrilov, D., and Rinaldo, P. Reduction of the false-positive rate in newborn screening by implementation of ms/ms-based second-tier tests: The mayo clinic experience (2004–2007). *Journal of Inherited Metabolic Disease*, 30(4):585–592, 2007. ISSN 1573-2665. doi: 10.1007/s10545-007-0691-y. URL `http://dx.doi.org/10.1007/s10545-007-0691-y`.

McDonald, J. H. *Handbook of biological statistics*, volume 2. Sparky House Publishing Baltimore, MD, 2009.

Neycenssac, F. Contrast enhancement using the laplacian-of-a-gaussian filter. *CVGIP: Graphical Models and Image Processing*, 55(6):447–463, 1993.

Pender, W., Saddler, D., Shea, J., and Ward, D. *Cambridge 2 Unit Mathematics Year 11 Enhanced Version PDF Textbook*. Cambridge Secondary Maths (Australia) Series. Cambridge University Press, 2011. ISBN 9781107679573. URL `https://books.google.com.ng/books?id=wDKLXdzQL5AC`.

Ramirez, V. A., Gutierrez, S. A. M., and Yanez, R. E. S. Quadrilateral detection using genetic algorithms. *Computación y Sistemas*, 15(2):181–193, 2011.

Scitovski, R. and Marošević, T. Multiple circle detection based on center-based clustering. *Pattern Recognition Letters*, 52:9–16, 2015.

Secker, A., Freitas, A. A., and Timmis, J. Aisec: an artificial immune system for e-mail clas-

sification. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 1, pages 131–138. IEEE, 2003.

Ulutas, B. H. and Kulturel-Konak, S. A review of clonal selection algorithm and its applications. *Artificial Intelligence Review*, 36(2):117–138, 2011.

Venema, G. *Exploring Advanced Euclidean Geometry with GeoGebra*. Classroom Resource Materials. Mathematical Association of America, 2013. ISBN 9780883857847. URL https://books.google.com.ng/books?id=VrAar6R486IC.

Vennila, G., Shalini, N. S., and Manikandan, M. Performance analysis of voip spoofing attacks using classification algorithms. In *Applications and Innovations in Mobile Computing (AIMoC), 2014*, pages 193–198, Feb 2014. doi: 10.1109/AIMOC.2014.6785540.

Wang, Z. and Bovik, A. C. Mean squared error: love it or leave it? a new look at signal fidelity measures. *Signal Processing Magazine, IEEE*, 26(1):98–117, 2009.

Weisstein, E. W. Line-Line Intersection, 2002a. URL http://mathworld.wolfram.com/Line-LineIntersection.html.

Weisstein, E. W. Point-Line Distance 3-Dimensional, 2002b. URL http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html.

Willmott, C. J. and Matsuura, K. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1): 79, 2005.

Witten, I., Frank, E., and Hall, M. *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011. ISBN 9780080890364. URL https://books.google.com.ng/books?id=bDtLM8CODsQC.

Wolfer, L. R., Derby, R., Lee, J.-E., and Lee, S.-H. Systematic review of lumbar provocation discography in asymptomatic subjects with a meta-analysis of false-

positive rates. *Pain physician*, 11(4):513—538, 2008. ISSN 1533-3159. URL http://europepmc.org/abstract/MED/18690280.

APPENDICES

APPENDIX A

Shape Points

```
function [Ptc, Ptq, Ptt, PtMxd1, PtMxd2, PtMxd3] = ShapePoints()
Ptc = {'Images/Ptc.jpg' {'c' [250, 120, 50;  ...
            100, 340, 80;  ...
            100, 100, 90;  ...
            400, 100, 70;  ...
            340, 340, 150;  ...
            360, 350, 100]
        'q' []
        't' []
        'p' []}};
Ptq = {'Images/Ptq.jpg' {'c' []
       'q' [10, 490, 10, 400;  10, 400, 100, 400;  100, 400, 100, 490;  100, 490, 10,
           490;  ...
            100, 380, 350, 380;  350, 380, 320, 470;  320, 470, 150, 470;  150, 470, 100,
                380;  ...
            350, 350, 400, 400;  400, 400, 450, 350;  450, 350, 400, 300;  400, 300, 350,
                350;  ...
            50, 300, 150, 350;  150, 350, 150, 200;  150, 200, 50, 150;  50, 150, 50,
               300;  ...
            200, 250, 300, 250;  300, 250, 300, 100;  300, 100, 200, 100;  200, 100, 200,
                250;  ...
            400, 240, 350, 30; 350, 30, 470, 50; 470, 50, 450, 200; 450, 200, 400, 240]
       't' []
       'p' []}};
Ptt = {'Images/Ptt.jpg' {'c' []
       'q' []
       't' [50, 450, 50, 350; 50, 350, 150, 350; 150, 350, 50, 450; ...
            250, 450, 350, 450; 350, 450, 250, 350; 250, 350, 250, 450; ...
            300, 360, 400, 450; 400, 450, 490, 360; 490, 360, 300, 360; ...
            20, 50, 140, 290; 140, 290, 190, 140; 190, 140, 20, 50; ...
            270, 300, 350, 160; 350, 160, 470, 300; 470, 300, 270, 300; ...
            300, 150, 250, 10; 250, 10, 450, 150; 450, 150, 300, 150]
       'p' []}};
PtMxd1 = {'Images/PtMxd1.jpg' {'c' [300, 170, 150]
          'q' [50, 450, 200, 450; 200, 450, 300, 350; 300, 350, 50, 350; 50, 350, 50,
              450]
          't' [300, 250, 200, 100; 200, 100, 400, 100; 400, 100, 300, 250]
          'p' []}};
PtMxd2 = {'Images/PtMxd2.jpg' {'c' [270, 400, 90; ...
               380, 120, 100]
          'q' [50, 450, 150, 450; 150, 450, 150, 320; 150, 320, 50, 250; 50, 250, 50,
              450; ...
               370, 480, 470, 480; 470, 480, 470, 310; 470, 310, 370, 310; 370, 310, 370,
                   480]
          't' [20, 200, 130, 10; 130, 10, 20, 10; 20, 10, 20, 200; ...
               100, 100, 200, 200; 200, 200, 230, 100; 230, 100, 100, 100]
          'p' []}};
PtMxd3 = {'Images/PtMxd3.jpg' {'c' [250, 380, 50; ...
               250, 350, 100]
          'q' [420, 350, 470, 350; 470, 350, 470, 300; 470, 300, 420, 300; 420, 300, 420,
               350; ...
```

```
43              300, 250, 450, 250; 450, 250, 450, 50; 450, 50, 300, 50; 300, 50, 300,
                    250]
44          't' [490, 490, 490, 400; 490, 400, 400, 400; 400, 400, 490, 490; ...
45               100, 300, 250, 100; 250, 100, 50, 100; 50, 100, 100, 300]
46          'p' [10, 400, 50, 450; 50, 450, 100, 450; 100, 450, 140, 400; 140, 400, 100,
                350; 100, 350, 50, 350; 50, 350, 10, 400]}};
47 end
```

APPENDIX B

Shape Maker

```
function ShapeMaker(PtAllt,Size)
Image = NaN(Size);
Image(:,:,:) = 255;
PtName = PtAllt{1,1};
PtAll = PtAllt{1,2};
for Shp = 1:size(PtAll,1)
    Shape = PtAll{Shp,1};
    Pt = PtAll{Shp,2};
    if ~isempty(Pt)
        ShapesCount = size(Pt,1);
        for i1 = 1:ShapesCount
            if strcmp(Shape,'c')
                xyMatc = MCA(Pt(i1,1),Pt(i1,2),Pt(i1,3));
                for i2 = 1:size(xyMatc,1)
                    Image(xyMatc(i2,1),xyMatc(i2,2),:) = 0;
                end
            elseif strcmp(Shape,'q')
                xyMatq = MLA([Pt(i1,1),Pt(i1,2)],[Pt(i1,3),Pt(i1,4)]);
                for i2 = 1:size(xyMatq,1)
                    Image(xyMatq(i2,2),xyMatq(i2,1),:) = 0;
                end
            elseif strcmp(Shape,'t')
                xyMatt = MLA([Pt(i1,1),Pt(i1,2)],[Pt(i1,3),Pt(i1,4)]);
                for i2 = 1:size(xyMatt,1)
                    Image(xyMatt(i2,2),xyMatt(i2,1),:) = 0;
                end
            elseif strcmp(Shape,'p')
                xyMatp = MLA([Pt(i1,1),Pt(i1,2)],[Pt(i1,3),Pt(i1,4)]);
                for i2 = 1:size(xyMatp,1)
                    Image(xyMatp(i2,2),xyMatp(i2,1),:) = 0;
                end
            else
                display('ShapeMaker Invalid Shape');
            end
        end
    end
end
imwrite(Image,PtName);
imshow(Image)
end
```

APPENDIX C

CSA for Detecting Multiple Circles

```
function [Pt,CornerImage,CornerMap,EdgeImage,EdgeMap,Image] = CSAcircle(ImageNo)
warning('off', 'MATLAB:singularMatrix');
warning('off','MATLAB:nearlysSingularMatrix');
[CornerImage,CornerMap,EdgeImage,EdgeMap,Image] = PrepImage(ImageNo);
NAb = 120; %Population of Antibodies
NMAb = 100; %Population of Memory Cells
NewAb = NAb - NMAb; %New random antibodies to be added to pool
NC = 100; %Total Number of clones
P = 3; %Step size/damping factor for mutation prob
Its = 100; %Number of iterations...
[~,iMaxEdges] = size(EdgeMap); %iMaxEdges stores the total edges in the image.
[~,iMaxCorners] = size(CornerMap); %iMaxCorners stores the total corners in the image
Pt = CreateAntibodies(NAb,iMaxEdges,EdgeMap,EdgeImage);
Pt = sortrows(Pt,-5);
GBest = Pt(1,:);
for its = 1:Its
    fprintf('%d %s\n %s\n',its,mat2str(GBest),mat2str([Pt(10,5),Pt(30,5),Pt(60,5),Pt
        (88,5)]));
    if GBest(1,5) == 1
        fprintf('Perfect\n');
        break;
    end
    Ak = Pt(1:NMAb,:); %Select Best NMAb memory antibodies
    Ak = C_HM_CS(Ak,NC,iMaxEdges,EdgeMap,EdgeImage,P); %perform C,HM and CS
    Ab_New = CreateAntibodies(NewAb,iMaxEdges,EdgeMap,EdgeImage);%Create new antibodies
    Pt = [Ak;Ab_New];%Replace worst antibodies with the newly created ones
    Pt = sortrows(Pt,-5);
    GBest = Pt(1,:);
end
    function Ak_New = C_HM_CS(Ak,NC,iMax,Map,EdgeImage,P)
        Ak_New = NaN(size(Ak));
        Den = sum(Ak(:,5));
        if Den == 0
            Den = 1e-12;
        end
        for i1 = 1:size(Ak_New,1)
            Qk = round(NC*(Ak(i1,5)/Den));
            Clones = ones(Qk+1,1); %An additional clone to retain the original antibody
            Clones = Clones*Ak(i1,:);
            MutProb = exp(-P*Ak(i1,5));
            Ak_New(i1,:) = Mutate(Clones,iMax,MutProb,Map,EdgeImage);
        end
    end
    function NewAk = Mutate(Clones,iMax,MutProb,Map,EdgeImage)
        Rws = size(Clones,1)-1; %In order to prevent mutationo of the original antibody
        i2 = 1;
        while i2 <= Rws
            tIndex = Clones(i2,1:3);
            for i3 = 1:3
                if rand <= MutProb
                    tIndex(1,i3) = randi(iMax);
                end
```

```
52              end
53              xy1 = Map{tIndex(1,1)}; %Extract the xy coordinates
54              xy2 = Map{tIndex(1,2)};
55              xy3 = Map{tIndex(1,3)};
56              if ~Collinear(xy1,xy2,xy3) && (numel(unique(tIndex))==size(tIndex,2))
57                  Clones(i2,1:3) = sort(tIndex(1,1:3));
58                  [x0, y0, R] = GetCenter(xy1, xy2, xy3);
59                  xy = MCA(x0, y0, R);
60                  Fitness = GetFitness(xy,EdgeImage);
61                  Clones(i2,5:8) = [Fitness,R,x0,y0];
62                  i2 = i2 + 1;
63              end
64          end
65          Clones = sortrows(Clones,-5);
66          NewAk = Clones(1,:);
67      end
68  end
```

APPENDIX D

CSA for Detecting Multiple Quadrilaterals

```
function [Pt,CornerImage,CornerMap,EdgeImage,EdgeMap,Image] = CSAquadrilateral(ImageNo)
warning('off', 'MATLAB:singularMatrix');
warning('off','MATLAB:nearlysSingularMatrix');
warning('off');
if ImageNo ==0
   [Image, EdgeImage,EdgeMap,CornerImage,CornerMap] = ShapeMaker();
else
    [CornerImage,CornerMap,EdgeImage,EdgeMap,Image] = PrepImage(ImageNo);
end
NAb = 120; %Population of Antibodies
NMAb = 100; %Population of Memory Cells
NewAb = NAb - NMAb; %New random antibodies to be added to pool
NC = 100; %Total Number of clones
P = 3; %Step size/damping factor for mutation prob
Its = 100; %Number of iterations...
[~,iMaxEdges] = size(EdgeMap); %iMaxEdges stores the total edges in the image.
[~,iMaxCorners] = size(CornerMap); %iMaxCorners stores the total corners in the image
Pt = CreateAntibodies(NAb,iMaxCorners,CornerMap,EdgeImage);
Pt = sortrows(Pt,-5);
GBest = Pt(1,:);
for its = 1:Its
    fprintf('%d %s\n %s\n',its,mat2str(GBest),mat2str([Pt(10,5),Pt(30,5),Pt(60,5),Pt
        (88,5)]));
    if GBest(1,5) == 1
        fprintf('Perfect\n');
        break;
    end
    Ak = Pt(1:NMAb,:); %Select Best NMAb memory antibodies
    Ak = C_HM_CS(Ak,NC,iMaxCorners,CornerMap,EdgeImage,P); %perform C,HM and CS
    Ab_New = CreateAntibodies(NewAb,iMaxCorners,CornerMap,EdgeImage);%Create new
        antibodies
    Pt = [Ak;Ab_New];%Replace worst antibodies with the newly created ones
    Pt = sortrows(Pt,-5);
    GBest = Pt(1,:);
end
    function Ak_New = C_HM_CS(Ak,NC,iMax,Map,EdgeImage,P)
        Ak_New = NaN(size(Ak));
        Den = sum(Ak(:,5));
        if Den == 0
            Den = 1e-12;
        end
        for i1 = 1:size(Ak_New,1)
            Qk = round(NC*(Ak(i1,5)/Den));
            Clones = ones(Qk+1,1);
            Clones = Clones*Ak(i1,:);
            MutProb = exp(-P*Ak(i1,5));
            Ak_New(i1,:) = Mutate(Clones,iMax,MutProb,Map,EdgeImage);
        end
    end
    function NewAk = Mutate(Clones,iMax,MutProb,Map,EdgeImage)
        Rws = size(Clones,1)-1; %In order to prevent mutation of the original antibody,
            the last antibody is not mutated in the pools of antibody clones to be
```

```
            mutated acording to the mutation probability
        i2 = 1;
        while i2 <= Rws
            tIndex = Clones(i2,1:4);
            for i3 = 1:size(tIndex,2)
                if rand <= MutProb %&&
                    tIndex(1,i3) = randi(iMax);
                end
            end
            xy1 = Map{tIndex(1,1)}; %Extract the xy coordinates
            xy2 = Map{tIndex(1,2)};
            xy3 = Map{tIndex(1,3)};
            xy4 = Map{tIndex(1,4)};
            if ~Collinear(xy1,xy2,xy3,xy4) && (numel(unique(tIndex))==size(tIndex,2))
                Clones(i2,1:4) = sort(tIndex(1,1:4));
                Fitness = GetQuadFitness(xy1,xy2,xy3,xy4,EdgeImage,2);
                Clones(i2,5:13) = [Fitness,xy1,xy2,xy3,xy4];
                i2 = i2 + 1;
            end
        end
        Clones = sortrows(Clones,-5);
        NewAk = Clones(1,:);
    end
end
```

APPENDIX E

CSA for Detecting Multiple Triangles

```
function [Pt,CornerImage,CornerMap,EdgeImage,EdgeMap,Image] = CSAtriangle(ImageNo)
warning('off', 'MATLAB:singularMatrix');
warning('off','MATLAB:nearlysSingularMatrix');
[CornerImage,CornerMap,EdgeImage,EdgeMap,Image] = PrepImage(ImageNo);
NAb = 120; %Population of Antibodies
NMAb = 100; %Population of Memory Cells
NewAb = NAb - NMAb; %New random antibodies to be added to pool
NC = 100; %Total Number of clones
P = 3; %Step size/damping factor for mutation prob
Its = 100; %Number of iterations...
[~,iMaxEdges] = size(EdgeMap); %iMaxEdges stores the total edges in the image.
[~,iMaxCorners] = size(CornerMap); %iMaxCorners stores the total corners in the image
Pt = CreateAntibodies(NAb,iMaxCorners,CornerMap,EdgeImage);
Pt = sortrows(Pt,-5);
GBest = Pt(1,:);
for its = 1:Its
    fprintf('%d %s\n %s\n',its,mat2str(GBest),mat2str([Pt(10,5),Pt(30,5),Pt(60,5),Pt
        (88,5)]));
    if GBest(1,5) == 1
        fprintf('Perfect\n');
        break;
    end
    Ak = Pt(1:NMAb,:); %Select Best NMAb memory antibodies
    Ak = C_HM_CS(Ak,NC,iMaxCorners,CornerMap,EdgeImage,P); %perform C,HM and CS
    Ab_New = CreateAntibodies(NewAb,iMaxCorners,CornerMap,EdgeImage);%Create new
        antibodies
    Pt = [Ak;Ab_New];%Replace worst antibodies with the newly created ones
    Pt = sortrows(Pt,-5);
    GBest = Pt(1,:);
end
    function Ak_New = C_HM_CS(Ak,NC,iMax,Map,EdgeImage,P)
        Ak_New = NaN(size(Ak));
        Den = sum(Ak(:,5));
        if Den == 0
            Den = 1e-12;
        end
        for i1 = 1:size(Ak_New,1)
            Qk = round(NC*(Ak(i1,5)/Den));
            Clones = ones(Qk+1,1);
            Clones = Clones*Ak(i1,:);
            MutProb = exp(-P*Ak(i1,5));
            Ak_New(i1,:) = Mutate(Clones,iMax,MutProb,Map,EdgeImage);
        end
    end
    function NewAk = Mutate(Clones,iMax,MutProb,Map,EdgeImage)
        Rws = size(Clones,1)-1; %In order to prevent mutation of the original antibody,
            the last antibody is not mutated in the pools of antibody clones to be
            mutated acording to the mutation probability
        i2 = 1;
        while i2 <= Rws
            tIndex = Clones(i2,1:3);
            for i3 = 1:size(tIndex,2)
```

```
49                  if rand <= MutProb
50                      tIndex(1,i3) = randi(iMax);
51                  end
52              end
53              xy1 = Map{tIndex(1,1)}; %Extract the xy coordinates
54              xy2 = Map{tIndex(1,2)};
55              xy3 = Map{tIndex(1,3)};
56              if ~Collinear(xy1,xy2,xy3) && (numel(unique(tIndex))==size(tIndex,2))
57                  Clones(i2,1:3) = sort(tIndex(1,1:3));
58                  Fitness = GetTriangleFitness(xy1,xy2,xy3,EdgeImage,2);
59                  Clones(i2,5:11) = [Fitness,xy1,xy2,xy3];
60                  i2 = i2 + 1;
61              end
62          end
63          Clones = sortrows(Clones,-5);
64          NewAk = Clones(1,:);
65      end
66  end
```

APPENDIX F

CSA for Detecting Multiple Shapes

```
function [PtAll,CornerImage,CornerMap,EdgeImage,EdgeMap,Image] = CSA(ImgName)
warning('off');
PtAll = {'c',[];'q',[];'t',[]};
% if ImageNo ==0
%    [Image, EdgeImage,EdgeMap,CornerImage,CornerMap] = ShapeMaker();
% else
    [CornerImage,CornerMap,EdgeImage,EdgeMap,Image] = PrepImage(ImgName);
% end
tic
NAb = 120; %Population of Antibodies
NMAb = 100; %Population of Memory Cells
NewAb = NAb - NMAb; %New random antibodies to be added to pool
NC = 100; %Total Number of clones
P = 3; %Step size/damping factor for mutation prob
Its = 100; %Number of iterations...

[~,iMaxEdges] = size(EdgeMap); %iMaxEdges stores the total edges in the image.
[~,iMaxCorners] = size(CornerMap); %iMaxCorners stores the total corners in the image
GBest = PtAll;
for i1 = 1:size(PtAll,1)
    if strcmp(PtAll{i1,1},'c')
        Ptemp = CreateAntibodies(NAb,iMaxEdges,EdgeMap,EdgeImage,'c');
    elseif strcmp(PtAll{i1,1},'q')
        Ptemp = CreateAntibodies(NAb,iMaxCorners,CornerMap,EdgeImage,'q');
    elseif strcmp(PtAll{i1,1},'t')
        Ptemp = CreateAntibodies(NAb,iMaxCorners,CornerMap,EdgeImage,'t');
    else
        display('invalid shape encountered at pos1 CSA')
    end
    Ptemp = sortrows(Ptemp,-5);
    GBest(i1,2) = {Ptemp(1,:)};
    PtAll(i1,2) = {Ptemp};
end
for its = 1:Its
%     fprintf('%d\n%s %d\n%s %d\n%s %d\n',...
%             its,...
%             GBest{1,1},GBest{1,2}(1,5),...
%             GBest{2,1},GBest{2,2}(1,5),...
%             GBest{3,1},GBest{3,2}(1,5))
%
   if GBest{1,2}(1,5) == 1 && GBest{2,2}(1,5) == 1 && GBest{3,2}(1,5) == 1
%         fprintf('Perfect\n');
        break;
   end

   AkAll = {PtAll{1,1},PtAll{1,2}(1:NMAb,:)
         PtAll{2,1},PtAll{2,2}(1:NMAb,:)
         PtAll{3,1},PtAll{3,2}(1:NMAb,:)};

   for i2 = 1:size(AkAll,1)
       Shape = AkAll{i2,1};
       Ak = AkAll{i2,2};
```

```
53        if strcmp(Shape,'c')
54            iMax = iMaxEdges;
55            Map = EdgeMap;
56        elseif strcmp(Shape,'q')||strcmp(Shape,'t')
57            iMax = iMaxCorners;
58            Map = CornerMap;
59        else
60            display('invalid Shape in CSA pos2')
61        end
62        Ak = C_HM_CS(Ak,NC,iMax,Map,EdgeImage,P,Shape);
63        Ab_New = CreateAntibodies(NewAb,iMax,Map,EdgeImage,Shape);
64        Pt = [Ak;Ab_New];
65        Pt = sortrows(Pt,-5);
66        GBest(i2,2) = {Pt(1,:)};
67        PtAll(i2,2) = {Pt};
68    end
69 end
70     function Ak_New = C_HM_CS(Ak,NC,iMax,Map,EdgeImage,P,Shape)
71         Ak_New = NaN(size(Ak));
72         Den = sum(Ak(:,5));
73         if Den == 0
74             Den = 1e-12;
75         end
76
77         for i3 = 1:size(Ak_New,1)
78             Qk = round(NC*(Ak(i3,5)/Den));
79             Clones = ones(Qk+1,1);
80             Clones = Clones*Ak(i3,:);
81
82             MutProb = exp(-P*Ak(i3,5));
83             Ak_New(i3,:) = Mutate(Clones,iMax,MutProb,Map,EdgeImage,Shape);
84         end
85     end
86     function NewAk = Mutate(Clones,iMax,MutProb,Map,EdgeImage,Shape)
87         if strcmp(Shape,'c')||strcmp(Shape,'t')
88             n = 3;
89         elseif strcmp(Shape,'q')
90             n = 4;
91         else
92             display('Invalid shape in Mutate function')
93         end
94
95         Rws = size(Clones,1)-1;
96         i4 = 1;
97         while i4 <=Rws
98             tIndex = Clones(i4,1:n);
99             for i5 = 1:size(tIndex,2)
100                if rand <= MutProb
101                    tIndex(1,i5) = randi(iMax);
102                end
103            end
104            tIndex = sort(tIndex(1,1:n));
105            xy1 = Map{tIndex(1,1)};
106            xy2 = Map{tIndex(1,2)};
107            xy3 = Map{tIndex(1,3)};
```

```
108             Test = ~Collinear(xy1,xy2,xy3) && (numel(unique(tIndex)) == size(tIndex,2));
109             if strcmp(Shape,'q')
110                 xy4 = Map{tIndex(1,4)};
111                 Test = ~Collinear(xy1,xy2,xy3,xy4) && (numel(unique(tIndex)) == size(
                        tIndex,2));
112             end
113             if Test
114                 Clones(i4,1:n) = tIndex;
115                 if strcmp(Shape,'c')
116                     [x0, y0, R] = GetCenter(xy1, xy2, xy3);
117                     xy = MCA(x0, y0, R);
118                     Fitness = GetFitness(xy,EdgeImage);
119                     Clones(i4,5:8) = [Fitness,R,x0,y0];
120                 elseif strcmp(Shape,'q')
121                     [~,xyMat,~] = QuadLines(xy1,xy2,xy3,xy4);
122                     Fitness = GetFitness(xyMat,EdgeImage);
123                     Clones(i4,5:13) = [Fitness,xy1,xy2,xy3,xy4];
124                 elseif strcmp(Shape,'t')
125                     [~,xyMat,~] = TriangleLines(xy1,xy2,xy3);
126                     Fitness = GetFitness(xyMat,EdgeImage);
127                     Clones(i4,5:11) = [Fitness,xy1,xy2,xy3];
128                 else
129                     display('invalid shape in Mutate p2');
130                 end
131                 i4 = i4 + 1;
132             end
133         end
134         Clones = sortrows(Clones,-5);
135         NewAk = Clones(1,:);
136     end
137 toc
138 end
```

APPENDIX G

Detecting Distinct Instances of a Shape

```
function FMemAbs = AllDistinctShapes(tAtbs,s,rmin,Image)
Img = size(Image);
rmax = min(Img(1:2));
Esth = (rmax - rmin)/s;
Shps = size(tAtbs,1);
Threshold = [0.9;0.9;0.9];
DCount = zeros(Shps,1);
FMemAbs = cell(size(tAtbs));
for Shp = 1:Shps
    Shape = tAtbs{Shp,1};
    Atbs = tAtbs{Shp,2};
    ShapeAbs = Atbs(Atbs(:,5) >= Threshold(Shp),:);
    ShapeAbs = sortrows(ShapeAbs,-5);
    FinalMemAbs = NaN(size(ShapeAbs));
    if size(ShapeAbs,1) > 0
        FinalMemAbs(1,:) = ShapeAbs(1,:);
        DCount(Shp) = 1;
        Distinct = 1;
        for i = 2:size(ShapeAbs,1)
            tempAb = ShapeAbs(i,:);
            for i2 = 1:DCount(Shp)
                if strcmp(Shape,'c')
                    Esdi = sum(abs(tempAb(6:8) - FinalMemAbs(i2,6:8)));
                elseif strcmp(Shape,'q')
                    Esdi = sum(abs(tempAb(6:13) - FinalMemAbs(i2,6:13)));
                elseif strcmp(Shape,'t')
                    Esdi = sum(abs(tempAb(6:11) - FinalMemAbs(i2,6:11)));
                else
                    display('Invalid shape encountered in AllDistinctShapes.m');
                end
                if Esdi < Esth
                    Distinct = 0;
                    break;
                else
                    Distinct = 1;
                end
            end
            if(Distinct)
                DCount(Shp) = DCount(Shp) + 1;
                FinalMemAbs(DCount(Shp),:) = tempAb;
            end
        end
    end
    if DCount(Shp) > 0
        FinalMemAbs = FinalMemAbs(1:DCount(Shp),:);
    else
        FinalMemAbs = double.empty(0,size(FinalMemAbs,2));
    end
    FMemAbs{Shp,1} = Shape;
    FMemAbs{Shp,2} = FinalMemAbs;
end
FMemAbs
```

```
53 DrawShape(FMemAbs,Image);
54 end
```

APPENDIX H

Collinearity Check

```
function Linear = Collinear(xy1,xy2,xy3,varargin)
if length(varargin) == 1;
    xy4 = varargin{1};
    [xy1,xy2,xy3,xy4] = ReArrange(xy1,xy2,xy3,xy4);
end
MinDistance = 25;
xy1 = [xy1,0];
xy2 = [xy2,0];
xy3 = [xy3,0];
Points = 'Non Linear';
a31 = xy3 - xy1;
b312 = xy1 - xy2;
d312 = norm(cross(a31,b312)) / norm(a31);
if d312 < MinDistance
    Points = 'Linear';
end
a12 = -b312; %i.e. xy1-xy2
b123 = -a31; %i.e. xy3-xy1
d123 = norm(cross(a12,b123)) / norm(a12);
if d123 < MinDistance
    Points = 'Linear';
end
if length(varargin) == 1
    xy4 = [xy4,0];
    b314 = xy1 - xy4;
    d314 = norm(cross(a31,b314)) / norm(a31);
    if d314 < MinDistance
        Points = 'Linear';
    end
    b124 = xy1 - xy4;
    d124 = norm(cross(a12,b124)) / norm(a12);
    if d124 < MinDistance
        Points = 'Linear';
    end
    a43 = xy4 - xy3;
    b431 = xy4 - xy1;
    d431 = norm(cross(a43,b431)) / norm(a43);
    if d431 < MinDistance
        Points = 'Linear';
    end
    b432 = xy4 - xy2;
    d432 = norm(cross(a43,b432)) / norm(a43);
    if d432 < MinDistance
        Points = 'Linear';
    end
    a42 = xy4 - xy2;
    b421 = xy2 - xy1;
    d423 = norm(cross(a42,b421)) / norm(a43);
    if d423 < MinDistance
        Points = 'Linear';
    end
    b423  = xy2 - xy3;
```

```
53      d423 = norm(cross(a42,b423)) / norm(a42);
54      if d423 < MinDistance
55          Points = 'Linear';
56      end
57  end
58  Linear = strcmpi(Points,'Linear');
59  end
```

APPENDIX I

Creating New Antibodies

```
function Pt = CreateAntibodies(NAb,iMax,Map,EdgeImage,Shape)
if strcmp(Shape,'c')
    Pt = NaN(NAb,8);
elseif strcmp(Shape,'q')
    Pt = NaN(NAb,13);
elseif strcmp(Shape,'t')
    Pt = NaN(NAb,11);
else
    display('Invalid Shape in CreateAntibodies');
end
i = 1;
while i <= NAb
    if strcmp(Shape,'c')||strcmp(Shape,'t')
        tIndex = randi(iMax,1,3);
    elseif strcmp(Shape,'q')
        tIndex = randi(iMax,1,4);
    else
        display('Invalid Shape in CreateAntibodies');
    end
    tIndex = sort(tIndex);
    xy1 = Map{tIndex(1)}; %Extract xy coordinates
    xy2 = Map{tIndex(2)};
    xy3 = Map{tIndex(3)};
    if strcmp(Shape,'c')||strcmp(Shape,'t')
        Test = ~Collinear(xy1,xy2,xy3) && (numel(unique(tIndex)) == size(tIndex,2));
    elseif strcmp(Shape,'q')
        xy4 = Map{tIndex(4)};
        Test = ~Collinear(xy1,xy2,xy3,xy4) && (numel(unique(tIndex)) == size(tIndex,2));
    else
        display('Invalid Shape in pos2 CreateAntibodies');
    end
    if Test
        if strcmp(Shape,'c')
            [x0,y0,R] = GetCenter(xy1,xy2,xy3);
            xy = MCA(x0,y0,R);
            Fitness = GetFitness(xy,EdgeImage);
            Pt(i,:) = [tIndex,0,Fitness,R,x0,y0];
        elseif strcmp(Shape,'q')
            [~,xyMat,~] = QuadLines(xy1,xy2,xy3,xy4);
            Fitness = GetFitness(xyMat,EdgeImage);
            Pt(i,:) = [tIndex,Fitness,xy1,xy2,xy3,xy4];
        elseif strcmp(Shape,'t')
            [~,xyMat,~] = TriangleLines(xy1,xy2,xy3);
            Fitness = GetFitness(xyMat,EdgeImage);
            Pt(i,:) = [tIndex,0,Fitness,xy1,xy2,xy3];
        else
            display('Invalid shape in pos3 createantibodies')
        end
        i = i + 1;
    end
end
end
```

APPENDIX J

Drawing Shapes

```
function DrawShape(PtAll,Image)
imshow(Image);
hold on
for Shp = 1:size(PtAll,1)
    Shape = PtAll{Shp,1};
    Pt = PtAll{Shp,2};
    if ~isempty(Pt)
        ShapesCount = size(Pt,1);
        for i1 = 1:ShapesCount
            if strcmp(Shape,'c')
                xyMat = MCA(Pt(i1,7),Pt(i1,8),Pt(i1,6));
                Step = round(size(xyMat,1) / 8);
                plot(xyMat(1,2),xyMat(1,1),'*b',xyMat(Step,2),xyMat(Step,1)...
                    ,xyMat(2*Step,2),xyMat(2*Step,1),'*b',xyMat(3*Step,2),xyMat(3*Step,1)
                        ,'*b'...
                    ,xyMat(4*Step,2),xyMat(4*Step,1),'*b',xyMat(5*Step,2),xyMat(5*Step,1)
                        ,'*b'...
                    ,xyMat(6*Step,2),xyMat(6*Step,1),'*b',xyMat(7*Step,2),xyMat(7*Step,1)
                        ,'*b')
                xy = {xyMat};
            elseif strcmp(Shape,'q')
                [xy,~,~] = QuadLines([Pt(i1,6),Pt(i1,7)],[Pt(i1,8),Pt(i1,9)],...
                                    [Pt(i1,10),Pt(i1,11)],[Pt(i1,12),Pt(i1,13)]);
                plot(Pt(i1,6),Pt(i1,7),'*b',Pt(i1,8),Pt(i1,9),'*b',Pt(i1,10),Pt(i1,11)...
                    ,'*b',Pt(i1,12),Pt(i1,13),'*b')
            elseif strcmp(Shape,'t')
                xy = {[MLA([Pt(i1,6),Pt(i1,7)],[Pt(i1,8),Pt(i1,9)])
                      MLA([Pt(i1,8),Pt(i1,9)],[Pt(i1,10),Pt(i1,11)])
                      MLA([Pt(i1,6),Pt(i1,7)],[Pt(i1,10),Pt(i1,11)])]};
                plot(Pt(i1,6),Pt(i1,7),'*b',Pt(i1,8),Pt(i1,9),'*b',Pt(i1,10),Pt(i1,11),'*
                    b')
            else
                display('invalid shape in drawshape')
            end
            for i2 = 1:size(xy,1)
                plot(xy{i2}(:,2),xy{i2}(:,1),'b')
            end
        end
    end
end
hold off
end
```

APPENDIX K

Get Circle Centre

```
1 function [Xo,Yo,R] = GetCenter(P1, P2, P3)
2     x1 = P1(1);     y1 = P1(2);
3     x2 = P2(1);     y2 = P2(2);
4     x3 = P3(1);     y3 = P3(2);
5     A = [2*(x1 - x2) 2*(y1 - y2);2*(x1 - x3) 2*(y1 - y3)]\[x1^2 - x2^2 + y1^2 - y2^2;x1^2
          - x3^2 + y1^2 - y3^2];
6     Xo = round(A(1));
7     Yo = round(A(2));
8     R  = round(((Xo - x3)^2 + (Yo - y3)^2)^(1/2));
9 end
```

APPENDIX L

Computing Fitness

```
function Fitness = GetFitness(xy,EdgeImage)
NRows = size(xy,1);
[xMax,yMax] = size(EdgeImage);
F_Sum = 0; % Counts Shape edges present in actual image
for i = 1:NRows
    if(1 <=xy(i,1)) && (xy(i,1) <= xMax)
        if (1 <= xy(i,2)) && (xy(i,2) <= yMax)
            if EdgeImage(xy(i,1),xy(i,2))
                F_Sum = F_Sum + 1;
            elseif xy(i,1) > 1
                if EdgeImage(xy(i,1)-1,xy(i,2))

                    F_Sum = F_Sum + 1;
                elseif xy(i,2) < yMax
                    if EdgeImage(xy(i,1)-1,xy(i,2)+1)
                        F_Sum = F_Sum + 1;
                    end
                end
            elseif xy(i,2) > 1
                if EdgeImage(xy(i,1),xy(i,2)-1)
                    F_Sum = F_Sum + 1;
                elseif xy(i,1) < xMax
                    if EdgeImage(xy(i,1)+1,xy(i,2)-1)
                        F_Sum = F_Sum + 1;
                    end
                end
            elseif (xy(i,1) > 1) && (xy(i,2) > 1)
                if EdgeImage(xy(i,1)-1,xy(i,2)-1)
                    F_Sum = F_Sum + 1;
                end
            elseif xy(i,1) < xMax
                if EdgeImage(xy(i,1)+1,xy(i,2))
                    F_Sum = F_Sum + 1;
                end
            elseif xy(i,2) < yMax
                if EdgeImage(xy(i,1),xy(i,2)+1)
                    F_Sum = F_Sum +1;
                end
            elseif (xy(i,2) < yMax) && (xy(i,1) < xMax)
                if EdgeImage(xy(i,1)+1,xy(i,2)+1)
                    F_Sum = F_Sum + 1;
                end
            end
        end
    end
end
Fitness = F_Sum/NRows;
end
```

APPENDIX M

Detecting Intersection of Two Lines

```
% Adapted from the code by Paulo Silva available
% at:http://uk.mathworks.com/matlabcentral/fileexchange/30502-find-intersection-of-two-lines/content/lineintersect.m
function Diagonal = LineIntersect(L1,L2)
Diagonal = 1;
mL1 = (L1(4) - L1(2)) / (L1(3) - L1(1));
mL2 = (L2(4) - L2(2)) / (L2(3) - L2(1));
if mL1==mL2
    Status = 'parallel';
    Diagonal = strcmpi(Status,'yes');
end
    bL1 = L1(2) - mL1 * L1(1);
    bL2 = L2(2) - mL2 * L2(1);
b = [bL1 bL2]';
a = [1 -mL1; 1 -mL2];
Pint = round(a\b);
xi = Pint(2);
yi = Pint(1);
L1minX = min([L1(1) L1(3)]);
L2minX = min([L2(1) L2(3)]);
L1minY = min([L1(2) L1(4)]);
L2minY = min([L2(2) L2(4)]);
L1maxX = max([L1(1) L1(3)]);
L2maxX = max([L2(1) L2(3)]);
L1maxY = max([L1(2) L1(4)]);
L2maxY = max([L2(2) L2(4)]);
if ((xi<L1minX) || (xi>L1maxX) || (yi<L1minY) || (yi>L1maxY) ||...
        (xi<L2minX) || (xi>L2maxX) || (yi<L2minY) || (yi>L2maxY))
    Status = 'no';
    Diagonal = strcmpi(Status,'yes');
end
end
```

APPENDIX N

Midpoint Circle Algorithm

```
% Adapted from the midpoint circle algorithm implementation by Jean-Yves
% Available
% at: http://www.mathworks.com/matlabcentral/profile/authors/858345-jean-yves-tinevez
function xy = MCA(x0, y0, radius)
octant_size = floor((sqrt(2)*(radius - 1) + 4)/2);
n_points = 8 * octant_size;
xc = NaN(n_points, 1);
yc = NaN(n_points, 1);
x = 0;
y = radius;
f = 1 - radius;
dx = 1;
dy = - 2 * radius;
xc(1) = x0 + x;
yc(1) = y0 + y;
     xc(8 * octant_size) = x0 - x;
     yc(8 * octant_size) = y0 + y;
     xc(4 * octant_size) = x0 + x;
     yc(4 * octant_size) = y0 - y;
     xc(4 * octant_size + 1) = x0 - x;
     yc(4 * octant_size + 1) = y0 - y;
     xc(2 * octant_size) = x0 + y;
     yc(2 * octant_size) = y0 + x;
     xc(6 * octant_size + 1) = x0 - y;
     yc(6 * octant_size + 1) = y0 + x;
     xc(2 * octant_size + 1) = x0 + y;
     yc(2 * octant_size + 1) = y0 - x;
     xc(6 * octant_size) = x0 - y;
     yc(6 * octant_size) = y0 - x;
     for i = 2 : n_points/8
         if f > 0
             y = y - 1;
             dy = dy + 2;
             f = f + dy;
         end
         x = x + 1;
         dx = dx + 2;
         f = f + dx;
         xc(i) = x0 + x;
         yc(i) = y0 + y;
         xc(8 * octant_size - i + 1) = x0 - x;
         yc(8 * octant_size - i + 1) = y0 + y;
         xc(4 * octant_size - i + 1) = x0 + x;
         yc(4 * octant_size - i + 1) = y0 - y;
         xc(4 * octant_size + i) = x0 - x;
         yc(4 * octant_size + i) = y0 - y;
         xc(2 * octant_size - i + 1) = x0 + y;
         yc(2 * octant_size - i + 1) = y0 + x;
         xc(6 * octant_size + i) = x0 - y;
         yc(6 * octant_size + i) = y0 + x;
         xc(2 * octant_size + i) = x0 + y;
         yc(2 * octant_size + i) = y0 - x;
```

```
53          xc(6 * octant_size - i + 1) = x0 - y;
54          yc(6 * octant_size - i + 1) = y0 - x;
55      end
56      xy = [xc,yc];
57  end
```

APPENDIX O

Midpoint Line Algorithm

```
% Adapted from Chandan Kumar's implementation
% Available at:
% http://uk.mathworks.com/matlabcentral/fileexchange/25544-line-drawing-by-bresenham-algorithm/content/bresenham_line.m
function   xy = MLA(xy1,xy2)
if (abs(xy2(2)-xy1(2)) > abs(xy2(1) - xy1(1)))
    x0 = xy1(2); y0 = xy1(1);
    x1 = xy2(2); y1 = xy2(1);
    token = 1;
else
    x0 = xy1(1); y0 = xy1(2);
    x1 = xy2(1); y1 = xy2(2);
    token = 0;
end
if(x0 > x1)
    temp1 = x0; x0 = x1; x1 = temp1;
    temp2 = y0; y0 = y1; y1 = temp2;
end
dx = abs(x1 - x0);
dy = abs(y1 - y0);
sx = sign(x1 - x0);
sy = sign(y1 - y0);
x = x0; y = y0;
param = 2*dy - dx;
x_coord = NaN(dx+1,1);
y_coord = NaN(dx+1,1);
for i = 1:dx+1
    x_coord(i) = x;
    y_coord(i) = y;
    param = param + 2*dy;
    if (param > 0)
        y = y + sy;
        param = param - 2*dx;
    end
    x = x + sx;
end
if token == 1
    xy = [x_coord,y_coord];
else
    xy = [y_coord,x_coord];
end
end
```

APPENDIX P

Preprocessing Image for CSA

```
function [CornerImage,CornerMap,EdgeImage,EdgeMap,Image] = PrepImage()
    [F,P] = uigetfile('*.pgm;*.jpg;*.jpeg;*.tif;*.tiff;*.bmp;*.png;*.hdf;*.pcx;*.xwd','
        Choose Image');
    if F == 0
        display('No file selected')
        return;
    end
        PF = [P,F];
        ext = PF(strfind(PF,'.')+1:end);
        if strcmp(ext,'pgm')
            Image = readpgm(PF);
        else
            Image = imread(PF);
        end
        Image2 = imgaussfilt(Image,0.94);
    if ndims(Image)== 3
        GrayImage = rgb2gray(Image);
        GrayImage2 = rgb2gray(Image2);
    elseif ismatrix(Image)
        GrayImage = Image;
        GrayImage2 = Image2;
    else
        fprintf('Image has too many dimensions\n');
        return;
    end
    EdgeImage = edge(GrayImage);
    EdgeMap = cell(0);
    [Row, Col] = size(EdgeImage);
    EdgeCount = 0;
    for i = 1:Row
        for j = 1:Col
            if EdgeImage(i,j)
                EdgeCount = EdgeCount + 1;
                EdgeMap(EdgeCount) = {[i,j]};
            end
        end
    end
    TempCornerImage1 = corner(GrayImage2,'Harris');
    TempCornerImage1 = sortrows(TempCornerImage1,1);
    TempCornerImage2 = zeros(size(GrayImage));
    CornerMap = cell(0);
    Row1 = size(TempCornerImage1,1);
    for i1 = 1:Row1
        CornerMap(i1) = {TempCornerImage1(i1,:)};
        TempCornerImage2(TempCornerImage1(i1,2),TempCornerImage1(i1,1))=1;
    end
    CornerImage = logical(TempCornerImage2);
    if strcmp(ext,'png')
        Image(Image==0)=255;
    end
end
```

APPENDIX Q

Get Bounding Edges of Quadrilateral

```
% Returns the xy coordinates of a Quadrilateral.
function [xyCell,xyMat,EdgeCount] = QuadLines(xy1,xy2,xy3,xy4)
[xy1,xy2,xy3,xy4] = ReArrange(xy1,xy2,xy3,xy4);
xy_12 = (MLA(xy1,xy2));
xy_23 = (MLA(xy2,xy3));
xy_34 = (MLA(xy3,xy4));
xy_41 = (MLA(xy4,xy1));
xyCell = {xy_12;xy_23;xy_34;xy_41};
xyMat = [xy_12;xy_23;xy_34;xy_41];
EdgeCount = size(xyMat,1);
end
```

APPENDIX R

Get Bounding Edges of Triangle

```
% Returns the xy coordinates of a triangle.
function [xyCell,xyMat,EdgeCount] = TriangleLines(xy1,xy2,xy3)
xy_12 = (MLA(xy1,xy2));
xy_23 = (MLA(xy2,xy3));
xy_31 = (MLA(xy3,xy1));
xyCell = {xy_12;xy_23;xy_31};
xyMat = [xy_12;xy_23;xy_31];
EdgeCount = size(xyMat,1);
end
```

APPENDIX S

Get Correct Vertex Arrangement for Quadrilaterals

```
function [xy1, xy2, xy3, xy4] = ReArrange(A,B,C,D)
if LineIntersect([A,C],[B,D])
    xy1 = A;
    xy2 = B;
    xy3 = C;
    xy4 = D;
elseif LineIntersect([A,B],[C,D])
    xy1 = A;
    xy2 = C;
    xy3 = B;
    xy4 = D;
else
    xy1 = A;
    xy2 = B;
    xy3 = D;
    xy4 = C;
end
end
```