MOVING OBJECT DETECTION USING CELLULAR NEURAL NETWORK (CNN)

PREMA LATHA SUBRAMANIAM

This thesis is submitted as partial fulfillment of the requirements for the award of the Bachelor Degree of Electrical Engineering (Control and Instrumentation)

Faculty of Electrical & Electronics Engineering
University Malaysia Pahang

NOVEMBER, 2008

MOVING OBJECT DETECTION USING CELLULAR NEURAL NETWORK (CNN)

PREMA LATHA SUBRAMANIAM

UNIVERSITY MALAYSIA PAHANG

"I hereby acknowledge that the scope and quality of this thesis is qualified for the award of the Bachelor Degree of Electrical Engineering (Control and Instrumentation)"

Signature : _____

Name : AMRAN BIN ABDUL HADI

Date : 14 NOVEMBER 2008

Signature : _____

Author : PREMA LATHA SUBRAMANIAM

Date : 15 NOVEMBER 2008

Specially dedicated to
my beloved parents and best friends for their full support
and love throughout my journey of education.

ACKNOWLEDGEMENT

I would like to thank my parents for their love, support and patience during the year of my study. I also would like to take this opportunity to express my deepest gratitude to my supervisor, En Amran bin Abdul Hadi for his patience and guidance in preparing this paper. Special thanks to all my friends who have directly or indirectly have contributed to my success in completing this thesis. Last but not least, I would like to thank God for being within me.

ABSTRACT

Detecting moving objects is a key component of an automatic visual surveillance and tracking system. Previous motion-based moving object detection approaches often use background subtraction and inter-frame difference or three-frame difference, which are complicated and takes long time. In this paper, we proposed a simple and fast method to detect a moving object using Cellular Neural Network. The main idea in Cellular Neural Network is that connection is allowed between adjacent units only. This paper comprises the implementation of the basic templates available in Cellular Neural Network. The templates are programmed in MATLAB. There are few rules in Cellular Neural Network that has to be implemented when programming the templates, such as the state equation, output equation, boundary condition and also the initial value. These templates are combined to create the most ideal algorithm to detect a moving object in an image. A video of a bouncing ball is recorded using a static camera. The video then are segmented into images using SC Video Developer. Ten images are selected to be used in this project. The algorithm created is used to detect the ball in the images. This paper also includes the use of Image Processing Toolbox in MATLAB. An analysis is conducted by comparing the ball's position in each image according to the time. This analysis indicates whether the object has shifted position or moved in the images. The efficiency of the result for this paper is 85%.

ABSTRAK

Mengesan pergerakan objek ialah satu komponen yang penting dalam sistem pengawasan automatik dan sistem pengesanan pergerakan. Kaedah pengesanan pergerakan objek yang sedia ada sering menggunakan cara penyingkiran latar belakang dan perbezaan antara lapisan di mana kaedah tersebut rumit dan mengambil masa yang lama. Untuk projek ini, kaedah yang lebih mudah dan pantas dicadangkan untuk mengesan pergerakan objek dengan menggunakan Cellular Neural Network. Sifat Cellular Neural Network yang utama ialah kebolehan sel-sel bersebelahan atau setempat berkomunikasi atau berinteraksi dengan sel-sel jiran. Projek ini mengaplikasikan model klon asas yang terdapat di dalam Cellular Neural Network. Model klon tersebut diprogramkan dengan menggunakan perisian MATLAB. Terdapat beberapa peraturan yang harus diambil kira dan dipatuhi semasa membuat pemprograman untuk model klon seperti persamaan keadaan, persamaan hasil, keadaan sempadan dan nilai awal. Model-model klon yang dihasilkan digabungkan bersama untuk mencipta satu algoritma yang sesuai untuk mengesan pergerakan objek di dalam imej. Satu rakaman video yang menunjukkan pergerakan bola yang melantun direkodkan dengan menggunakan kamera statik. Rakaman video ini kemudian disegmentasikan dengan menggunakan perisian SC Video Developer. Sepuluh imej dipilih untuk digunakan dalam projek ini. Algoritma yang dicipta digunakan untuk mengesan pergerakan bola dalam imej-imej tersebut. Projek ini juga mengaplikasikan Image Processing Toolbox yang terdapat di dalam perisian MATLAB. Analisis yang menunjukkan perbandingan kedudukan atau koordinat bola di dalam imej-imej tersebut dihasilkan. Tahap ketepatan keputusan untuk projek ini ialah 85%.

TABLE OF CONTENTS

LIST OF TABLE

LIST OF FIGURE

LIST OF APPENDICES

CHAPTER I

INTRODUCTION

1.1 Overview

Moving object detection is always an important task in this world of technology. Moving object detection plays an important role in automatic visual surveillance, tracking system and also to avoid collision.

Cellular Neural Network was invented by Leon O. Chun and Lin Yang in Berkeley in 1988. In Cellular Neural Network, the time is continuous and the interaction values are real values. Each processing cell interacts or communicates with its nearest neighbouring cells through a program or an algorithm. Cells are only connected within a certain neighbourhood but not to the entire network, thus it is easy for extension without readjusting the whole network. Due to this, Cellular Neural Network can be used in applications such as high speed target recognition, real-time visual inspection of manufacturing process and also any brain-like information processing tasks.

This thesis implements the basic templates from Cellular Neural Network in creating an algorithm using MATLAB as the programming platform. The process starts by recoding a moving ball or bouncing ball video using stationary camera. Then, the images are edited using Image Processing Toolbox in MATLAB. Templates are created using MATLAB and then an ideal algorithm is selected to detect the moving object. An analysis comparing the object previous and new position is done.

1.2 Objectives

The objectives of this program are:

i. To understand the concept of Cellular Neural Network and its application.

In this project, the concept of Cellular Neural Network must be understood in order to apply it. The concept of Cellular Neural Network is its characteristic and the way it works in certain condition. The characteristic of Cellular Neural Network is elaborated in detail in the literature review. Cellular Neural Network has a lot of applications and it can be used in most of electric and electronic projects instead of the traditional methods used before.

ii. To detect a moving object captured by static camera using an algorithm developed in Cellular Neural Network.

In this project, the main idea is to detect the moving object or the motion of an object. A static camera will capture or in another word, record a moving object. The moving object is then detected using Cellular Neural Network. In detail, an algorithm is developed using Cellular Neural Network templates which can be used to detect a motion. This algorithm is actually a combination of several templates available in Cellular Neural Network. This templates and its application is explained in the literature review.

1.3 Scope

i. An algorithm is developed based on templates in Cellular Neural Network and simulates the programming in MATLAB to detect moving object.

An algorithm is developed using the templates in Cellular Neural Network. This algorithm is applied through MATLAB to detect a moving object. The programming is done is MATLAB and by simulating the programming, the moving object will be detected.

ii. An analyze of the object's positions in the images according to the time.

To make image segmentation and analyze the object's position in each images using MATLAB. The object is identified using coordinate system. A comparison of the object's previous and current coordinates will be done to indicate the movement of the object in the images.

1.4 Problem Statement

i. Moving object detection requires real time processing, which is fast, thus Cellular Neural Network is excellent choice as it is a parallel paradigm which provides fast processing.

ii. Compared to existing method, such as spatio-temporal constraints, it takes longer time to detect moving object and more complicated than Cellular Neural Network.

1.5 Thesis Outline

This thesis consists of five chapters. Chapter I cover on the introduction of the project, objectives of project, scopes of project and also the problem statement. Chapter II is mainly about the literature review done for this project. This chapter discusses the Cellular Neural Network, its basic notation and definitions, application of Cellular Neural Network in different areas, and also the main or basic templates in Cellular Neural Network. Chapter III focuses on the methodology for the whole project and also methodology on the Cellular Neural Network template programming architecture. Chapter IV shows the results obtained from this project, analysis and also the discussion. The last chapter, Chapter V consists of the conclusion of the project, recommendation for further development and also cost and commercialization of this project.

CHAPTER II

LITERATURE REVIEW

2.1 Cellular Neural Network

Cellular Neural Network is also known as Nonlinear Neural Network or CNNs. The Cellular Neural Network was invented by Leon O. Chua and Lin Yang in Berkeley in 1988. Cellular Neural Network is an array of analog dynamic processors or cells. Cellular Neural Network host processors accept and generate analog signals. Other than that, the interaction values are also real values. Moreover, the input of the Cellular Neural Network array plays an important role as Cellular Neural Network becomes rigorous framework for complex systems exhibiting emergent behavior and the various forms of emergent computations.

The Cellular Neural Network Universal Chip is a milestone in information technology because it is the first operational, fully programmable industrial-size brain-like stored-program dynamic array computer in the world. Each Cellular Neural Network cell is interfaced with its nearest neighbours and this massively parallel focal-array computer is capable of processing 3 trillion equivalent digital per operations per second (in analog mode), a performance which can be matched only by supercomputers. In terms of SPA (power, speed, area) measures, this Cellular Neural Network Universal chip is far superior to any equivalent DSP implementation by at least three orders of magnitude. The applications include high-speed track target recognition and tracking, real-time visual inspection of manufacturing processes, intelligence vision capable of

recognizing context sensitive and moving scenes, as well as applications requiring real-time fusing of multiple modalities, such as multispectral images involving visible, infrared, long-wave infrared, and polarized lights [1].

Cellular Neural Network is a parallel computing paradigm defined in discrete N-dimensional spaces. Cellular Neural Network is an N-dimensional regular array of elements or cells. A standard Cellular Neural Network architecture consists of an M x N rectangular cells (C(i, j)) with Cartesian coordinates (i, j). The cell grid can be for example, a planar array with rectangular, triangular or hexagonal geometry, a 2-D or 3-D torus, a 3-D finite array, or a 3-D sequence of 2-D arrays. Cells are multiple input-single output processors, all described by one or just some few parametric functional. A cell is characterized by an internal state variable, sometimes not directly observable from outside the cell itself. More than one connection network can be present, with different neighbourhood sizes.

Cellular Neural Network is a system of cells defined on a normalized space. In the system, cell is the basic circuit unit containing linear and nonlinear circuit element, which are linear capacitors, linear resistors, linear and nonlinear controlled sources and independent sources. The main idea is that the connection is allowed between adjacent units only. Any cell in the Cellular Neural Network is connected to only its neighbour cells. But cells can affect each other indirectly. The propagation effects of the continuous time dynamics of the Cellular Neural Network provides the interaction between cells in space.

A Cellular Neural Network dynamical system can operate both in continuous (CT-CNN) or discrete time (DT-CNN). Cellular Neural Network data and parameters are typically continuous values. Cellular Neural Network operate typically with more than one iteration, they are recurrent networks. Cellular Neural Network main characteristic is the locality of the connections between the units. In fact the main

difference between Cellular Neural Network and other Neural Networks paradigms is the fact that information is directly exchanged just between neighbouring units. This characteristic allows also obtaining global processing. Communications between non directly (remote) connected units are obtained passing through other units.

It is possible to consider the Cellular Neural Network paradigm as an evolution of Cellular Automata paradigm. Moreover it has been demonstrated that Cellular Neural Network paradigm is universal, being equivalent to the Turing Machine.

2.2 Basic Notations and Definition

2.2.1 Standard CNN architecture

A standard CNN architecture consists of an M × N rectangular array of cells (C(i, j)) with Cartesian coordinates (i, j), i = 1, 2,…M, j = 1, 2…N

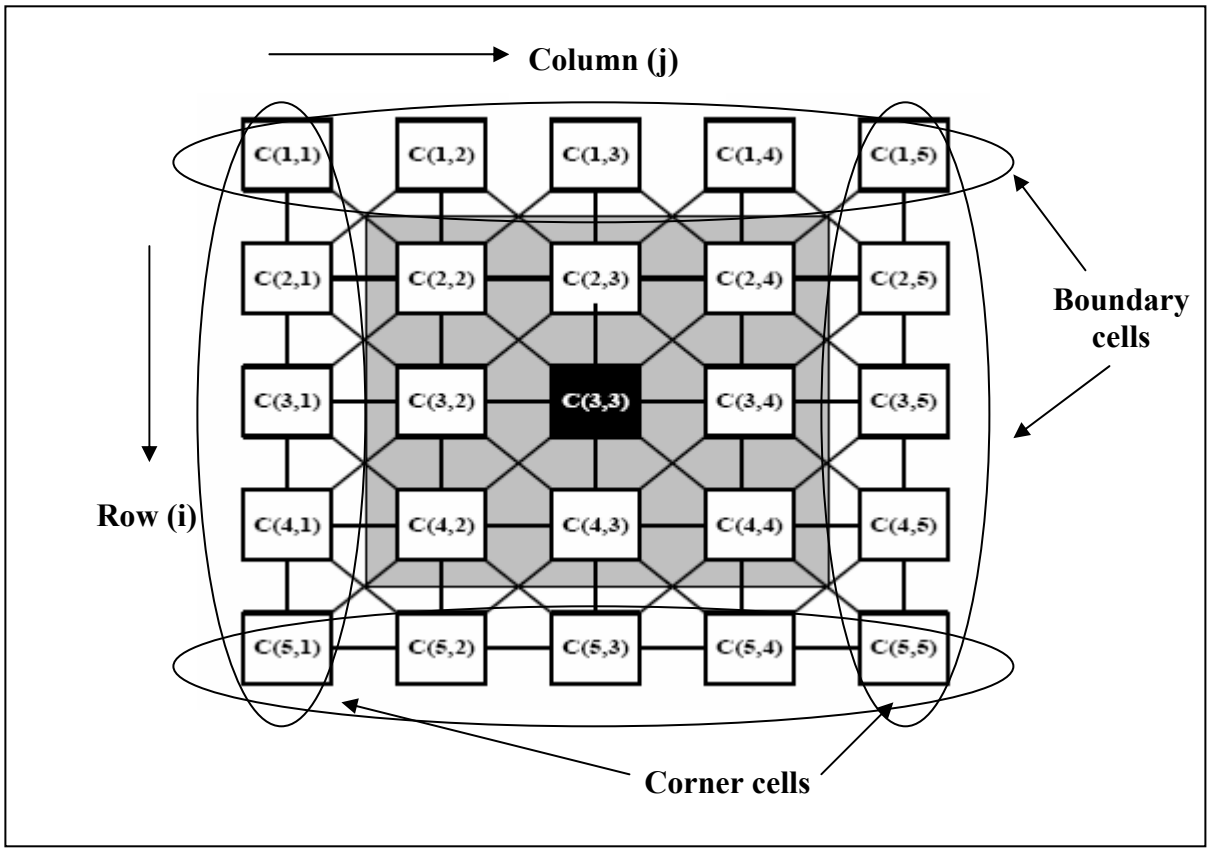


Figure 2.1: Standard CNN 5 X 5 Architecture

2.2.2 Sphere of Influence of Cell C(i, j)

The sphere of influence, Sr(i, j), of the radius r of cell C(i, j) is defined to be the set of all the neighbourhood cells satisfying the following property.

$$Sr(i,j) = \{C(k,l) \mid \max\{|k-i|,|l-j|\} \le r\}$$
$$1 \le k \le M, 1 \le l \le N$$

where r is a positive integer

2.2.3 Regular and Boundary Cells

A cell C(i, j) is called regular cell with respect to Sr(i, j) if and only if all neighbourhood cells $C(k,l) \in Sr(i,j)$ exist. Otherwise, C(i, j) is called a boundary cell.

2.2.4 Standard CNN

A class 1 M × N standard CNN is defined by M × N rectangular array of cells C(i,j) located at side (i, j), i = 1, 2,…M, j = 1, 2…N. Each cell C(i, j) is defined mathematically by:

Definition 1: State equation

$$\dot{x}ij = -xij + \sum_{C(k,l) \in Sr(i,j)} \mathrm{A}(i,j;k,l)ykl + \sum_{C(k,l) \in Sr(i,j)} \mathrm{B}(i,j;k,l)ukl + \mathrm{Z}ij$$

where $xij \in R$, $ykl \in R$, $ukl \in R$ and $Zij \in R$ are called state, output, input and threshold of cell C(i, j) respectively. $A(i, j; k, l)$ and $B(i, j; k, l)$ are called the feedback.

Definition 2: Output equation

$$yij = f(xij) = \frac{1}{2}|xij + 1| - \frac{1}{2}|xij - 1|$$

This is called the standard nonlinearity.

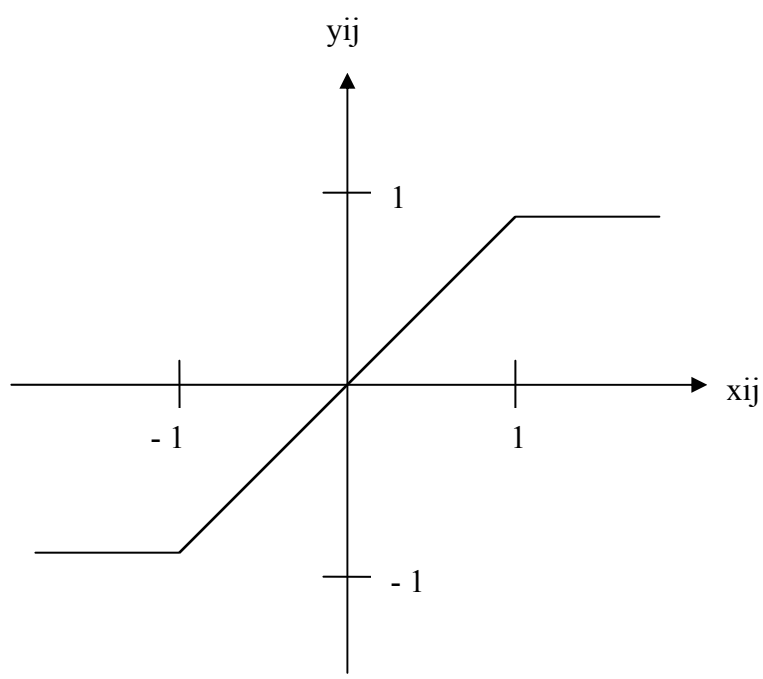


Figure 2.2: Standard Nonlinearity

Definition 3: Boundary Conditions

The boundary conditions are those specifying ykl and ukl for each cells belonging to $Sr(i, j)$ of edge cells but lying outside of M x N array.

Definition 4: Initial state

$$xij(0), \qquad i = 1,\ldots,M, \quad j = 1,\ldots,N$$

2.3 Applications

There are many applications of Cellular Neural Network especially in healthcare. For example, Cellular Neural Network is used in clinical diagnosis known as Papnet. Papnet is a commercial Cellular Neural Network based computer program for assisting screening of Pap (cervical) smears. In this Pap smear test, cells taken from uterine cervix are examined for signs of precancerous and cancerous changes. If detected early, cervical cancer has an almost 100% chance of cure. The traditional method, which is relying on human eyes to detect abnormal cells under microscope, has difficulty in detecting cancer in early stage. Since a patient with a serious abnormality can have fewer than a dozen abnormal cells among the 30,000 - 50,000 normal cells on her Pap smear, it is very difficult to detect all cases of early cancer by this "needle-in-a-haystack" search [2]. Using Cellular Neural Network results in more accurate screening process thus, leading to an earlier and more effective detection of pre-cancerous cells in the cervix.

Other than that, Cellular Neural Network is also used in image analysis and interpretation particularly in medicine. Pattern recognition is widely used to identify and extract important features in radiographies, ECTs or MRIs. Filtering, segmentation and edge detection techniques using Cellular Neural Network improves resolution in brain tomographies, and also improves global frequency correction for the detection of microcalcifications in mammograms. Furthermore, under healthcare, Cellular Neural Network is also used in signal analysis and interpretation and drug development.

Cellular Neural Network is also used in other applications beside healthcare. For instance Cellular Neural Network is used in lip reading. The three main parts of the system include a face tracker, lip modeling and speech processing. Automatic speech

reading is based on a robust lip image analysis. The analysis is based on truecolor video images. The system allows for real-time tracking and storage of the lip region and robust off-line lip model matching. A neural classifier detects visibility of teeth edges and other attributes. At this stage of the approach, the edge closed lips is automatically modeled if applicable is based on neural network's decision.

To achieve high flexibility during lip-model development, a model description language has been defined and implemented. The language allows the definition of edge models (in general) based on knots and edge functions. Inner model forces stabilize the overall model shape. User defined image processing functions may be applied along the model edges. These functions and the inner forces contribute to an overall energy function. Adaptation of the model is done by gradient descent or simulated annealing like algorithms.

Another application of Cellular Neural Network is detecting and tracking of moving targets. The moving target detection and track methods here are "track before detect" methods. They correlate sensor data versus time and location, based on the nature of actual tracks. Compared to conventional fixed matched filter techniques, these methods have been shown to reduce false alarm rates by up to a factor of 1000 based on simulated SBIRS data for very weak ICBM targets against cloud and nuclear backgrounds, with photon, quantization, and thermal noise, and sensor jitter included.

The methods are designed to overcome the weaknesses of other advanced track-before-detect methods, such as 3+-D matched filtering, dynamic programming (DP), and multi-hypothesis tracking (MHT). Loosely speaking, 3+-D matched filtering requires too many filters in practice for long-term track correlation. DP cannot realistically exploit the non-Markovian nature of real tracks, and strong targets mask out weak targets, and MHT cannot support the low pre-detection thresholds required for very

weak targets in high clutter. They have developed and tested versions of the above (and other) methods in their research, as well as Kalman-filter probabilistic data association (KF/PDA) methods, which they use for post-detection tracking. Space-time-adaptive methods are used to deal with correlated, non-stationary, non-Gaussian clutter, followed by a multi-stage filter sequence and soft-thresholding units that combine current and prior sensor data, plus feed back of prior outputs, to estimate the probability of target presence.

Cellular Neural Network is also used in real-time target identification based for security application. The system localizes and tracks peoples' faces as they move through a scene. It integrates the techniques such as motion detection, tracking people based upon motion and tracking faces using an appearance model. Faces are tracked robustly by integrating motion and model-based tracking. Cellular Neural Network is also used in ATM network, noise reduction, finger print match, face recognition, biomedical and word sporting.

High speed detection and classification of the objects, symbols, and characters with an acceptable error rate is a task which is always considered when new computing architecture, suitable for image processing and pattern recognition. In Cellular Neural Network concept, the research area of these is locally connected, regularly repeated analog arrays have shown remarkable growth. Recently, based on the Cellular Neural Network paradigm, a universal hardware architecture has been designed, called the Cellular Neural Network Universal Machine. This new algorithmically programmable analog array computer is an ideal environment for "dual computing" for example to execute complex Cellular Neural Network analogic algorithms. In these algorithms, analog operations which are controlled by various Cellular Neural Network templates are combined with local logic on the cell level. Using this concept, complex decisions can be made on images without reading out the Cellular Neural Network chip which makes this method extremely time effective.

2.4 Templates

Different role of the control and feedback matrices in Cellular Neural Network templates is also applied to detect motion. In the past few years, several researchers also attempted character recognition by incorporating the Cellular Neural Network concept. T. Matsumoto presents some simple Cellular Neural Network templates for binary image processing. Later, these templates (horizontal, vertical, diagonal CCD and Shadow Detector) were used by Suzuki to introduce a new character recognition method by Cellular Neural Network preprocessing and a back propagation classification. T. Szirányi and J. Csicsvári completed the above mentioned templates by the Hole-Filler one and used a novel type of CNND architecture. K. Nakayama and Y. Chigawa used CNN for extracting line segment features (middle point, length and angle of the line segment) and for character recognition combined it with modified self-organizing feature mapping. Most of them turned out to be very efficient in recognition of distorted and translated patterns.

Each template has its own feedback, input synaptic, threshold values, boundary conditions and initial states that need to be fulfilled to obtain the results. The input and threshold are continuous functions of time according to the uniqueness theorem.

2.4.1 EDGE: Binary Edge Detection Template

A =

0	0	0
0	0	0
0	0	0

B =

-1	-1	-1
-1	8	-1
-1	-1	-1

z =

-1

Global Task

Given	: static binary image **P**
Input	: **U**(t) = **P**
Initial state	: **X**(0) = Arbitrary (in examples we choose $x_{ij}(0) = 0$)
Boundary conditions	: Fixed type, $u_{ij} = 0$, $y_{ij} = 0$ for all virtual cells, denoted by **[U]** = **[Y]** = [0]
Output	: **Y**(t) $\Rightarrow$ **Y**(∞) = Binary image showing all edges of **P** in black

Remark

The Edge CNN template is designed to work correctly for binary input images only. If **P** is a gray-scale image, **Y**(∞) will be in general be gray-scale where black pixels correspond to sharp edges, near-black pixels correspond to fuzzy edges, and near-white pixels correspond to noise.

2.4.2 CORNER: Convex Corner Detection Template

A =

0	0	0
0	2	0
0	0	0

B =

-1	-1	-1
-1	8	-1
-1	-1	-1

z =

-8.5

Global Task

Given : static binary image **P**

Input : **U**(t) = **P**

Initial state : **X**(0) = 0

Output : **Y**(t) $\Rightarrow$ **Y**(∞) = Binary image, where black pixels correspond to convex corners in **P** (where, roughly speaking, a black pixel is a convex corner if it is a part of a convex corner boundary line of the input image).

2.4.3 LOGNOT: Logic NOT and set complementation ($P \rightarrow \overline{P} = P^c$) template

A =

0	0	0
0	1	0
0	0	0

B =

0	0	0
0	-2	0
0	0	0

z =

0

Global Task

Given : static binary image **P**

Input : **U**(t) = **P**

Initial state : **X**(0) = 0

Output : **Y**(t) $\Rightarrow$ **Y**(∞) = Binary image where each pixel in **P** becomes white, and vice versa. In set-theoretic or logic notation: **Y**(∞) = $P^c = \overline{P}$, where the bar denotes the "Complement" or "Negation" operator.

2.4.4 LOGOR: Logic OR and set union $\cup$ (disjunction $\vee$) template

A =

0	0	0
0	3	0
0	0	0

B =

0	0	0
0	3	0
0	0	0

z =

2

Global Task

Given : two static binary image $\mathbf{P}_1$ and $\mathbf{P}_2$

Input : $\mathbf{U}(t) = \mathbf{P}_1$

Initial state : $\mathbf{X}(0) = \mathbf{P}_2$

Output : $\mathbf{Y}(t) \Rightarrow \mathbf{Y}(\infty)$ = Binary output of the logic operation **OR** between $\mathbf{P}_1$ and $\mathbf{P}_2$. In logic notation, $\mathbf{Y}(\infty) = \mathbf{P}1 \vee \mathbf{P}2$, where $\vee$ denotes the "disjunction" operator. In set-theoretic, $\mathbf{Y}(\infty) = \mathbf{P}1 \cup \mathbf{P}2$, where $\cup$ denotes the "set union" operator.

2.4.5 LOGAND: Logic AND and set intersection $\cap$ (conjunction $\wedge$) template

A =

0	0	0
0	1.5	0
0	0	0

B =

0	0	0
0	1.5	0
0	0	0

z =

-1.5

Global Task

Given : two static binary image $\mathbf{P}_1$ and $\mathbf{P}_2$

Input : $\mathbf{U}(t) = \mathbf{P}_1$

Initial state : $\mathbf{X}(0) = \mathbf{P}_2$

Output : $\mathbf{Y}(t) \Rightarrow \mathbf{Y}(\infty)$ = Binary output of the logic operation **"AND"** between $\mathbf{P}_1$ and $\mathbf{P}_2$. In logic notation, $\mathbf{Y}(\infty) = \mathbf{P}_1 \wedge \mathbf{P}_2$, where $\wedge$ denotes the "conjunction" operator. In set-theoretic notation, $\mathbf{Y}(\infty) = \mathbf{P}_1 \cap \mathbf{P}_2$, where $\cap$ denotes the "intersection" operator.

2.5 Moving Object Detection

Identifying moving object is a critical task in image and video segmentation, which is used in many computer vision applications such as remote sensing, video surveillance and traffic monitoring. In the research done, there are some common methods used to detect a moving object. One of the popular methods is using spatio-temporal constraints.

In spatio-temporal constraints method, there are many steps to be done before we can actually detect the object. Steps such as background removal or background subtraction, colour analysis, image depth, and object's shape analysis are necessary. Spatio-temporal databases deal with objects that change their location or shape over time. A typical example of spatio-temporal databases is moving objects in the D-dimensional space. Moving objects learn about their own location via location detection devices, such as GPS devices. Then, the objects report their locations to the server using the underlying communication network, like the wireless networks. The server stores the updates from the moving objects and keeps a history of the spatio-temporal coordinates of each moving object. In addition, the server stores additional information to help predict the future positions of moving objects. As can be seen above, the spatio-temporal constraints method is very complicated as there are many steps involved.

In recent years, using Cellular Neural Network in moving object detection has gained much popularity. In the research done for moving object detection, there are many ways to detect a moving object using Cellular Neural Network. The most commonly used type of Cellular Neural Network is Delayed Cellular Neural Network and Analogic Cellular Neural Network. Delayed Cellular Neural Network was first introduced in 1993 where it involves 2-D images. Moving object detection is the most appealing task in the field of image processing. In this Delayed Cellular Neural

Network, the study was focused on two parts, first is without considering motion and the second part detection of moving object. Besides Cellular Neural Network, processing of moving images requires the introduction of delay in the signals transmitted among the cells. In the Delayed Cellular Neural Network, the delay τ is introduced in the state equation.

In the Analogic Cellular Neural Network, the ring-coding is used for detection of moving object. An object can be described by drawing a few circles around a few initial points called a central point (o) and integrating the grayness in the created rings (g1, g2, g3) according to the formula where the object is placed in the (x,y) plane. This mapping of a few real numbers leads to its rotation invariant description. But before applying the ring-coding method, there are a few rules have to be clarified such as the maximum size of object considered, the amount of rings needed for the description, and how to calculate the inner and outer radius of individual rings.

Relying on the templates and methods, a more complex Cellular Neural Network can be created to detect the colour, size and rotation shape of the object. Furthermore, the speed, direction and depth of the motion can also be classified.

CHAPTER III

METHODOLOGY

3.1 Overview of Cellular Neural Network for moving object detection

Detecting moving objects is a key component in automatic visual surveillance and tracking system. Besides that, moving object detection is also used to avoid collision. In the development of technology, moving object detection has become an important task in many different areas of application.

In this project, the most vital part is to develop the templates available in Cellular Neural Network. Then, an algorithm created from these templates is designed to detect the motion of an object, in this case, it is a bouncing ball. In order to create the programming for the algorithm, there are several other processes to be learnt beforehand. This includes the process of understanding the Cellular Neural Network and its notations, basic knowledge of image processing and also basic programming language, C or C++.

The video of a bouncing ball is recorded using stationary camera, in this project a digital camera is placed at certain point. Then, the video recorded is segmented into images or frames using SC Video Developer. These images are then used as the input for Cellular Neural Network templates. But before using it as an input, the images have to be edited using the Image Processing Toolbox. The original images are big in size and in Windows Bitmap (bmp) format.

The images are resized and changed to binary images as most of the input for Cellular Neural Network templates are binary images. The images are resized because the time taken to run the templates will be effected by the image size. Then some Cellular Neural Network templates are created for experiment. These templates include the Edge Detection Template, Convex Corner Detection Template, Logic NOT Template, Logic OR Template and Logic AND Template. But only certain templates are chosen to create the ideal algorithm. The Cellular Neural Network template are programmed considering the initial state, boundary condition, pixel value, sphere of influence, feedback synaptic, input synaptic and the threshold value.

Finally, the analysis is done by image segmentation. This image segmentation here means the segmentation of the output images from the algorithm. The object or ball in the image is identified using coordinate system to indicate any movement of the ball. As the last step of analysis, pixel counting is done to clearly show the object in the images. In this part, the pixel of the areas containing the object will be different when compared to the image background.

This chapter discusses the methodology of the project step by step from the programming of templates until the image segmentation.

3.2 Research Methodology

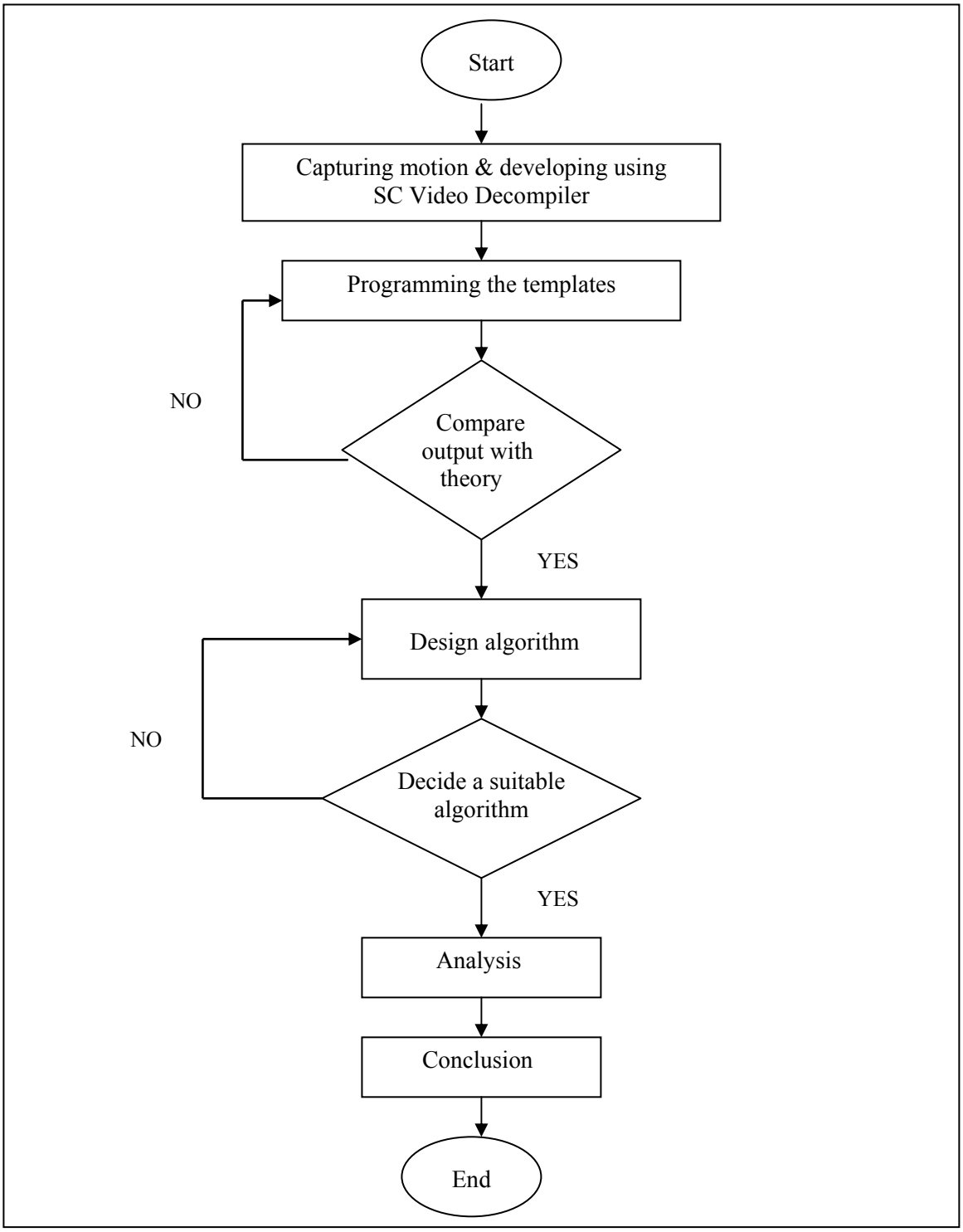


Figure 3.1: Project flowchart

3.3 System Design

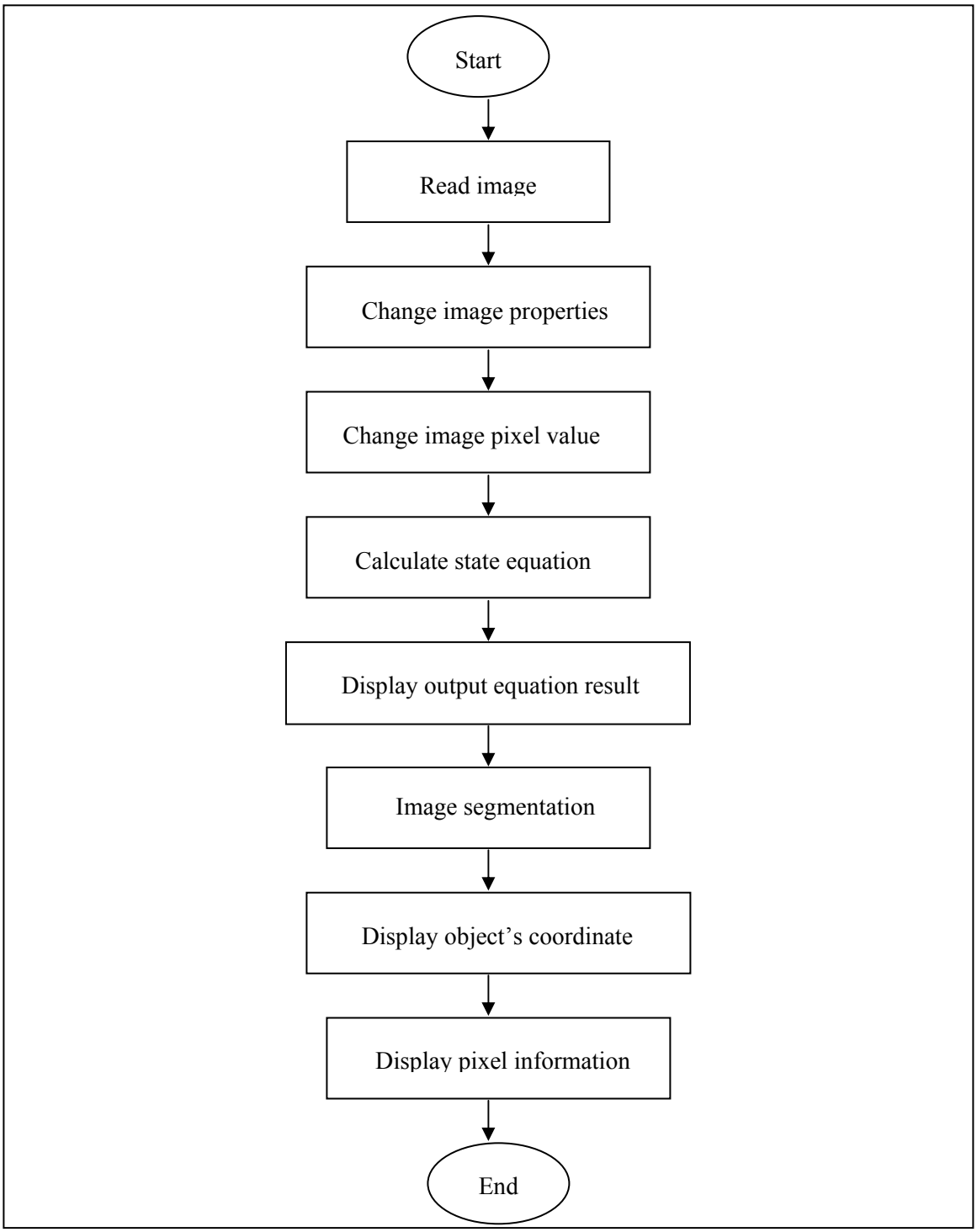


Figure 3.2: System Design

3.4 Step by step CNN simulation procedure in MATLAB

3.4.1 Image Initialization

Firstly, the M and N represents the size of array of the image. The size of the image can be determined by using the command **[M, N] = size (X)**, where it returns the size of matrix X in separate variables M and N. The M and N for this image has been reduced by 2 to obey the boundary conditions in the Cellular Neural Network.

Next, the feedback synaptic (A), input synaptic (B) and threshold (z) as given in the templates are initialized. Next, the image used is read by MATLAB. The image is read using the **imread** command. The example syntax is;

A = imread('filename.*fmt*')

Note that the text string *fmt* specify the format of the file by its standard file extension, For example, specify 'jpeg' for Joint Photographic Experts Group images.

The table below is the list of commonly used images and its standard file extensions.

Format Name	Description	Standard File Extension
BMP	Windows Bitmap	.bmp
GIF	Graphics Interchange Format	.gif
JPEG	Joint Photographic Experts Group	.jpg / .jpeg
PNG	Portable Network Graphics	.png
TIFF	Tagged Image File Format	.tif / .tiff

Table 3.1: Standard File Extension for Images

The image is then converted from matrix to grayscale image by using the **mat2gray** command. Then the image is converted to binary image using **im2bw** command.

The following table lists all the image type conversion functions in Image Processing Toolbox.

Function	Description
dither	Use dithering to convert a grayscale image to a binary image or to convert a truecolor image to an indexed image
gray2ind	Convert a grayscale image to an indexed image
grayslice	Convert a grayscale image to an indexed image by using multilevel thresholding
im2bw	Convert a grayscale image, indexed image, or truecolor image, to a binary image, based on a luminance threshold
ind2gray	Convert an indexed image to a grayscale image
ind2rgb	Convert an indexed image to a truecolor image
mat2gray	Convert a data matrix to a grayscale image, by scaling the data
rgb2gray	Convert a truecolor image to a grayscale image
rgb2ind	Convert a truecolor image to an indexed image

Table 3.2: Image Type Conversion Functions

Finally the image is converted to double precision. The example syntax;
double(x)

Note that it returns the double-precision value for x. If x is already a double-precision array, **double** has no effect.

```
%=====================================================================
%*********************PARAMETER DECLARATIONS************************
%*********************************************************************

clear all;% Clear variables and functions from memory
clc;% Clear command window

%-----------------------Constant Declarations----------------------

M=254; % No. of rows in the CNN structure
N=254; % No. of columns in the CNN structure

%-------------------------CNN template parameters------------------

A=[0 0 0; 0  1 0; 0 0 0]; % Feedback Operator
B=[0 0 0; 0 -2 0; 0 0 0]; % Input Synaptic Operator
z=[0];                    % Threshold  Value
X=zeros(256,256);  %CNN template initial state parameters


%=====================================================================
%*********************READ AND PREPARE DATA*************************
%*********************************************************************

READ_IMAGE=imread('pic4.jpeg'); % Read image from graphics file and
                                % map between 0 and 255

INST_IMAGE=mat2gray(READ_IMAGE); % Convert matrix to intensity image
                                % (map between 0 and 1)

BIN_IMAGE=im2bw(INST_IMAGE); % Convert image to binary image by
                                % thresholding map to either 0 or 1

U1=double(BIN_IMAGE);    % Convert to double precision
```

Figure 3.3: Image Initialization

3.4.2 Changing pixel value

For this part, the **if** and **else** command is used for loop control. In this part of the coding, it checks one by one row and column for the pixel. If the pixel is '1', it remains the same but if the pixel is '0', it is changed to '-1' according to Cellular Neural Network rule for binary images.

Nested **if** statements must each be paired with a matching **end**. The general form of statement using if, else and else if;

if expression1
  statements1
elseif expression2
    statements2
else
    statements3
end

```
[ROWS COLUMNS]=size(U1); % Size of array, returns the two-element
                         % row vector

 for row=1:ROWS;
     for col=1:COLUMNS;
         TEMP = U1(row,col);
         if(TEMP == 1);
             U(row,col)= 1;
         elseif(TEMP == 0);
             U(row,col)=  -1;   % Local Rule
         else
             U(row,col)=  U1(row,col);
         end
     end
 end
```

Figure 3.4: Changing pixel value

3.4.3: Initialization of Output Matrix

This step initializes the output matrix Y which is later calculated using the output from state equation. The output equation is;

$$yij = \frac{1}{2}|xij+1| - \frac{1}{2}|xij-1|$$

where

```
%=================================================================
%****************COMPUTATION OF INITIAL OUTOUT MATRIX*************
%*****************************************************************

for i=1:M+2;
    for j=1:N+2;
        Y(i,j)=0.5*abs(X(i,j)+1)-0.5*abs(X(i,j)-1); %Output Equation
    end
end
```

Figure 3.5: Initialization of Output Matrix

3.4.4: Computation of State Equation

The step 1 is to compute the sphere of influence radius. Here the size A is the size of the feedback synaptic. In this part, we can either write it as size (A) or size (B) as B is the input synaptic.

The step 2 is the summation of all terms, which is the equation of state equation. The coding here is linked to a function file, **normfun.m.** The function file will be explained later in this chapter. According to the state equation;

$$\dot{x}ij = -xij + \sum_{C(k,l)\in Sr(i,j)} \mathrm{A}(i,j;k,l)ykl + \sum_{C(k,l)\in Sr(i,j)} \mathrm{B}(i,j;k,l)ukl + \mathrm{Z}ij$$

As can be seen above, the state equation is actually the summation of state term, feedback synaptic, input synaptic and also the threshold.

Step 3 actually solves the state equation by using the **dsolve** command. This command is used because the state equation is a differential equation. The example syntax;

r = dsolve('equation, 'condition','v')

Note that it solves ordinary differential equation using **v** as the independent variable or in this case, the initial condition. The initial condition is given for each template.

Step 4 is the calculation of output equation after the state equation is solved. In this part, the result from state equation, **X2** is used to calculate the output equation.

```
%=================================================================
%***************COMPUTATION OF STATE EQUATION*********************
%*****************************************************************

%-STEP1--------Computation of the radius of sphere of influence----

[r1,c1] = size(A);
r = fix(r1/2);

for i=1+r:M+r;
    for j=1+r:N+r;

%-STEP2---------Computation of the ALL SUMMATION TERM--------------

        S  = normfun(r,i,j,A,B,Y,U);

        X1 = X(i,j); % STATE TERM
        F1 = S(1,1); % FEEDBACK TERM
        I1 = S(1,2); % INPUT TERM
        Z1 = z;      % THRESOLD TERM


%-STEP3---------Computation of the STATE EQUATION ----------------


% Solution of the given state equation by using dsolve function is
% computed as "X2 = dsolve('DX = -X + F1 +I1 - Z1', 'X(0)= 0')"
% which gives X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1)as output

        t = 20;


        X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1); % STATE Equation


%-STEP4---------UPDATION of the OUTPUT EQUATION-------------------

        Y(i,j)=0.5*abs(X2+1)-0.5*abs(X2-1); % OUTPUT Equation

    end
end
```

Figure 3.6: Computation of State Equation

3.4.5 Display result

This part is display the result of the output equation. Since the output is an image, the command **imshow** is used.

```
%=================================================================
%*************************DISPLAY RESULTS*************************
%*****************************************************************

imshow(Y);% displays image Y

%=================================================================
%****************************THE END******************************
%*****************************************************************
```

Figure 3.7: Display result

3.4.7 Calculation of Feedback Term and Input Term in Function File

This file is a function file, **normfun.m** which is linked to the main file. Only one function file is used for all the templates since function file consist of only the feedback synaptic and input synaptic which are the same for all templates. In this part of coding, the feedback synaptic and input synaptic is calculated. Since $|k-i| \leq r$, so k = i –r and i + r. Same goes for $|l-j| \leq r$, l = j-r, j + r according to mathematics.

S = [FEED_TERM INPT_TERM] links back to Step 2 in Figure 3.6 where S(1,1) indicates the feedback synaptic and S(1,2) indicates the input synaptic.

```
function S = normfun(r,i,j,A,B,Y,U);

 FEED_TERM = 0;
 for k=i-r:i+r;
    for l=j-r:j+r;
       FEED_TERM = FEED_TERM + A(k-i+2,l-j+2)*Y(k,l);% FEEDBACK TERM
    end
 end

 INPT_TERM = 0;
 for k=i-r:i+r;
    for l=j-r:j+r;
       INPT_TERM = INPT_TERM+ B(k-i+2,l-j+2)*U(k,l);  % INPUT TERM
    end
 end


 S = [FEED_TERM INPT_TERM];
```

Figure 3.8: Function File

3.5 Step by step image segmentation procedure in MATLAB

Step 1: Image initialization

After the algorithm is successfully designed, the next step is analysis. This part of analysis includes the image segmentation and pixel information. In the image segmentation, the image is divided by 4 X 4. This step is crucial to determine the coordinate of the object in each image.

Step 1 is the image initialization where the image is read and changed back into binary. As earlier we have change the pixel values of the image in the template coding, here we have to change back to the original pixel value so that it is easier for further image processing.

The command **edge** specifies the Prewitt method. This command is used to find the edges of the object in the images. The example syntax;

BW = edge(I,'prewitt')

Note that the Prewitt method finds edges using the Prewitt approximation to the derivative. It returns edges at those points where the gradient of I is maximum.

The command **strel** divide the image horizontally and vertically. The example syntax;

se2 = strel('line',10,45)

Note that the 'line ' means straight line, '10' indicates the amount of lines and '45' indicates the degree on the line.

Next is to dilate the object using **imdilate** command where it dilates the grayscale and binary . The command **imdilate** is normally followed by the **strel** command. The example syntax;

IM2 = imdilate(IM, SE)

Note that the **IM** is the image and **SE** is returned by strel function.

The last part of step 1 is to fill the object in the images by using the **imfill** command. The example syntax is;

BW2 = imfill(BW,'holes')

Note that this command fills the holes in the binary image BW.

```
%==========================Initialization==========================

clear all;
clc;

%==================================================================

I   = imread('r55.jpg');  %read the image
I1  = im2bw(I);          %convert image to binary image
I2  = double(I1);         %convert to double precision
BW1 = edge(I2,'prewitt'); %find edges in intensity image

se90 = strel('line', 3, 90);
se0  = strel('line', 3, 0);

I3  = imdilate(BW1, [se90 se0]);

I4  = imfill(I3, 'holes');

imshow(I4); %displays the intensity image

%==================================================================
```

Figure 3.9: Step 1 of Image Segmentation

Step 2: Setting the rows and columns

Step 2 divides the row and column. In this project, since we are doing image segmentation 4 X 4, the rows and columns are divided by 4.

```
[ROWS COLUMNS]=size(I4);% Size of array

R1 = ((1*ROWS)/4);
C1 = ((1*COLUMNS)/4);
R2 = ((2*ROWS)/4);
C2 = ((2*COLUMNS)/4);
R3 = ((3*ROWS)/4);
C3 = ((3*COLUMNS)/4);
R4 = ((4*ROWS)/4);
C4 = ((4*COLUMNS)/4);
```

Figure 3.10: Step 2 of Image Segmentation

Step 3: Detect Object and Display Coordinate

Step 3 detects the object in the image. The concept here is to detect the white pixel or '1' pixel because the object is white in colour while the background of the image is black in colour. This coding checks the pixel row by row and column by column. Since we have divided the image 4 X 4, we could generate 16 coordinates starting from RICI until R4C4.

The coding is same for all the coordinates, with the difference of the coordinate itself. For example, to run the detection for row 1 column 1, RICI, the coding has to be written;

```
for R=1:R1;
      for C=1:C1;
```

So for row 3 column 2, R3C2, the coding will be;

```
for R=R2:R3;
      for C=C1:C2;
```

Note that the limit for row and column has changed. The same goes for all the 16 coordinates.

The **display** command will display the coordinate of the object if it is detected in the particular coordinate.

```
%============================R1C1================================

DONE = 0;    % initialize the value of DONE
for R=1:R1;  % for all rows one by one
    for C=1:C1;     % for all columns one by one
        PIXEL = I4(R,C); % select the pixel value of either one or
                         % zero at each location of row and column
        if(PIXEL == 1 && DONE == 0);  % compare the pixel value with
                                      % 1, to detect the presence
                                      % the ball
            display('PIXEL:R1C1');  % to display the result
            DONE = 1;  % DONE is set to exit the loop
        end
    end
end
```

Figure 3.11: Step 3 of Image Segmentation

Step 4: Calculate and Display White Pixel

Step 4 is to calculate the amount of white pixel in each coordinate. The concept is nearly the same as image segmentation where it checks row by row and column by column. The **COUNT** coding will count only the white pixel and using command below to display the amount of white pixel;

```
disp(sprintf('%d total white pixel',COUNT));
```

```
%==============================R1C1==============================

 COUNT = 0;      % initialize the value of COUNT
 for R=1:R1;     % for all rows one by one
     for C=1:C1;  % for all columns one by one
         PIXEL = I4(R,C); % select the pixel value at row or column
         if(PIXEL == 1);  % to detect the presence of white pixel
             COUNT = COUNT + 1; % to calculate the white pixel
         end
     end
 end

 disp(sprintf('%d total white pixel',COUNT)); % to display the amount
                                               % of white pixel
```

Figure 3.12: Step 4 of Image Segmentation

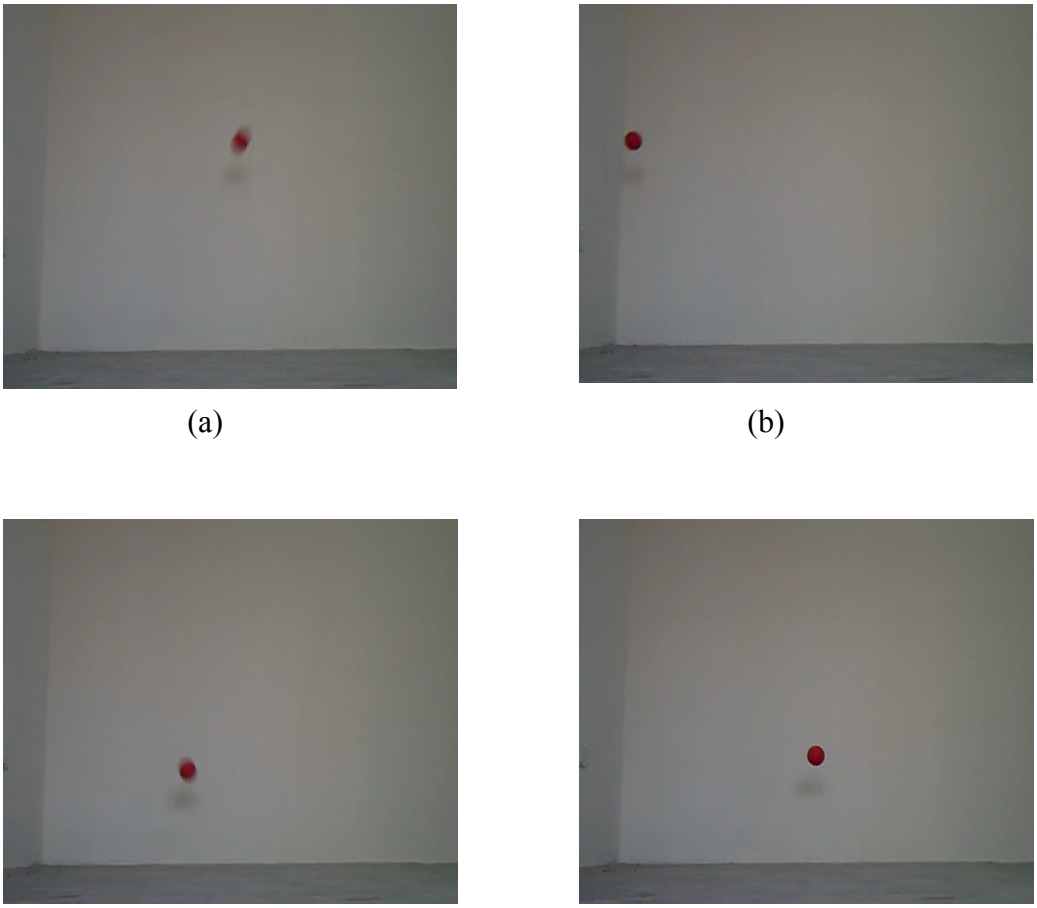
CHAPTER IV

RESULT AND DISCUSSION

4.1 Discussion and Analysis

Both the objectives and scopes in this project have been successfully achieved. This chapter discusses all the results obtained in this project which includes the video segmentation, result of image editing, and result from the templates, the designed algorithm, the object's coordinates and also the pixel information.

4.2 Video Clip Segmentation



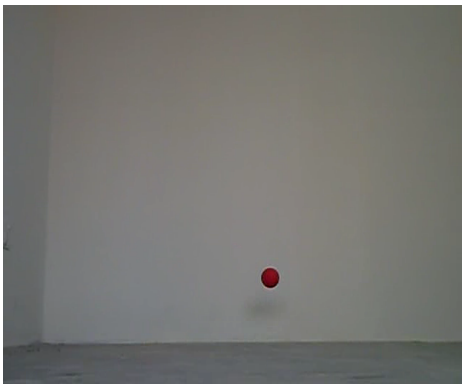
(a) (b)

(c) (d)

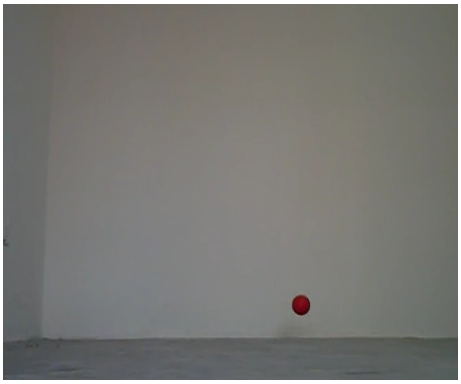
Figure 4.1: Video segmentation images

a) Frame 11 b) Frame 21

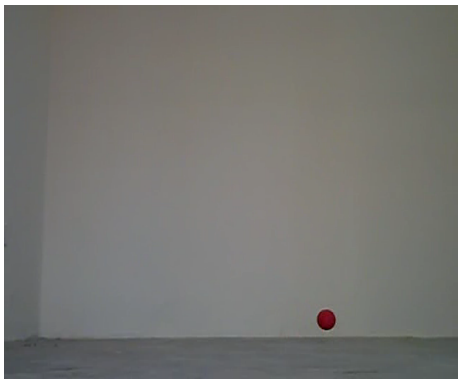
c) Frame 31 d) Frame 41



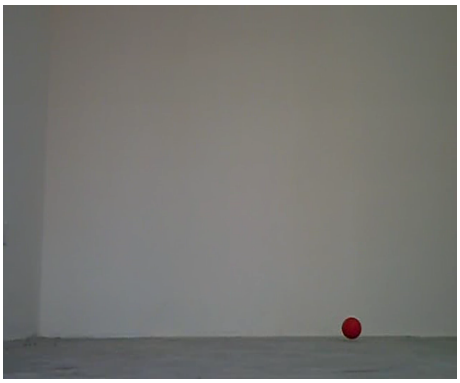
(e)



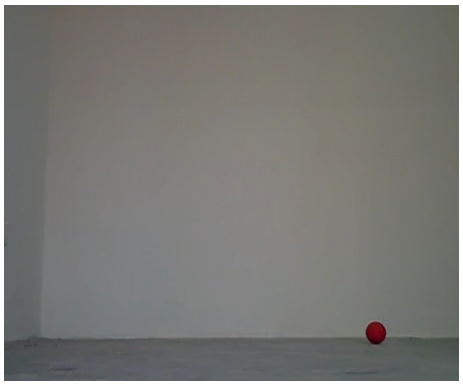
(f)



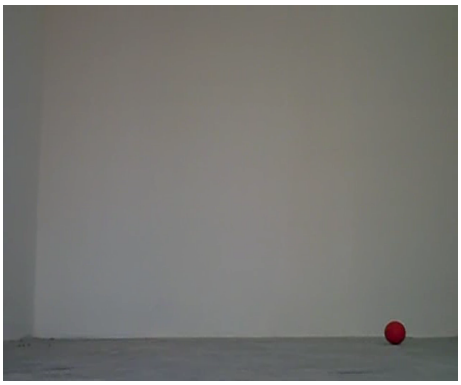
(g)



(h)



(i)



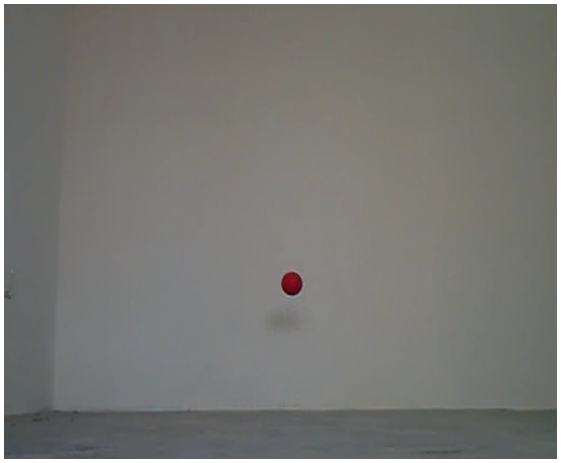
(j)

Figure 4.2: Video Image Segmentation
e) Frame 51 f) Frame 61 g) Frame 71
h) Frame 81 i) Frame 91 j) Frame 101

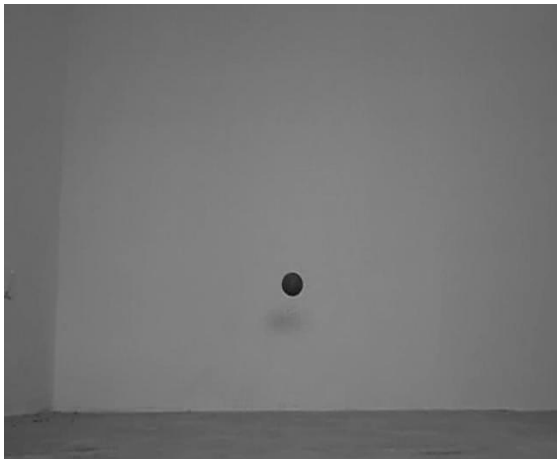
4.3 Conversion of Image Types

4.3.1 RGB to Grayscale

```
I = imread ('frame41.jpeg');
J = rgb2gray (I);
figure, imshow (I), figure, imshow (J);
```



Truecolor image



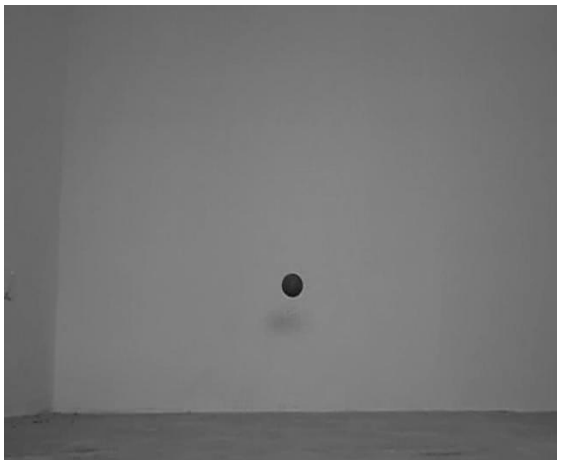
Grayscale image

Figure 4.3: RGB to Grayscale conversion result

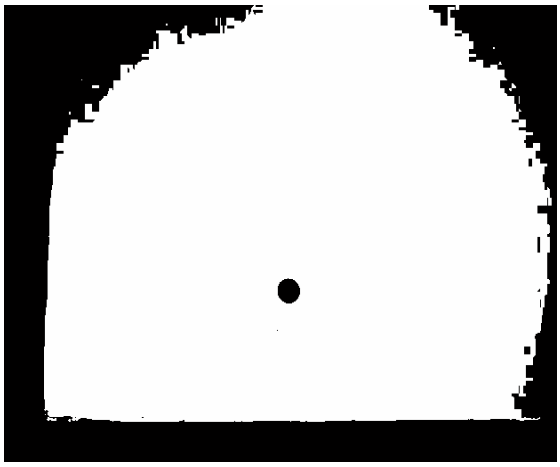
Comment: The output image shows a grayscale version of the truecolor image. As can been seen above, the output image is exactly the same as the input image with the difference of colour only.

4.3.2 Grayscale to Binary

```
I = imread ('framegray.jpeg');

level = graythresh (I);

BW = im2bw (I, level);

imshow (BW)
```



Grayscale image



Binary image

Figure 4.4: Grayscale to Binary conversion result

Comment: The conversion of the image from grayscale to binary shows that the output binary image is not 100% exact to the input grayscale image. This is because other than the object, the image shows the difference of light in the background. As can be seen in the grayscale image, the lower part of the image is more bright compared to the upper part of the image.

4.4 Result of Templates

There are five templates analyzed in this project before creating the algorithm. The algorithm was created based on the result of each template.

4.4.1 Edge Detection Template

The images below are the input and also the result of edge detection template simulated by MATLAB.

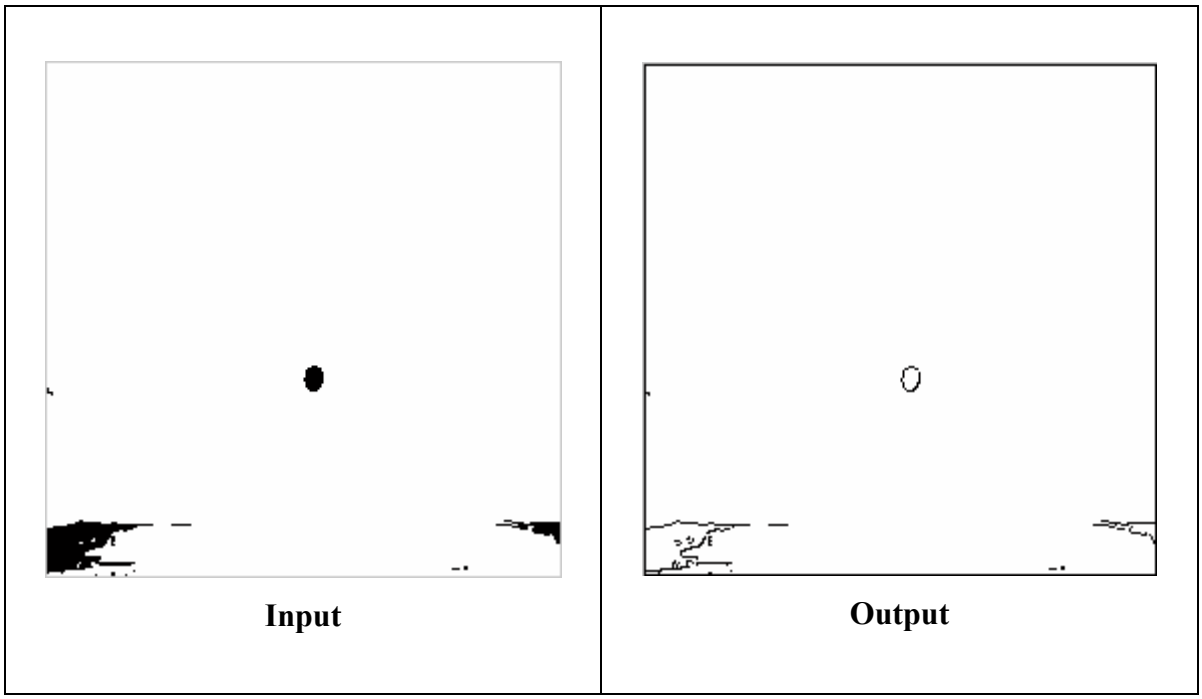
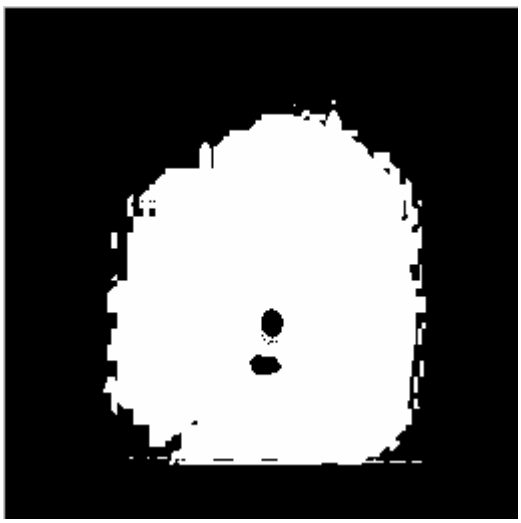


Figure 4.5: Result of Edge Detection Template

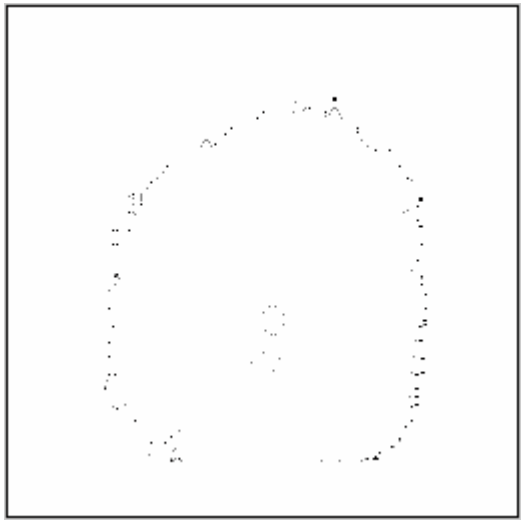
Comment: The result of this template shows quite precised edge detection, as the object has been clearly detected. But then, there is also some minor side effects as the template also detected the difference of light in the image.

4.4.2 Convex Corner Detection Template

The images below are the input and the result of corner detection template simulated by MATLAB



Input



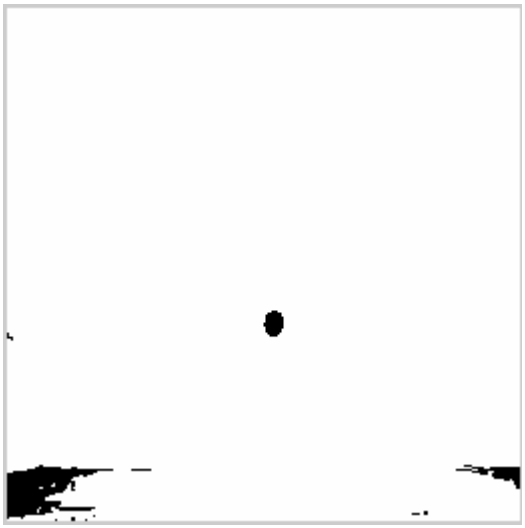
Output

Figure 4.6: Result of Convex Corner Template

Comment: The result of this template shows the corner detection in this image. As can be seen, there are two circles in the output image, the smaller circle being the object and the bigger circle being the effect of light. However, this template is not suitable as because in this project it is supposed to detect only the object and not it's surrounding as well.

4.4.3 Logic Not Template

The images below are the input and the result of logic not template simulated by MATLAB.



Input

Output

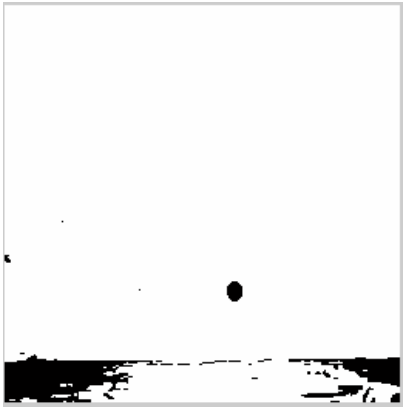
Figure 4.7: Result of Logic Not Template

Comment: This template shows the output image in black colour and the object in white. It shows very clearly the object in the output image but like the other templates outputs, it also has some minor effects due to the effects of light in the original images.

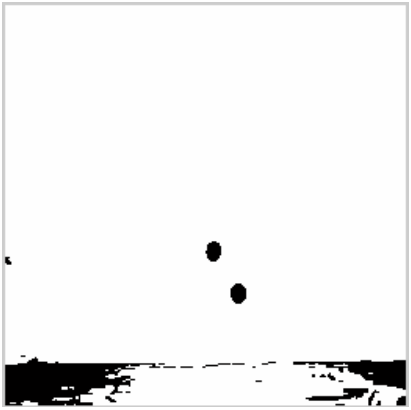
4.4.4 Logic OR Template

The images below are the inputs and the result of Logic OR template simulated by MATLAB.

Input 1



Input 2



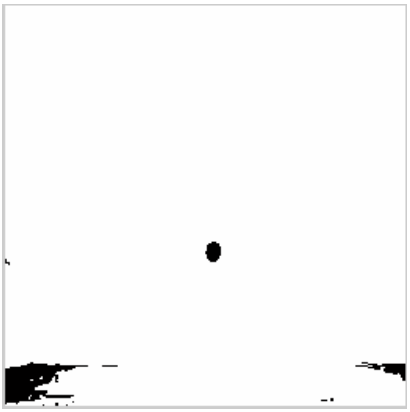
Output

Figure 4.8: Result of Logic OR Template

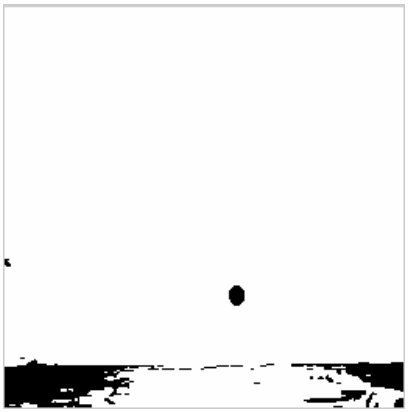
Comment: The template combined both objects in each images. It clearly shows both the object but like all other templates, it also shows the light of surrounding in the output.

4.4.5 Logic AND Template

The images below are the inputs and the result of Logic AND template simulated by MATLAB.



Input 1



Input 2

Output

Figure 4.9: Result of Logic AND Template

Comment: This template will only combine contents that overlap each other in both images. Since the objects have different position in both images, the output does not show the object but only the light effects, hence it is not suitable for this project.

4.5 Algorithm

After much research, the best combination of templates is shown below.

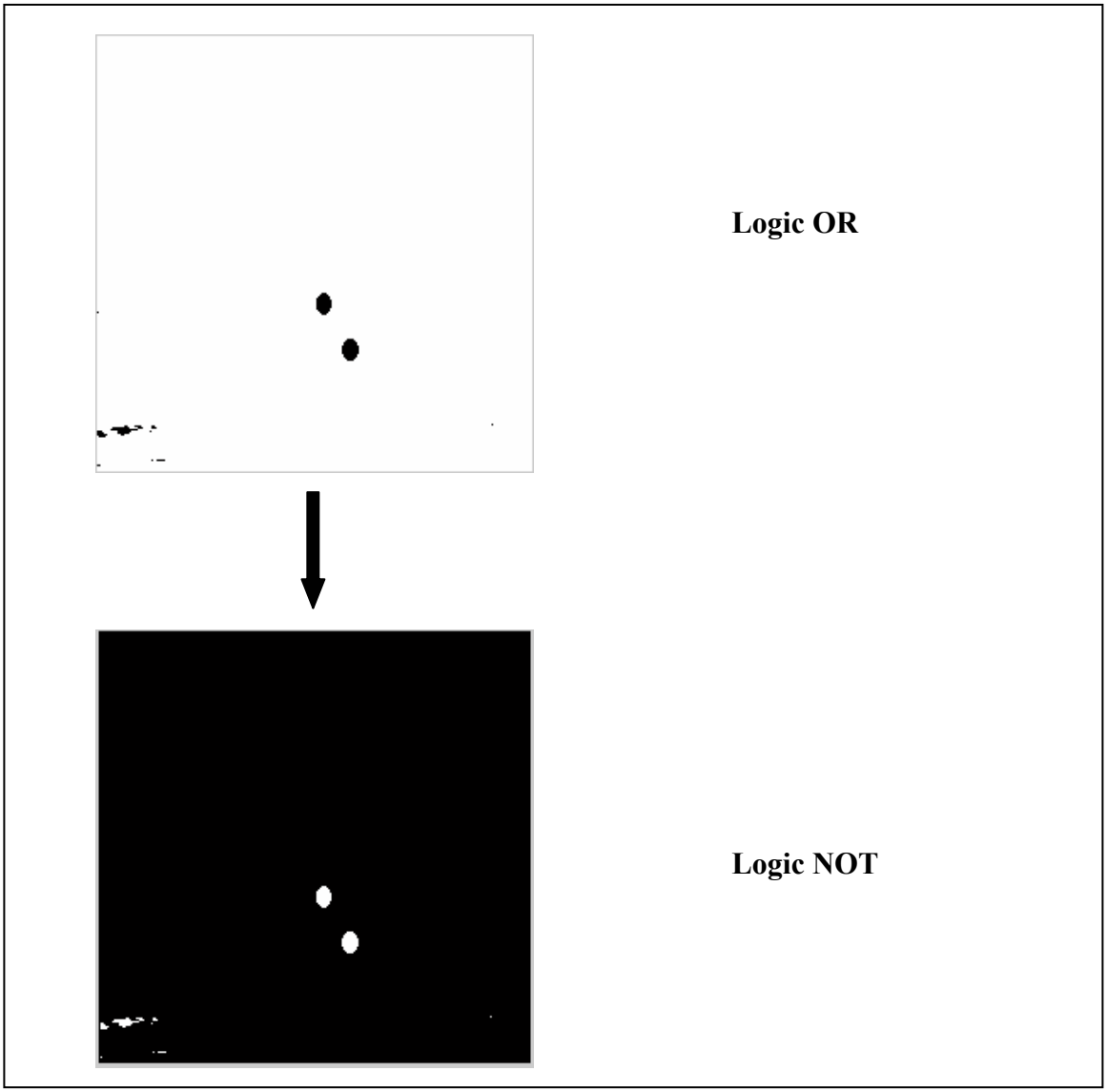


Figure 4.10: Algorithm

Comment: The output image of the algorithm shows the objects in white colour with a black background. Some very minor light effects can be seen as well.

4.6 Result of Algorithm

In the Logic OR template, two images are combined, for example, frame 11 and frame 21 are combined to produce only one output image. The next step is to use the result from Logic OR as the input for Logic NOT. The diagram below shows all the images from frame 11 until frame 101 as the result of the algorithm.

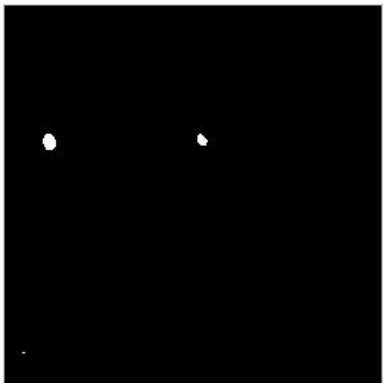


Image 1: Frame 11 &21

Image 2: Frame 31 & 41

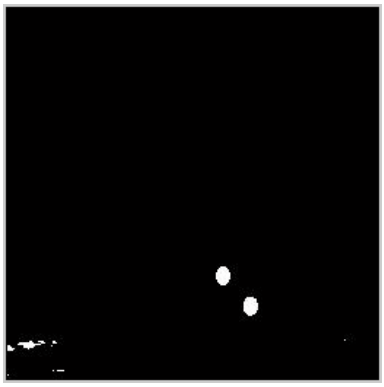


Image 3: Frame 51 & 61

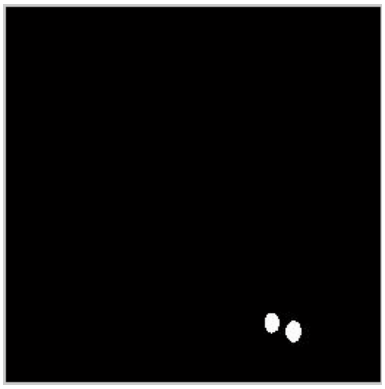


Image 4: Frame 71 & 81

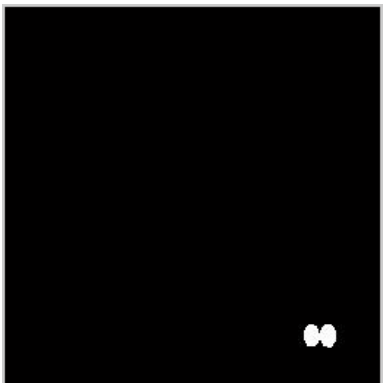


Image 5: Frame 91 & 101

Figure 4.11: Results of Algorithm

4.7 Object's Coordinates

The table below show the object's coordinate in each on the images. The 1st second represent the frame 11, the 2nd second represent frame 21 and so on. The table shows the object's previous position and current position according to the time.

TIME	PREVIOUS POSITION	CURRENT POSITION
1 second	-	R2C3
2 second	R2C3	R2C1
3 second	R2C1	R3C2
4 second	R3C3	R3C3
5 second	R3C3	R3C3
6 second	R3C3	R4C3
7 second	R4C3	R4C3
8 second	R4C3	R4C4
9 second	R4C4	R4C4
10 second	R4C4	R4C4

Table 4.1: Object's Coordinates

4.8 Pixel Information

The table below shows the white pixel information at each of the coordinate. The coding has been done to detect white pixel in the coordinate. Hence, if the object is in a particular coordinate, it displays the amount of white pixel as the object is in white colour and display ‘0’ if no object. The minor light effect in the algorithm output images have to be eliminated so that the coding will only detect the object and not the background as well.

	IMAGE 1	**IMGAE 2**	**IMAGE 3**	**IMAGE 4**	**IMAGE 5**
R1C1	0	0	0	0	0
R1C2	0	0	0	0	0
R1C3	0	0	0	0	0
R1C4	0	0	0	0	0
R2C1	0	0	0	0	0
R2C2	0	0	0	0	0
R2C3	**79**	0	0	0	0
R2C4	0	0	0	0	0
R3C1	0	0	0	0	0
R3C2	0	**129**	0	0	0
R3C3	**137**	**149**	**161**	0	0
R3C4	0	0	0	0	0
R4C1	0	0	0	0	0
R4C2	0	0	0	0	0
R4C3	0	0	**180**	**200**	0
R4C4	0	0	0	**187**	**402**

Table 4.2: Pixel Information

Comment: The pixel information table shows all the pixel reading of 5 images. As can be seen, for Image 1, the white pixel is read at coordinates R2C3 and R3C3. For Image 2, the coordinates are at R2C2 and R3C3 while for Image 3 the objects are at R3C3 and R4C3. In Image 4, the objects or balls are found at coordinates R4C3 and R4C4. Finally in Image 5, the coordinate is at R4C4. Image 5 only gives one coordinate because both the balls are too near to each other thus, due to the effect of image segmentation, only one coordinate is read. Besides that, we can also see that in Image 5, the value of white pixel, 402 is the highest compared to other positions. This is because that particular coordinate reads the white pixel of two balls instead of one ball like all the other coordinates.

CHAPTER V

CONCLUSION AND FUTURE DEVELOPMENT

5.1 Conclusion

This project consists of two parts; developing the Cellular Neural Network templates and the analysis of the output image from the algorithm. This project focuses on the software or the programming of the templates using MATLAB as the programming platform.

Generally, this project has achieved all its objectives and scopes but the efficiency of this project is about 85 %. This is because original images were not used in the project, instead the images have been edited to smaller size, where it is not reasonable in real world application. The other reason is the analysis of the image was not 100% accurate as there were some problems in image segmentation.

The segmentation has been tailored for the set of images used this project, which means if another set of images were used, the image segmentation would have not given an accurate result. This is because when doing the images segmentation, we have to be careful not to divide or split the object into two. Each object supposed to have only one coordinate but sometimes two objects are so close that it end up being read as one coordinate or one object gives two coordinates because the object was split during segmentation.

5.2 Future Development

Since there is always a room for improvement in any matter, so does this project. Although we have managed to create a fully functional template, this project still has the capacity for further improvement.

For improvement, instead of using video segmentation images, we can directly use the video of moving object as the input for the templates. As real time image processing becoming very popular nowadays, I suggest than an interface between a computer and camera itself is made. The coding has to be improved as well if the video is used as input for the templates, where u_{kl} will be a function of time instead of being constant.

Besides that, the image segmentation can be improved also. A more detailed segmentation can be done in order to get a very precise coordinate for the objects in the images. Hence the analysis will be more perfect and accurate. Other than that, more samplings or frames of video segmentation images can be used for a more accurate result of moving object detection.

5.3 Costing and Commercialization

The cost for this project only includes a laptop or personal computer, a digital camera and related software like SC Video Developer and MATLAB. In terms of cash spent, it is only needed to buy the license for the software used.

In term of commercialization, current condition of this project is not yet suitable for market. It can be improved and updated for commercialization purposes. The target customer will be in security industries like bank, company or any building that needs security system.

REFERENCES

1. Leon O. Chua and Tamas Roska, (2000). Cellular Neural Network and Visual Computing: Foundation and applications. Berkeley-Budapest, May 2000.

2. Application of Cellular Neural Network in Healthcare
URL: http:www.openclinic/org/doc/int/neuratnetworks011
Access date: March 20, 2007

3. Image Processing Toolbox 5 User's Guide
URL: http://www.mathworks.com
Access date: March 30, 2007

4. Chapter 2: Cellular Paradigms Theory and Simulation
URL: www.worldscibooks.com
Access date: April 15, 2007

5. Cellular Neural Network
URL: http://lab.analogic/stzaki.hu/cnnintro/html
Access Date: September 17, 2007

6 Solving ODEs in MATLAB
URL: http://coweb.cc.gatech.edu/process/198
Access date: April 19, 2008

7. Mariofanna G. Milanova, Adel Elmaghraby, Stuart Rubin "Cellular Neural Networks for Segmentation of Image Sequence", University of Louisville, USA, 1999.

8. Shih Fu-Chang & Di Zhong ,“Moving Object Segmentation and Tracking Using Spatio” ISCAS'97, Hong Kong, June 9-12, 1997.

9. J. C. Choi, S.-W. Lee, and S.-D. Kim, “Spatio-temporal video segmentation using a joint similarity measure,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 279–286, Apr. 1997.

10. Osman N. Uan and Lokman Auyrman, “Moving Object Detection Using Delayed-Cellular Neural Network”, Istanbul University Electrical Engineering Department, 1997.

11. Eryanie Binti Kaimi, ‘Cellular Neural Network Algorithm For Car Plate Recognition”, Faculty of Electrical & Electronics Engineering, University Malaysia Pahang, November 2007.

APPENDIX A

Software programming for Edge Detection Template

```
%=====================================================================
%*********************PARAMETER DECLARATIONS*************************
%*********************************************************************

clear all;% Clear variables and functions from memory
clc;% Clear command window

%------------------------Constant Declarations------------------------

M=254; % No. of rows in the CNN structure
N=254; % No. of columns in the CNN structure

%------------------------CNN template parameters----------------------

A=[ 0  0  0;  0  0  0;  0  0  0]; % Feedback Operator
B=[-1 -1 -1; -1  8 -1; -1 -1 -1]; % Input Synaptic Operator
Z=[-1];                           % Threshold  Value
X=zeros(256,256);    %CNN template initial state parameters


%=====================================================================
%*********************READ AND PREPARE DATA**************************
%*********************************************************************

READ_IMAGE=imread('pic4.jpeg'); % Read image from graphics file and map
                                % between 0 and 255

INST_IMAGE=mat2gray(READ_IMAGE); % Convert matrix to intensity image
                                 % (map between 0 and 1)

BIN_IMAGE=im2bw(INST_IMAGE,0.4); % Convert image to binary image by
                                 % thresholding map to either 0 or 1

U1=double(BIN_IMAGE);    % Convert to double precision

[ROWS COLUMNS]=size(U1); % Size of array, returns the two-element row
                         % vector

for row=1:ROWS;
    for col=1:COLUMNS;
        TEMP = U1(row,col);
        if(TEMP == 1);
            U(row,col)= 1;
        elseif(TEMP == 0);
```

```
            U(row,col)=  -1;     % Local Rule
        else
            U(row,col)=  U1(row,col);
        end
    end
end


%=====================================================================
%*****************COMPUTATION OF INITIAL OUTOUT MATRIX*****************
%*********************************************************************

for i=1:M+2;
    for j=1:N+2;
        Y(i,j)=0.5*abs(X(i,j)+1)-0.5*abs(X(i,j)-1);% Output Equation
    end
end

%=====================================================================
%*****************COMPUTATION OF STATE EQUATION************************
%*********************************************************************

%-STEP1----------Computation of the radius of sphere of influence------

[r1,c1] = size(A);
r = fix(r1/2);

for i=1+r:M+r;
    for j=1+r:N+r;

%-STEP2-----------Computation of the ALL SUMMATION TERM----------------

        S  = normfun(r,i,j,A,B,Y,U);

        X1 = X(i,j); % STATE TERM
        F1 = S(1,1); % FEEDBACK TERM
        I1 = S(1,2); % INPUT TERM
        Z1 = Z;      % THRESOLD TERM


%-STEP3-----------Computation of the STATE EQUATION -------------------


% Solution of the given state equation by using dsolve function is
% computed as "X2 = dsolve('DX = -X + F1 +I1 - Z1', 'X(0)= 0')"
% which gives X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1)as output

        t = 20;


        X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1); % STATE Equation
```

```
%-STEP4-----------UPDATION of the OUTPUT EQUATION---------------------

        Y(i,j)=0.5*abs(X2+1)-0.5*abs(X2-1); % OUTPUT Equation

    end
end

%=========================================================================
%*************************DISPLAY RESULTS*****************************
%*************************************************************************

imshow(Y); % displays image Y

%=========================================================================
%****************************THE END**********************************
%*************************************************************************
```

APPENDIX B

Software programming for Convex Corner Detection Template

```
%=====================================================================
%*********************PARAMETER DECLARATIONS*************************
%*********************************************************************

clear all;% Clear variables and functions from memory
clc;% Clear command window

%------------------------Constant Declarations------------------------

M=254; % No. of rows in the CNN structure
N=254; % No. of columns in the CNN structure

%------------------------CNN template parameters----------------------

A=[ 0  0  0;  0  2  0;  0  0  0]; % Feedback Operator
B=[-1 -1 -1; -1  8 -1; -1 -1 -1]; % Input Synaptic Operator
Z=[-8.5];                         % Threshold  Value
X=zeros(256,256);    %CNN template initial state parameters


%=====================================================================
%*********************READ AND PREPARE DATA**************************
%*********************************************************************

READ_IMAGE=imread('pic4.jpeg'); % Read image from graphics file and map
                                % between 0 and 255

INST_IMAGE=mat2gray(READ_IMAGE); % Convert matrix to intensity image
                                 % (map between 0 and 1)

BIN_IMAGE=im2bw(INST_IMAGE,0.75); % Convert image to binary image by
                                  % thresholding map to either 0 or 1

U1=double(BIN_IMAGE);    % Convert to double precision

[ROWS COLUMNS]=size(U1); % Size of array, returns the two-element row
                         % vector

for row=1:ROWS;
    for col=1:COLUMNS;
        TEMP = U1(row,col);
        if(TEMP == 1);
            U(row,col)= 1;
        elseif(TEMP == 0);
```

```
         U(row,col)=  -1;   % Local Rule
        else
            U(row,col)=  U1(row,col);
        end
    end
end


%=====================================================================
%*****************COMPUTATION OF INITIAL OUTOUT MATRIX*****************
%*********************************************************************

for i=1:M+2;
    for j=1:N+2;
        Y(i,j)=0.5*abs(X(i,j)+1)-0.5*abs(X(i,j)-1);% Output Equation
    end
end

%=====================================================================
%*****************COMPUTATION OF STATE EQUATION************************
%*********************************************************************

%-STEP1----------Computation of the radius of sphere of influence------

[r1,c1] = size(A);
r = fix(r1/2);

for i=1+r:M+r;
    for j=1+r:N+r;

%-STEP2-----------Computation of the ALL SUMMATION TERM----------------

        S  = normfun(r,i,j,A,B,Y,U);

        X1 = X(i,j); % STATE TERM
        F1 = S(1,1); % FEEDBACK TERM
        I1 = S(1,2); % INPUT TERM
        Z1 = Z;      % THRESOLD TERM


%-STEP3-----------Computation of the STATE EQUATION -------------------


% Solution of the given state equation by using dsolve function is
% computed as "X2 = dsolve('DX = -X + F1 +I1 - Z1', 'X(0)= 0')"
% which gives X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1)as output

        t = 20;


      X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1); % STATE Equation
```

```
%-STEP4-----------UPDATION of the OUTPUT EQUATION---------------------

        Y(i,j)=0.5*abs(X2+1)-0.5*abs(X2-1); % OUTPUT Equation

    end
end

%=========================================================================
%*************************DISPLAY RESULTS*****************************
%*************************************************************************

imshow(Y); % displays image Y

%=========================================================================
%****************************THE END**********************************
%*************************************************************************
```

APPENDIX C

Software programming for Logic NOT Template

```
%=====================================================================
%*********************PARAMETER DECLARATIONS*************************
%*********************************************************************

clear all;% Clear variables and functions from memory
clc;% Clear command window

%------------------------Constant Declarations------------------------

M=254; % No. of rows in the CNN structure
N=254; % No. of columns in the CNN structure

%--------------------------CNN template parameters-------------------

A=[0 0 0; 0  1 0; 0 0 0]; % Feedback Operator
B=[0 0 0; 0 -2 0; 0 0 0]; % Input Synaptic Operator
Z=[0];                    % Threshold  Value
X=zeros(256,256);  %CNN template initial state parameters


%=====================================================================
%*********************READ AND PREPARE DATA**************************
%*********************************************************************

READ_IMAGE=imread('pic4.jpeg'); % Read image from graphics file and map
                                % between 0 and 255

INST_IMAGE=mat2gray(READ_IMAGE); % Convert matrix to intensity image
                                 % (map between 0 and 1)

BIN_IMAGE=im2bw(INST_IMAGE); % Convert image to binary image by
                                 % thresholding map to either 0 or 1

U1=double(BIN_IMAGE);    % Convert to double precision

[ROWS COLUMNS]=size(U1); % Size of array, returns the two-element row
                         % vector

for row=1:ROWS;
    for col=1:COLUMNS;
        TEMP = U1(row,col);
        if(TEMP == 1);
            U(row,col)= 1;
        elseif(TEMP == 0);
```

```
            U(row,col)=  -1;   % Local Rule
        else
            U(row,col)=  U1(row,col);
        end
    end
end


%=====================================================================
%*****************COMPUTATION OF INITIAL OUTOUT MATRIX*****************
%*********************************************************************

for i=1:M+2;
    for j=1:N+2;
        Y(i,j)=0.5*abs(X(i,j)+1)-0.5*abs(X(i,j)-1);% Output Equation
    end
end

%=====================================================================
%*****************COMPUTATION OF STATE EQUATION************************
%*********************************************************************

%-STEP1----------Computation of the radius of sphere of influence------

[r1,c1] = size(A);
r = fix(r1/2);

for i=1+r:M+r;
    for j=1+r:N+r;

%-STEP2-----------Computation of the ALL SUMMATION TERM----------------

        S  = normfun(r,i,j,A,B,Y,U);

        X1 = X(i,j); % STATE TERM
        F1 = S(1,1); % FEEDBACK TERM
        I1 = S(1,2); % INPUT TERM
        Z1 = Z;      % THRESOLD TERM


%-STEP3-----------Computation of the STATE EQUATION -------------------


% Solution of the given state equation by using dsolve function is
% computed as "X2 = dsolve('DX = -X + F1 +I1 - Z1', 'X(0)= 0')"
% which gives X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1)as output

        t = 20;

        X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1); % STATE Equation
```

```
%-STEP4-----------UPDATION of the OUTPUT EQUATION---------------------

        Y(i,j)=0.5*abs(X2+1)-0.5*abs(X2-1); % OUTPUT Equation

    end
end

%=========================================================================
%*************************DISPLAY RESULTS*****************************
%*************************************************************************

figure(1):imshow(U1);% displays image U1
figure(2):imshow(Y);% displays image Y

%=========================================================================
%****************************THE END**********************************
%*************************************************************************
```

APPENDIX D

Software programming for Logic OR Template

```
%=====================================================================
%*********************PARAMETER DECLARATIONS*************************
%*********************************************************************

clear all;% Clear variables and functions from memory
clc;% Clear command window

%------------------------Constant Declarations------------------------

M=254; % No. of rows in the CNN structure
N=254; % No. of columns in the CNN structure

%------------------CNN template parameters---------------------------

A=[0 0 0; 0 3 0; 0 0 0]; % Feedback Operator
B=[0 0 0; 0 3 0; 0 0 0]; % Input Synaptic Operator
Z=2;                     % Threshold  Value
X=zeros(256,256);  %CNN template initial state parameters

%=====================================================================
%*********************READ AND PREPARE DATA**************************
%*********************************************************************

%-------------------------read image1---------------------------------

READ_IMAGE1=imread('pic3.jpeg'); % Read image from graphics file and
                                 % map between 0 and 255

INST_IMAGE1=mat2gray(READ_IMAGE1); % Convert matrix to intensity image
                                   % (map between 0 and 1)

BIN_IMAGE1=im2bw(INST_IMAGE1,0.4); % Convert image to binary image by
                                   % thresholding map to either 0 or 1

U1=double(BIN_IMAGE1); % Convert to double precision



%-------------------------read image2---------------------------------

READ_IMAGE2=imread('pic4.jpeg'); % Read image from graphics file and
                                 % map between 0 and 255
```

```
INST_IMAGE2=mat2gray(READ_IMAGE2); % Convert matrix to intensity image
                                   % (map between 0 and 1)

BIN_IMAGE2=im2bw(INST_IMAGE2,0.4); %Convert image to binary image by
                                   % thresholding map to either 0 or 1

X1=double(BIN_IMAGE2);% Convert to double precision

[ROWS COLUMNS]=size(X1); % Size of array, returns the two-element row
                         % vector

for row=1:ROWS;
    for col=1:COLUMNS;
        if (X1(row,col)== 0 | U1(row,col)==0)
            X(row,col)=-1;
            U(row,col)=-1;
        else
            X(row,col)=1;
            U(row,col)=1;
        end
    end
end


%=====================================================================
%*****************COMPUTATION OF INITIAL OUTOUT MATRIX****************
%*********************************************************************

for i=1:M+2;
    for j=1:N+2;
        Y(i,j)=0.5*abs(X(i,j)+1)-0.5*abs(X(i,j)-1); % Output Equation
    end
end

%=====================================================================
%*****************COMPUTATION OF STATE EQUATION***********************
%*********************************************************************

%-STEP1----------Computation of the radius of sphere of influence------

[r1,c1] = size(A);
r = fix(r1/2);

for i=1+r:M+r;
    for j=1+r:N+r;
```

```
%-STEP2-----------Computation of the ALL SUMMATION TERM----------------

        S  = normfun(r,i,j,A,B,Y,U);

        X1 = X(i,j); % STATE TERM
        F1 = S(1,1); % FEEDBACK TERM
        I1 = S(1,2); % INPUT TERM
        Z1 = Z;      % THRESOLD TERM
%-STEP3-----------Computation of the STATE EQUATION -------------------

% Solution of the given state equation by using dsolve function is
% computed as "X2 = dsolve('DX = -X + F1 +I1 - Z1', 'X(0)= 0')"
% which gives X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1)as output

        t = 20;

        X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1); % STATE Equation


%-STEP4-----------UPDATION of the OUTPUT EQUATION---------------------

        Y(i,j)=0.5*abs(X2+1)-0.5*abs(X2-1); % OUTPUT Equation

    end
end

%=====================================================================
%*************************DISPLAY RESULTS******************************
%*********************************************************************

imshow(Y); % displays image Y

%=====================================================================
%****************************THE END**********************************
%*********************************************************************
```

APPENDIX E

Software programming for Logic AND Template

```
%=====================================================================
%*********************PARAMETER DECLARATIONS**************************
%*********************************************************************

clear all;% Clear variables and functions from memory
clc;% Clear command window

%------------------------Constant Declarations-------------------------

M=254; % No. of rows in the CNN structure
N=254; % No. of columns in the CNN structure

%------------------CNN template parameters----------------------------

A=[0 0 0; 0 1.5 0; 0 0 0]; % Feedback Operator
B=[0 0 0; 0 1.5 0; 0 0 0]; % Input Synaptic Operator
Z=-1.5;                     % Threshold  Value
X=zeros(256,256);  %CNN template initial state parameters

%=====================================================================
%*********************READ AND PREPARE DATA***************************
%*********************************************************************

%---------------------------read image1-------------------------------

READ_IMAGE1=imread('pic1.png'); % Read image from graphics file and map
                                % between 0 and 255

INST_IMAGE1=mat2gray(READ_IMAGE1); % Convert matrix to intensity image
                                   % (map between 0 and 1)

BIN_IMAGE1=im2bw(INST_IMAGE1); % Convert image to binary image by
                               % thresholding map to either 0 or 1

U0=double(BIN_IMAGE1);% Convert to double precision

%----------------------------read image2-------------------------------

READ_IMAGE2=imread('pic2.png'); % Read image from graphics file and map
                                % between 0 and 255
```

```
INST_IMAGE2=mat2gray(READ_IMAGE2); % Convert matrix to intensity image
                                   %(map between 0 and 1)

BIN_IMAGE2=im2bw(INST_IMAGE2); % Convert image to binary image by
                               % thresholding map to either 0 or 1

X1=double(BIN_IMAGE2); % Convert to double precision

[ROWS COLUMNS]=size(X1); % Size of array, returns the two-element row
                         % vector

for row=1:ROWS;
    for col=1:COLUMNS;
        if (X1(row,col)== 0 & U0(row,col)==0)
            X(row,col)=-1;
            U(row,col)=-1;
        else
            X(row,col)=1;
            U(row,col)=1;
        end
    end
end

%=====================================================================
%*****************COMPUTATION OF INITIAL OUTOUT MATRIX*****************
%*********************************************************************

for i=1:M+2;
    for j=1:N+2;
        Y(i,j)=0.5*abs(X(i,j)+1)-0.5*abs(X(i,j)-1); % Output Equation
    end
end

%=====================================================================
%*****************COMPUTATION OF STATE EQUATION************************
%*********************************************************************

%-STEP1----------Computation of the radius of sphere of influence------

[r1,c1] = size(A);
r = fix(r1/2);

for i=1+r:M+r;
    for j=1+r:N+r;
```

```
%-STEP2-----------Computation of the ALL SUMMATION TERM---------------

        S  = normfun(r,i,j,A,B,Y,U);

        X1 = X(i,j); % STATE TERM
        F1 = S(1,1); % FEEDBACK TERM
        I1 = S(1,2); % INPUT TERM
        Z1 = Z;      % THRESOLD TERM

%-STEP3-----------Computation of the STATE EQUATION -------------------

% Solution of the given state equation by using dsolve function is
% computed as "X2 = dsolve('DX = -X + F1 +I1 - Z1', 'X(0)= 0')"
% which gives X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1)as output

        t = 20;

        X2= F1+I1-Z1+exp(-t)*(-F1-I1+Z1); % STATE Equation


%-STEP4-----------UPDATION of the OUTPUT EQUATION----------------------

        Y(i,j)=0.5*abs(X2+1)-0.5*abs(X2-1); % OUTPUT Equation

    end
end

%=========================================================================
%*************************DISPLAY RESULTS*****************************
%*************************************************************************

imshow(Y);% displays image Y

%=========================================================================
%****************************THE END*********************************
%*************************************************************************
```

APPENDIX F

Software programming for Function File (All Templates)

```
function S = normfun(r,i,j,A,B,Y,U);

FEED_TERM = 0;
for k=i-r:i+r;
   for l=j-r:j+r;
      FEED_TERM = FEED_TERM + A(k-i+2,l-j+2)*Y(k,l);  % FEEDBACK TERM
   end
end

INPT_TERM = 0;
for k=i-r:i+r;
   for l=j-r:j+r;
      INPT_TERM = INPT_TERM+ B(k-i+2,l-j+2)*U(k,l);  % INPUT TERM
   end
end


S = [FEED_TERM INPT_TERM];
```

APPENDIX G

Software programming for image segmentation

```
%==========================Design Description==========================

% Implementation of Image Segmentation to detect the objects and to
% count the pixel. The entire image is divided into 4X4 parts
% vertically and horizontally.

%===========================Initialization============================

clear all;
clc;

%=====================================================================

I   = imread('r55.jpg');  %read the image
I1  = im2bw(I);          %convert image to binary image
I2  = double(I1);         %convert to double precision
BW1 = edge(I2,'prewitt'); %find edges in intensity image

se90 = strel('line', 3, 90);
se0  = strel('line', 3, 0);

I3  = imdilate(BW1, [se90 se0]);

I4  = imfill(I3, 'holes');

imshow(I4); %displays the intensity image

%=====================================================================

[ROWS COLUMNS]=size(I4);% Size of array

R1 = ((1*ROWS)/4);
C1 = ((1*COLUMNS)/4);
R2 = ((2*ROWS)/4);
C2 = ((2*COLUMNS)/4);
R3 = ((3*ROWS)/4);
C3 = ((3*COLUMNS)/4);
R4 = ((4*ROWS)/4);
C4 = ((4*COLUMNS)/4);
```

```
%=====================================R1C1==============================

DONE = 0;   % initialize the value of DONE
for R=1:R1;  % for all rows one by one

    for C=1:C1;    % for all columns one by one
        PIXEL = I4(R,C); % select the pixel value of either one or zero
                         % at each location of row and column
        if(PIXEL == 1 && DONE == 0);  % compare the pixel value with 1,
                                      % to detect the presence the ball
            display('PIXEL:R1C1');  % to display the result
            DONE = 1;  % DONE is set to exit the loop
        end
    end
end

COUNT = 0;        % initialize the value of COUNT
for R=1:R1;       % for all rows one by one
    for C=1:C1;   % for all columns one by one
        PIXEL = I4(R,C);      % select the pixel value at row or column
        if(PIXEL == 1);       % to detect the presence of white pixel
            COUNT = COUNT + 1;      % to calculate the white pixel
        end
    end
end

  disp(sprintf('%d total pixel',COUNT)); % to display the amount of
                                         % white pixel

%=====================================R1C2==============================

DONE = 0;
for R=1:R1;
    for C=C1:C2;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R1C2');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=1:R1;
    for C=C1:C2;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));
```

```
%=====================================R1C3=============================

DONE = 0;
for R=1:R1;
    for C=C2:C3;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R1C3');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=1:R1;
    for C=C2:C3;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));

%=====================================R1C4=============================

DONE = 0;
for R=1:R1;
    for C=C3:C4;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R1C4');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=1:R1;
    for C=C3:C4;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));
```

```
%=====================================R2C1==============================

DONE = 0;
for R=R1:R2;
    for C=1:C1;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R2C1');
            DONE = 1;
        end
    end
end
COUNT = 0;
for R=R1:R2;
    for C=1:C1;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));

%=====================================R2C2==============================

DONE = 0;
for R=R1:R2;
    for C=C1:C2;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R2C2');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R1:R2;
    for C=C1:C2;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));
```

```
%=====================================R2C3=============================

DONE = 0;
for R=R1:R2;
    for C=C2:C3;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R2C3');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R1:R2;
    for C=C2:C3;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));

%=====================================R2C4=============================

DONE = 0;
for R=R1:R2;
    for C=C3:C4;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R2C4');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R1:R2;
    for C=C3:C4;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));
```

```
%=====================================R3C1=============================

DONE = 0;
for R=R2:R3;
    for C=1:C1;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R3C1');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R2:R3;
    for C=1:C1;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));

%=====================================R3C2=============================

DONE = 0;
for R=R2:R3;
    for C=C1:C2;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R3C2');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R2:R3;
    for C=C1:C2;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));
```

```
%=====================================R3C3=============================

DONE = 0;
for R=R2:R3;
    for C=C2:C3;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R3C3');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R2:R3;
    for C=C2:C3;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));



%=====================================R3C4=============================

DONE = 0;
for R=R2:R3;
    for C=C3:C4;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R3C4');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R2:R3;
    for C=C3:C4;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));
```

```
%=====================================R4C1==============================

DONE = 0;
for R=R3:R4;
    for C=1:C1;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R4C1');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R3:R4;
    for C=1:C1;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));

%=====================================R4C2==============================

DONE = 0;
for R=R3:R4;
    for C=C1:C2;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R4C2');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R3:R4;
    for C=C1:C2;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));
```

```
%=================================R4C3============================

DONE = 0;
for R=R3:R4;
    for C=C2:C3;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R4C3');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R3:R4;
    for C=C2:C3;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));

%=================================R4C4============================

DONE = 0;
for R=R3:R4;
    for C=C3:C4;
        PIXEL = I4(R,C);
        if(PIXEL == 1 && DONE == 0);
            display('PIXEL:R4C4');
            DONE = 1;
        end
    end
end

COUNT = 0;
for R=R3:R4;
    for C=C3:C4;
        PIXEL = I4(R,C);
        if(PIXEL == 1);
            COUNT = COUNT + 1;
        end
    end
end

disp(sprintf('%d total pixel',COUNT));

%*********************************************************************
%=================================DONE============================
%*********************************************************************
```