IMPROVED EDGE BASED IMAGE STEGANOGRAPHIC TECHNIQUE

By

JHILIK GUHA (11700114034)

ORCHI SAHA (11700114043)

RIYA ROY (11700114051)

UNDER THE GUIDANCE OF

SOMENATH NAG CHOUDHURY

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

RCC INSTITUTE OF INFORMATION TECHNOLOGY

SESSION 2017-2018

श्रमम् बिना न किमपि साध्यम्

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RCC INSTITURTE OF INFORMATION TECHNOLOGY
[Affiliated to Maulana Abul Kalam Azad University of Technology]
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA-700015

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
RCC INSTITUTE OF INFORMATION TECHNOLOGY

TO WHOM IT MAY CONCERN

I hereby recommend that the Project entitled **DEVELOPMENT OF AN IMPROVED EDGE BASED IMAGE STEGANOGRAPHIC TECHNIQUE** prepared under my supervision by **Jhilik Guha (11700114034, CSE/2014/017), Orchi Saha (11700114043, CSE/2014/028), Riya Roy (11700114051, CSE/2014/001)** of B-Tech (7th Semester), may be accepted in partial fulfilment for the degree of **Bachelor of Technology in Computer Science & Engineering** under Maulana Abul Kalam Azad University of Technology (MAKAUT).

………………………………………………………..
Project Supervisor
Department of Computer Science and Engineering
RCC Institute of Information Technology

Countersigned:

………………………………………………………..
Head
Department of Computer Science & Engineering,
RCC Institute of Information Technology
Kolkata – 700015.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
RCC INSTITUTE OF INFORMATION TECHNOLOGY

श्रमम् बिना न किमपि साध्यम्

CERTIFICATE OF APPROVAL

The foregoing Project is hereby accepted as a credible study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve the project only for the purpose for which it is submitted.

FINAL EXAMINATION FOR
EVALUATION OF PROJECT

1. ______________________________

2. ______________________________

(Signature of Examiners)

ACKNOWLEDGEMENT

We would like to express our gratitude to our teacher Mr Somenath Nag Choudhury, for giving us a good guideline for our project. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in this project.

Our sincerest gratitude to many people, especially our classmates and team members itself, for making valuable comment suggestions which gave us an inspiration to improve our project. We thank all the people for their help directly and indirectly for this project.

ABSTRACT

This paper proposes a novel steganography technique, where edges in the cover image have been used to embed the messages. The amount of data to be embedded plays an important role on the selection of edges, i.e., more the amount of data to be embedded, larger the use of weaker edges for embedding. Experimental results have shown that the proposed technique performs better or at least at par with the state-of-the-art steganography techniques but provides higher embedding capacity.

Table of Contents

List of figures:

Chapter 1:

INTRODUCTION

STEGANOGRAPHY comes from Greek Words: STEGANOS – "Covered", GRAPHIE – "Writing". Generally the sender writes an innocuous message and conceals a secret message on the same piece of paper. The main goal of steganography is communicating securely in a completely undetectable manner and avoiding the draw of suspicion to the transmission of hidden data. It is not only to keep others from knowing the hidden information, but to keep others from thinking that the information even exists.

Steganography is an art of secure transmission of messages from sender to receiver. It should ensure that none can reliably conclude on the secret communication between the sender and the receiver. To achieve such secrecy, the message is hidden in some cover media which may not raise any suspicion on the possibility of carrying the secret message to the third party. The cover medium is distorted due to embedding. This distortion in visual and statistical properties of the cover medium may lead to steganographic detectability. The objective of any steganography technique is to preserve these properties while embedding the message in cover media.

Images are the preferred medium for the current steganography techniques. Content adaptability, visual resilience, and smaller size of images make them good carrier to transmit the secret messages over internet.

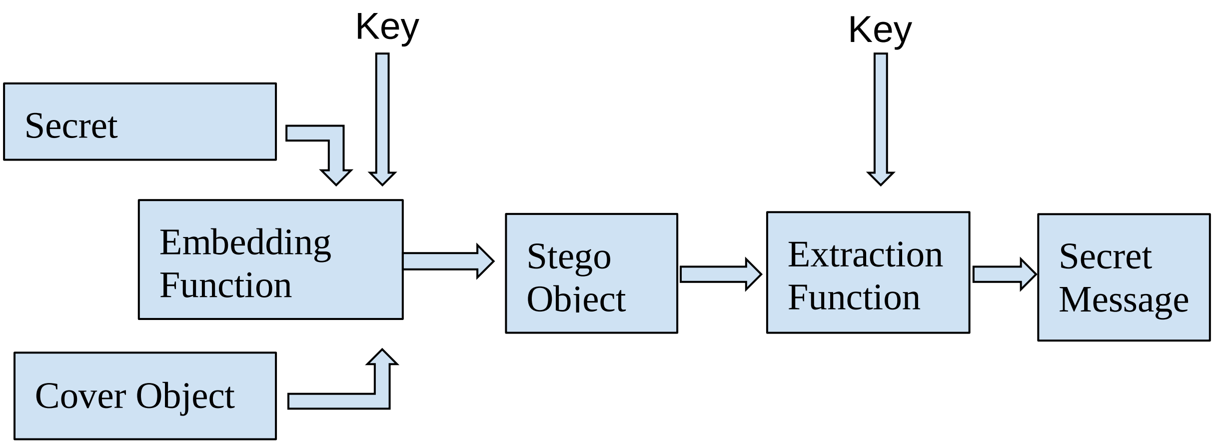


Fig1: A model of the steganography process with cryptography

There exist a large number of image steganography techniques which are accompanied by various attacks on the steganography systems. Security of any steganography technique is dependant on the selection of pixels for embedding. Noisy pixels and those in textured areas are better choices for embedding because they are difficult to model. Edge pixels can be seen as noisy pixel because their intensities are either higher or lower than their neighbouring pixels due to the sudden change in the coefficient gradient. Edges are difficult to model in comparison to pixels in smoother area due to these sharp changes in the visual and statistical properties. Therefore, edges make better option to hide secret data than any other region of an image where a small distortion is much more noticeable. The figure below is an image with 20% of pixels modified to produce distortion. An image has some smooth parts such as sky and some parts with high concentration of edges, such as trees and buildings. Some areas from both the smoother part and the high texture part are cropped and zoomed as shown. It can be seen that the modified pixels in the smoother parts are clearly noticeable, whereas the distortions in the high texture parts are harder to detect. In this paper, we have proposed a steganography technique which can hide secret messages only in the edges of the cover image. The proposed steganography technique is found to have an excellent security against steganalysis attacks.

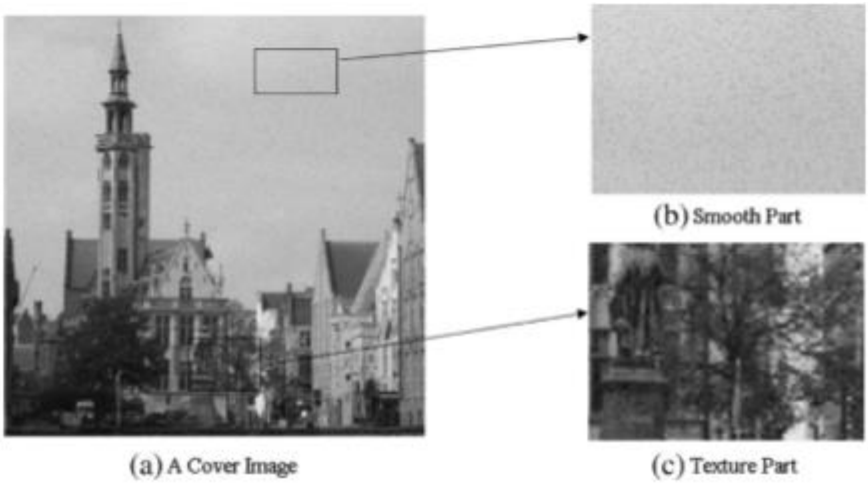


Fig2

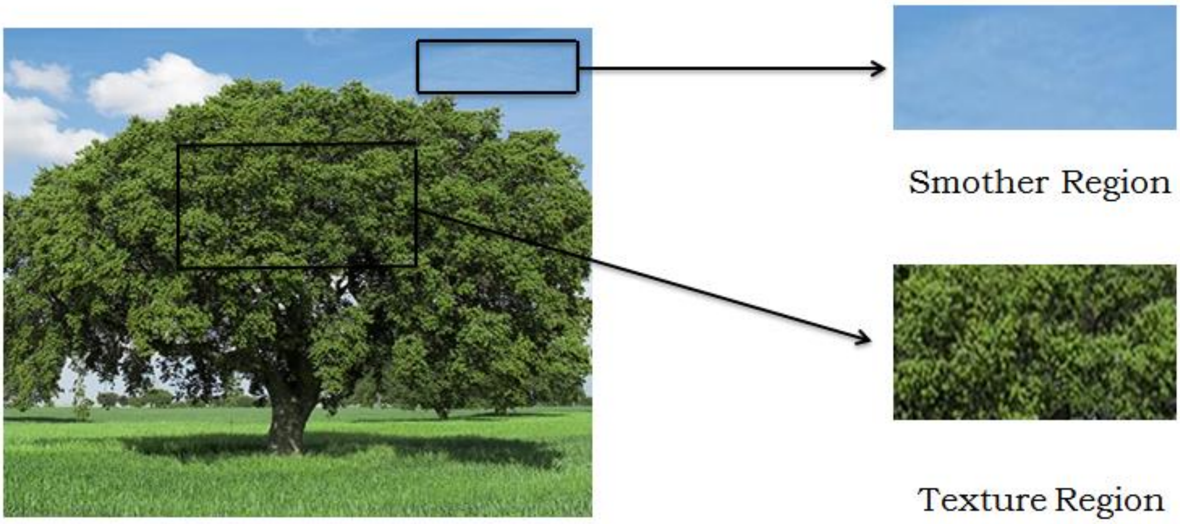


Fig3

Chapter 2:

REVIEW OF LITERATURE

There exists several steganographic techniques that can securely embed data in a carrier medium and tools to detect the presence of any secret messages in a steganogram reliably. Steganographic technique consists of embedding and extracting mechanisms. Image-based steganographic techniques is classified into two major categories: spatial domain and frequency (transform) domain.

A secret message is generally considered as encrypted data, where bits of the encrypted message are embedded in pixels of the cover image. The trivial steganographic technique is based on the least significant bit (LSB) substitution method in which the LSB of the pixels is modified to embed the secret message. In the spatial domain, theis type of technique can be broadly classified into two main categories**:** LSB replacement and LSB matching. In case of LSB replacement **[1, 2]**, the least significant bit of each pixel of the cover image is replaced by the next bit of the secret message to be embedded. In LSB matching **[3]**, if a mismatch occurs between least significant bit of a byte in the cover image and next bit of the secret message to be embedded, then embedding, in general, is done by increasing or decreasing randomly the content of the byte of the cover image by 1, except at the boundary values. In some techniques, the decision to increase or decrease the content of a byte is governed by the score of the distortion function **[4]**. Embedding in two least significant bits is an extension of LSB replacement. There are multiple ways to embed data by flipping the least and the second least bits of a cover image **[5]**.

In case of transform domain, the LSB-based embedding is done by modifying the LSB of non-zero DCT coefficients of a cover image. Several ways to embed data in transform domain exists, such as modification of quantization table, heuristic based, utilizing non-shared selection, and side information at sender side **[6]**.

Steganalysis tools track the distortion caused during the data embedding to detect the presence of the secret message in an image. These tools are classified as visual, structural, and non-structural **[7,8]**. Visual steganalysis attacks analyse images for some distortions that are visible to human vision system. The distortions could be visible in stego image or in LSB plane extracted from the stego image. Structural properties of an image is analysed by structural attacks to find any anomaly which are introduced by steganography. Structural detectors such as histogram attack **[9]**, sample pair analysis (SPA) **[10]**, RS method **[11]**, and weighted stego **[12]** can detect the presence of stego data and even estimate message length reliably. Non-structural detectors use feature extractors to model cover image and to compute distortion between the cover and the stego image to detect embedding. A classifier is trained by the feature set from large number of stego and cover images. During training, the classifier learns the differences in features, and this learning is used to classify a fresh image into stego or clean image. Non-structural detectors such as subtractive pixel adjacency matrix (SPAM) **[13]** and spatial-rich model (SRM) **[14]** claim better probability of detection of embedding in a stego image. Features based on steganalysis techniques use support vector machine (SVM) or ensemble classifiers **[15]** for supervised learning. SVM is not suitable for any high-dimension feature vector, while this is not the case with ensemble classifier but its performance is comparable to SVM.

Most of the current steganography techniques are based on model-preserving principles. These techniques are designed by finding a model for cover images, and embedding modifications are done in such a way that this model is preserved. Highly undetectable stego (HUGO) **[4]**, ASO **[16]**, universal wavelet relative distortion (UNIWARD) **[17]**, and maximum mean discrepancy (MMD) **[18]** are designed on this principle. HUGO preserves features used by SPAM for steganalysis, thus preserving features space model. Similarly, UNIWARD preserves a wavelet-based model, while MMD preserves parametric-based model. Generally, these techniques embed message by minimizing a defined embedding distortion function heuristically. But, in **[19]**, a non-heuristic distortion function is used to preserve the Kullback Leibler distance.

In **[20]**, an embedding technique, known as pixel value difference technique (PVD) has been proposed. In this technique, the image is divided into non-overlapping blocks of adjacent pixels which are randomly selected, and data is embedded into each of its pixels. The amount of data embedded, i.e., the number of last significant bits used, is directly proportional to the differences in the intensities of adjacent pixels. This uneven embedding in PVD leads to unusual steps in the histogram of pixel difference in the stego image. An improved technique (IPVD), proposed in **[21]**, has exploited this vulnerability. Adaptive edge LSB technique (AE-LSB) **[22]** has also removed this uneven pixel difference by introducing a readjusting phase and has provided better capacity. All these techniques are edge adaptive in a way that they can embed more data where pixel difference is high but they have one fundamental limitation. These techniques consider pixel pair at random, rather than selecting on the basis of higher differences. So, they may end up by embedding data at random places in the image and by distorting the texture in LSB plane of the image. Performance of these techniques are found to be poor **[23]**.

In hiding behind corners (HBC) **[24]** technique, corner pixels are used to contain hidden data. Data is embedded by using simple LSB substitution. Such embedding leads to many structural asymmetries and could easily be detected by structural steganalysis tools like chi-square **[9]**, sample pair analysis (SP) **[25]**, and weighted stego (WS) **[26]**. Thus, the HBC technique which maintains texture in LSB plane, offers poor security.

Edge adaptive image steganography (EALMR) **[23]** technique is based on LSB matching revisited (LSBMR) **[2]** technique which alleviates some of the above said limitations. EALMR calculates the difference between two adjacent pixels. If this difference is greater than a pre-defined threshold, then both pixels are marked as edge pixels, and one bit of data is hidden in each of them using LSBMR. This technique has some limitations. Difference of intensities of adjacent pixels may not be an edge point; any such technique may embed data in smoother parts even though there are some unused prominent edges. So, any well-known edge detection algorithm can be used to find edge pixels and to hide data in the detected edges. Further, since EALMR compares a pixel with its adjacent pixel, it can find edges only in one direction. To overcome this limitation, an image can be divided into some non-overlapping but equal size blocks, and each block is rotated in the range of set {0°, 90°, 180°, 270°} to see edge pixels in more than one direction inside a given block. But, poor edge selection results in detection by steganalysis tools like targeted attack **[27]** and blind attacks SPAM **[13]** and SRM **[14]**.

In **[4]**, HUGO steganographic technique is presented. Its design is derived from the image model obtained from the feature set of SPAM steganalyzer. It is based on the minimum-embedding impact principle, where embedding is done in such a way that the distortion in a stego image is minimum. It preserves a model utilized by SPAM steganalyzer to derive steganalytic features in such a way that it does not over-fit to a SPAM feature set. Dimensionality of the feature set has been tremendously enhanced so that the technique is not detectable by minor modification in SPAM steganalyzer. Instead of using Markov transition matrix to compute SPAM features, co-occurrence matrix is used to derive those features. But, it may have minor degradation in performance. Detectable parts of the model are identified by Fisher linear discriminant (FLD criteria) **[28]**. It rates individual features' importance for embedding changes. The parts of the model not vulnerable to embedding changes are identified using criteria optimized in FLD. In **[29]**, it is shown that HUGO is vulnerable against steganalysis that uses other models drawn from different domains.

In **[30]**, embedding distortion cost is computed through directional residual obtained using Daubechies wavelet filter bank **[31]**. The objective is to limit the embedding changes to those parts of the cover image that are difficult to model in multiple directions. Embedding is done in textures or noisy parts and avoiding smooth regions and clean edges of empirical cover images. Distortion function, called as UNIWARD **[30]**, is used to compute delectability map. Syndrome trellis code (STC) **[32]** and detectability map are used to embed payload while minimizing the embedding distortion. The same distortion design technique can be used for spatial and transform domains.

There are a large number of cryptographic and steganographic methods that most of us are familiar with. The most widely used two techniques are:

I. RSA Algorithm: RSA algorithm is a message encryption cryptosystem in which two prime numbers are taken initially and then the product of these values is used to create a public and a private key, which is further used in encryption and decryption. The RSA algorithm could be used in combination with Hash-LSB in a way that original text is embedded in the cover image in the form of cipher text.

II. LSB Insertion Method: One of the most common techniques used in steganography today, least significant bit (LSB) insertion is the process of adjusting the least significant bit pixels of the carrier image. It is a simple approach for embedding message into the image. In this method some information from the pixel of the carrier image is replaced with the message information so that it can't be observed by the human visual system, therefore it exploits some limitations of the human visual system.

Chapter 3:

SYSTEM DESIGN

SENDER'S PART

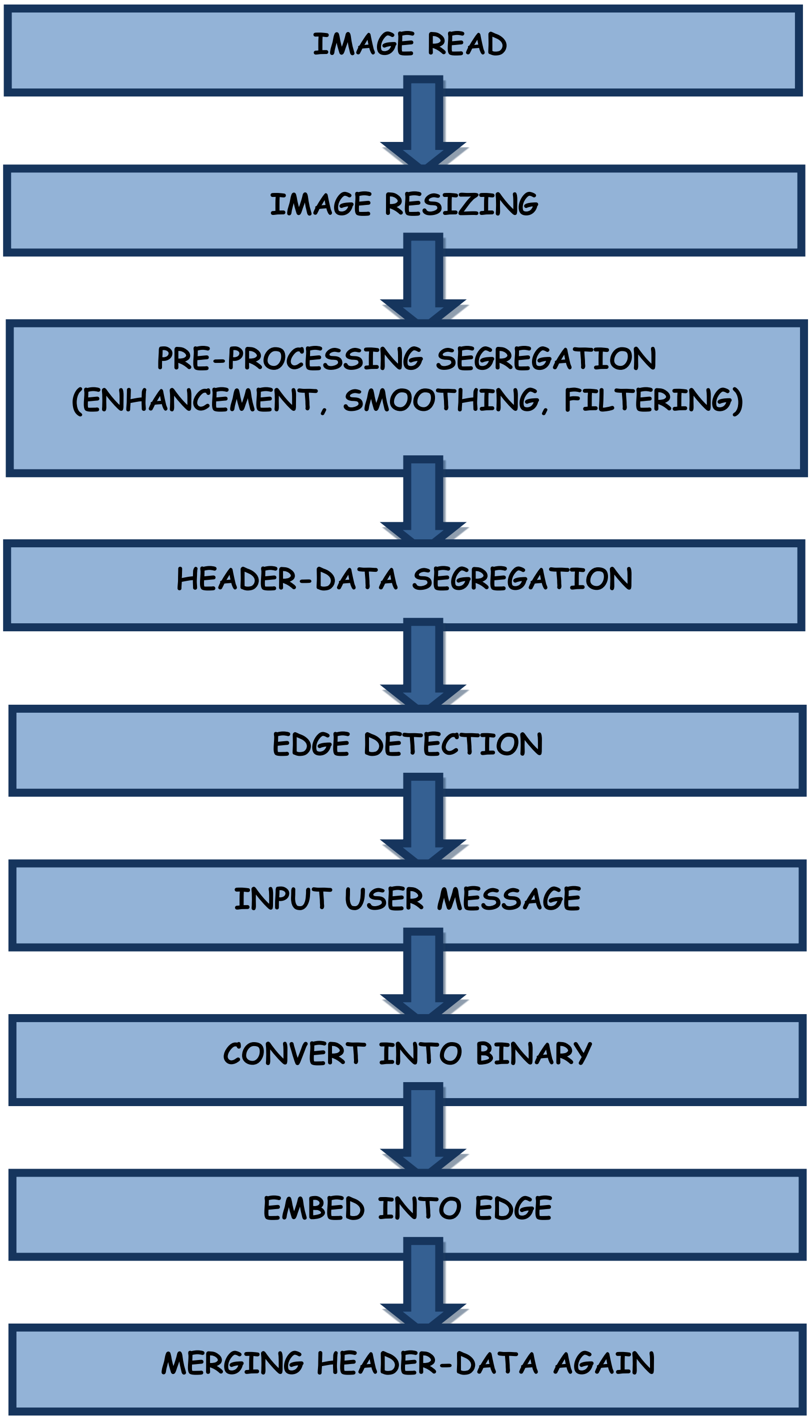


Fig4

RECEIVER'S PART

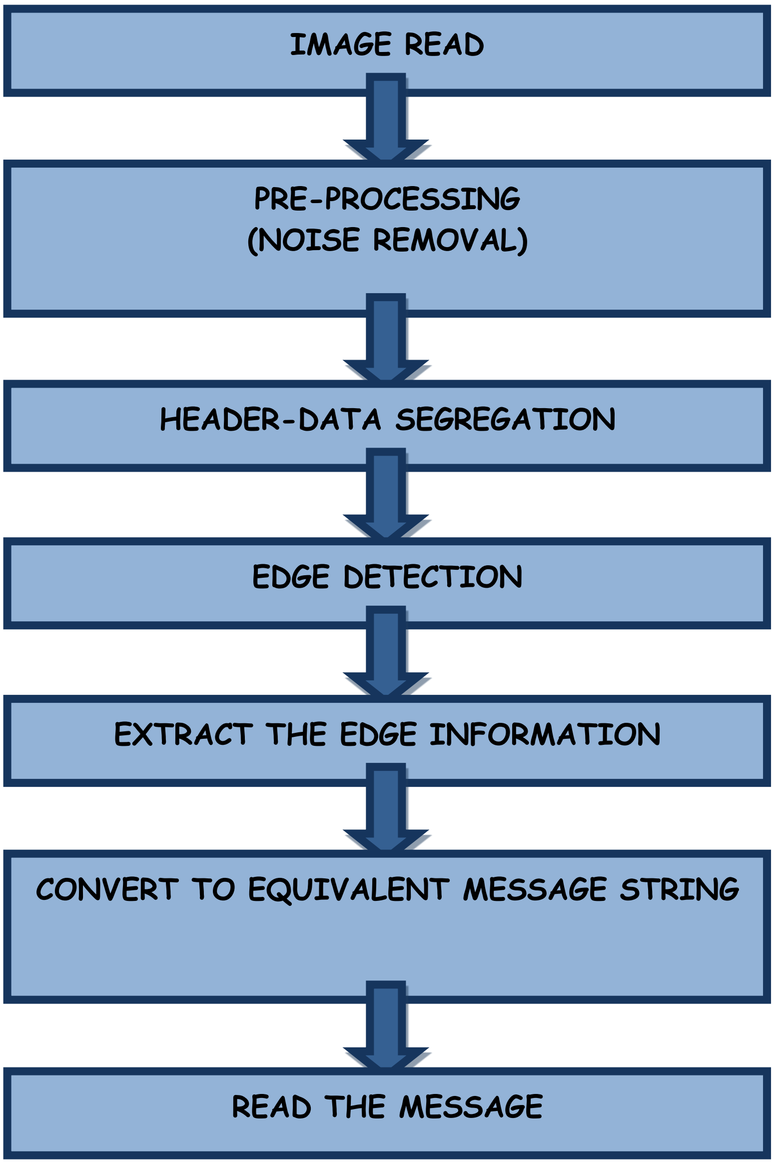


Fig5

Chapter 4:

METHODOLOGY FOR IMPLEMENTATION

Aim1: Appropriate Positions To Hide

TASK 1: BOUNDARY TRACING

AIM- To Gather Boundary Pixels

PURPOSE- To Store Random Values

Contour tracing:

Contour tracing is a technique that is applied to digital bi-level images in order to extract their boundary.

A ***digital image*** is a group of pixels on a square tessellation each having a certain ***value***. In ***bi-level*** images each pixel can have one of 2 possible values, namely:

***1**, where the pixel is considered to be a **"black"** pixel and will be part of the **pattern**,*

OR

***0**, where the pixel is considered to be a **"white"** pixel and will be part of the **background**.*

The boundary of a pattern given as **P**, is the set of ***border pixels*** of **P**. In a square tessellation, there are 2 kinds of boundary (or ***border***) pixels: ***4-border*** pixels and ***8-border*** pixels. (A 4-border pixel may be taken as an 8-border pixel. An 8-border pixel may or may not be taken as a 4-border pixel.)

It is not enough to merely **identify** the boundary pixels of a pattern in order to extract its contour. An **ordered sequence** of the boundary pixels is necessary from which the general **shape** of the pattern can be extracted.

Contour tracing or boundary tracing is used to find the boundary pixels of the image. These pixels are used to store the location of the edge pixels and corner point pixels used to store the message and length of the message respectively.

Square Tracing Algorithm:

Idea:

The idea behind the square tracing algorithm is very simple. The algorithm was one of the first attempts to extract the contour of a binary pattern.

A digital pattern i.e. a group of black pixels, on a background of white pixels or a grid is given. A black pixel is located and declared as the "start" pixel. (Locating a "start" pixel can be done in various ways; It can start at the bottom left corner of the grid, scan each column of pixels from the bottom going upwards - starting from the leftmost column and proceeding to the right - until a black pixel is encountered. This pixel is declared the "start" pixel.)

Now, imagine a bug (ladybird) standing on the start pixel. In order to extract the contour of the pattern,

- every time the bug stands on a black pixel, it has to turn left, and
- every time it stands on a white pixel, it has to turn right,

until it encounters the start pixel again.
The contour of the pattern will be the black pixels it walked over.

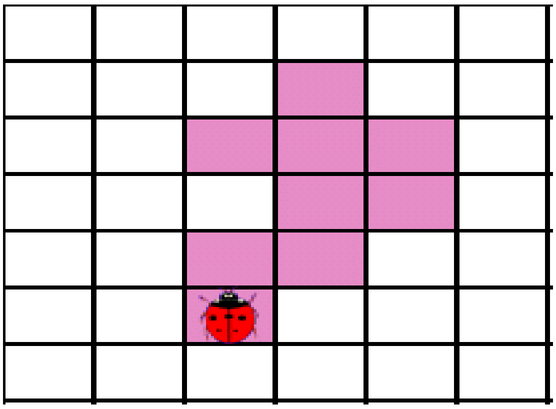


Fig6.

The important thing in the square tracing algorithm is the "sense of direction". The left and right turns are made with respect to the current positioning, which depends on the way the bug entered the pixel it is standing on. Therefore, it is important to keep track of the current orientation in order to make the right moves.

Demonstration:

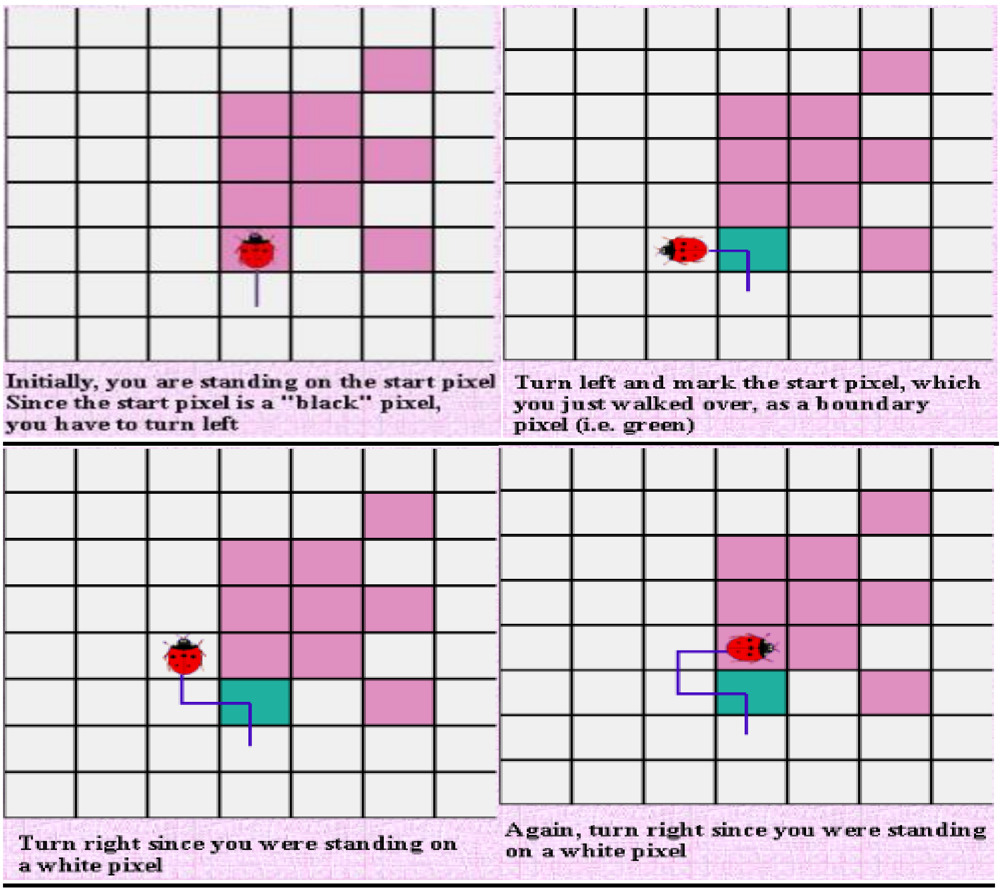


Fig7.

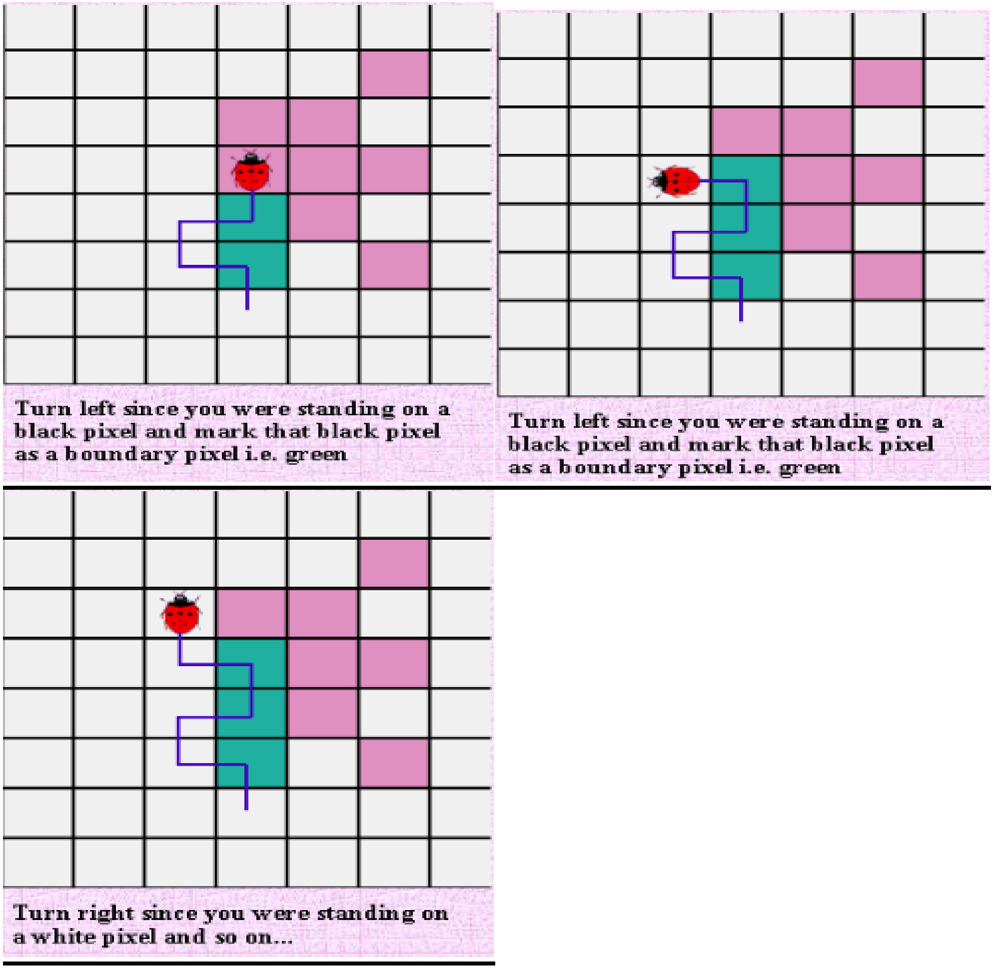


Fig8.

<u>*Radial Sweep Algorithm:*</u>

Idea:

The Radial Sweep algorithm searches the Moore neighborhood (i.e. the set of 8 pixels that share a vertex or edge with that pixel) of the current boundary pixel in a certain direction (clockwise), until it finds a black pixel. It declares this pixel as the current boundary pixel and then proceeds as before.

The Radial Sweep algorithm is an interesting method in order to find the next black pixel in the Moore neighborhood of a given boundary pixel.

The idea behind this method is as following:

Every time a new boundary pixel is located within the image, it is made the current pixel, **P,** and an **imaginary line segment** joining **P** to the **previous** boundary pixel is drawn. Then, the segment is rotated about **P** in a clockwise direction until it hits a black pixel in **P**'s Moore neighbourhood.

Demonstration:

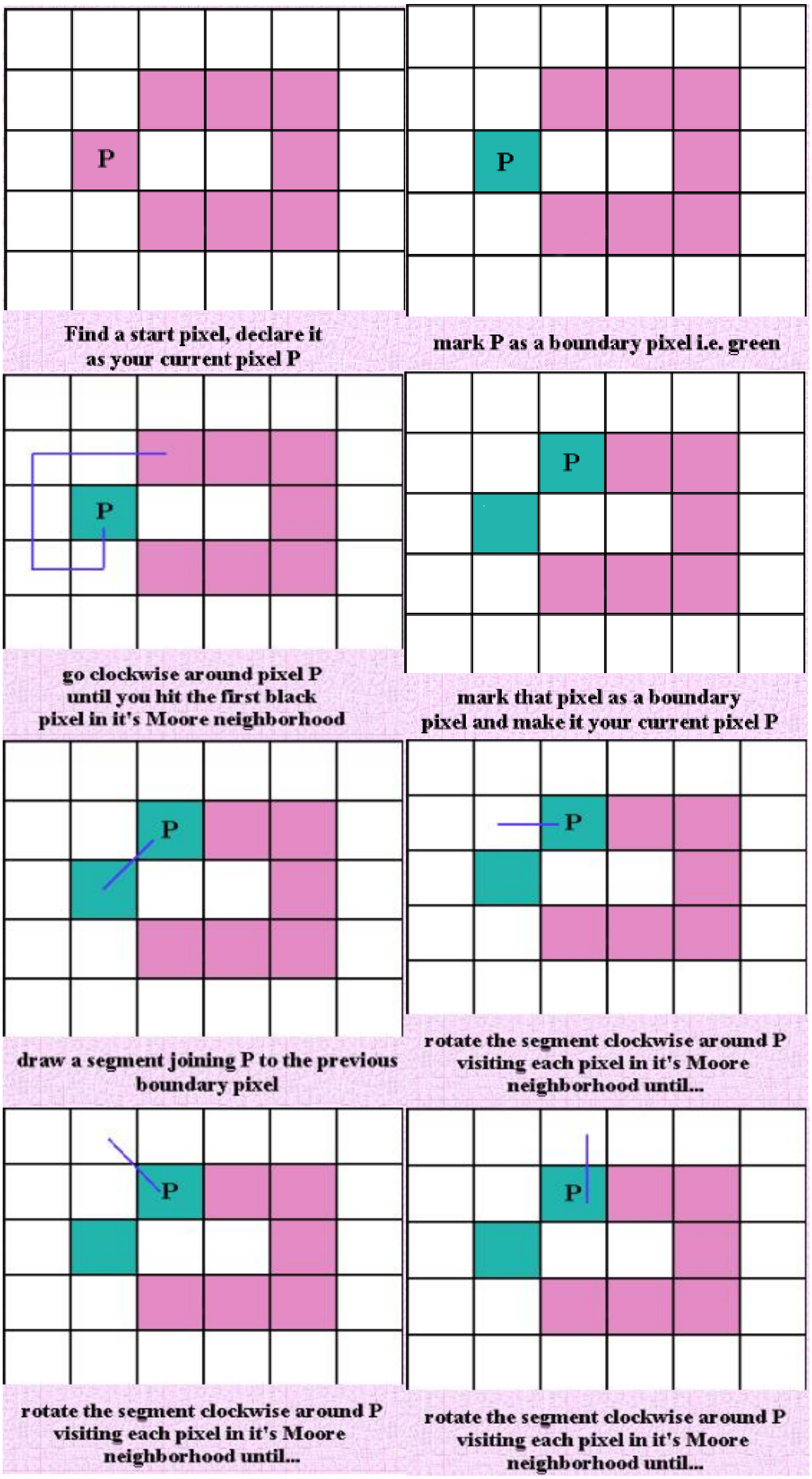


Fig9.

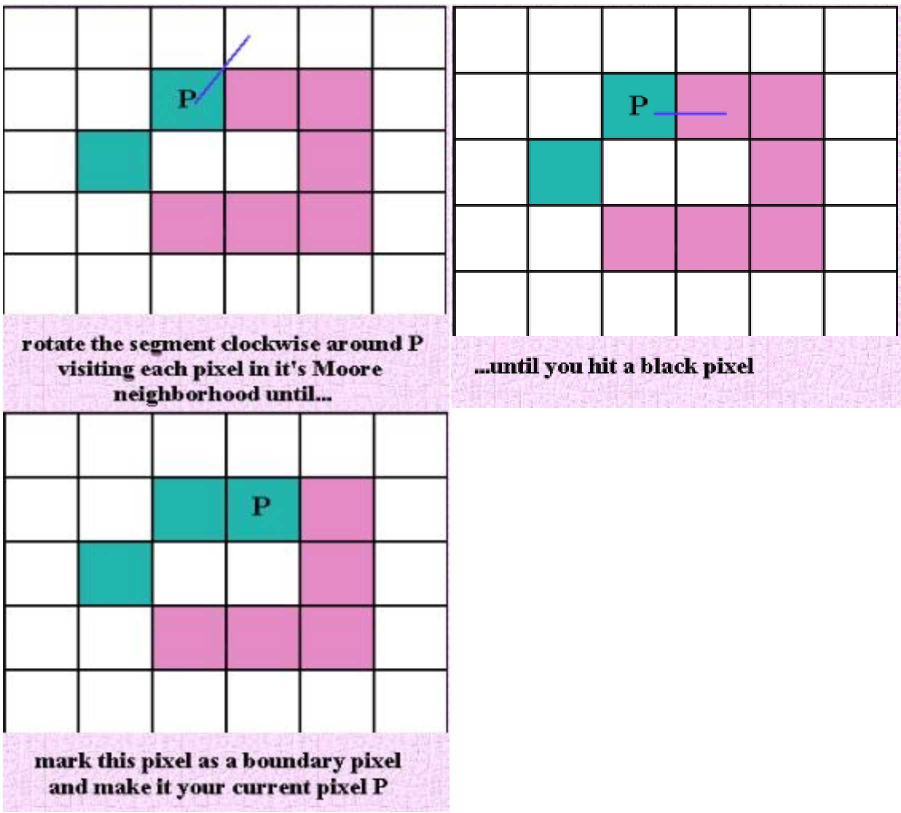


Fig10.

<u>Theo Pavlidis' Algorithm:</u>

Idea:

This algorithm is one of the more recent contour tracing algorithms and was proposed by Theo Pavlidis. Pavlidis chooses to trace the contour in a counter-clockwise direction.

Given a digital pattern i.e. a group of black pixels, on a background of white pixels, or a grid. A **start** pixel is chosen satisfying the following restriction imposed on the choice of a start pixel for Pavlidis' algorithm:

Important restriction regarding the direction in which the start pixel is entered:

ANY black boundary pixel can be chosen to be the **start** pixel as long as when initially standing on it, the left adjacent pixel is NOT black. In other words, it should be made sure that the **start** pixel is entered in a direction which ensures that the left adjacent pixel to it will be white ("left" here is taken with respect to the direction in which the **start** pixel is entered).

Now, imagine a bug (ladybird) standing on the **start** pixel.

Throughout the algorithm, the pixels which interest you at any time are the 3 pixels in front i.e. **P1, P2** and **P3**. **P2** is defined to be the pixel right **in front**, **P1** is the pixel adjacent to **P2** from the left and **P3** is the right adjacent pixel to **P2**.

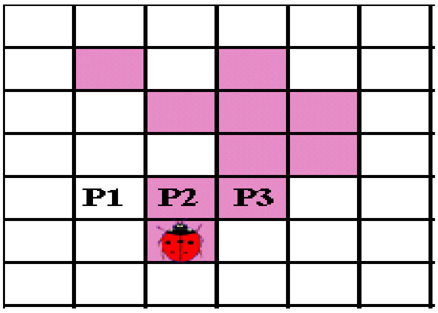


Fig11.

Every time the bug is standing on the current boundary pixel (which is the **start** pixel at first) it does the following:

First, check pixel **P1.** If **P1** is black, then declare **P1** to be your current boundary pixel and **move one step forward followed by one step to your current left** to land on P1. The **order** in which the moves are made is very important.
The figure below demonstrates the path to follow in order to land on **P1** drawn in blue.

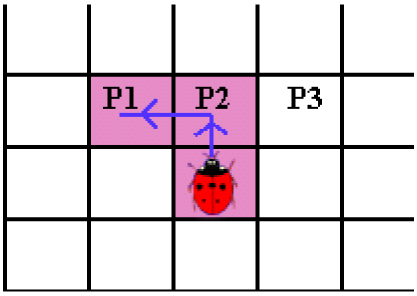


Fig12.

Only if P1 is white proceed to check P2...

If **P2** is black, then declare **P2** to be the current boundary pixel and **move one step forward** to land on **P2**. ***The figure below*** below demonstrates the path to follow in order to land in **P2** drawn in blue.

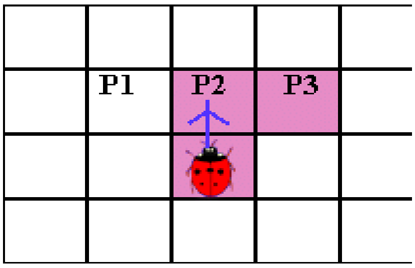


Fig13.

Only if both P1 and P2 are white proceed to check P3...

If **P3** is black, then declare **P3** to be the current boundary pixel and **move one step to your right followed by one step to your current left** as demonstrated in Figure 4 below.

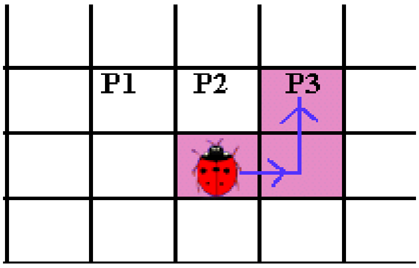
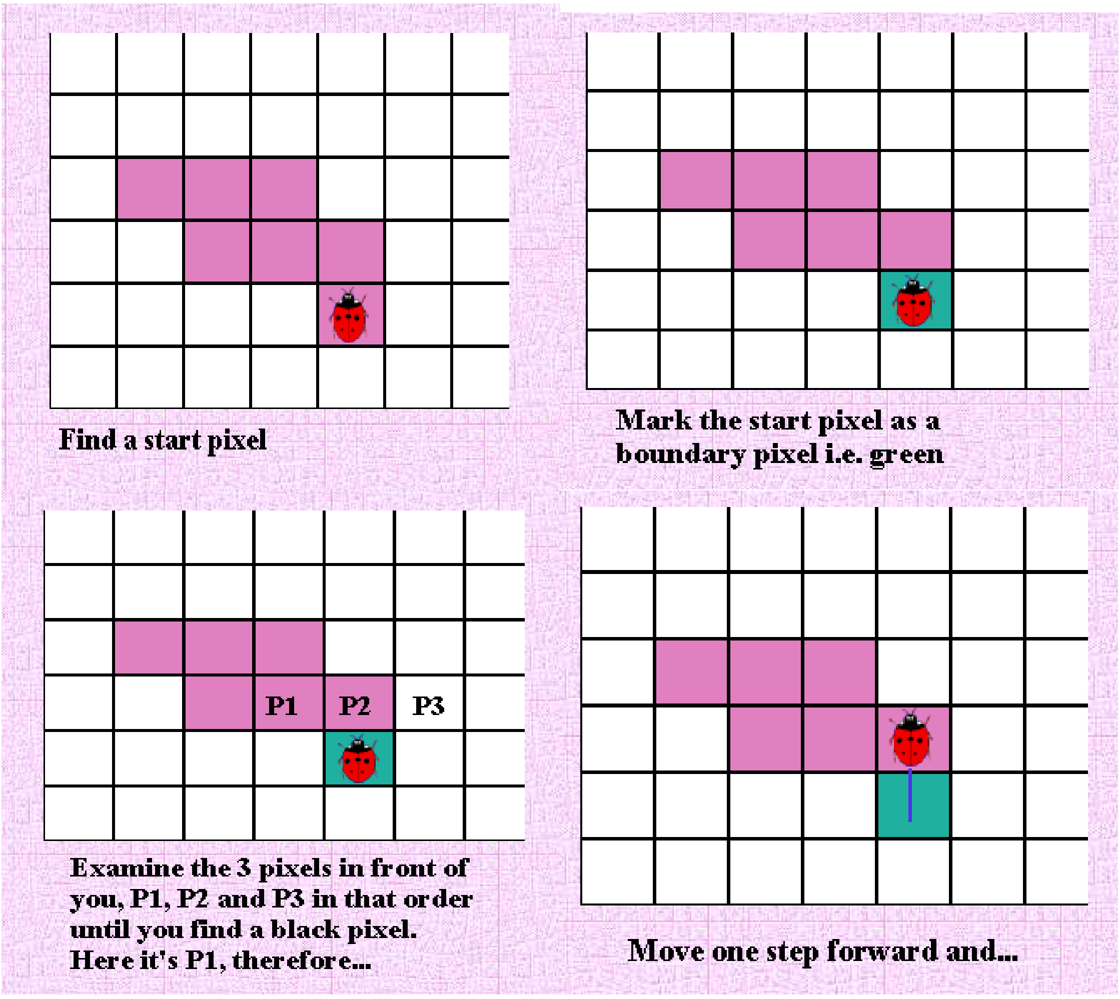
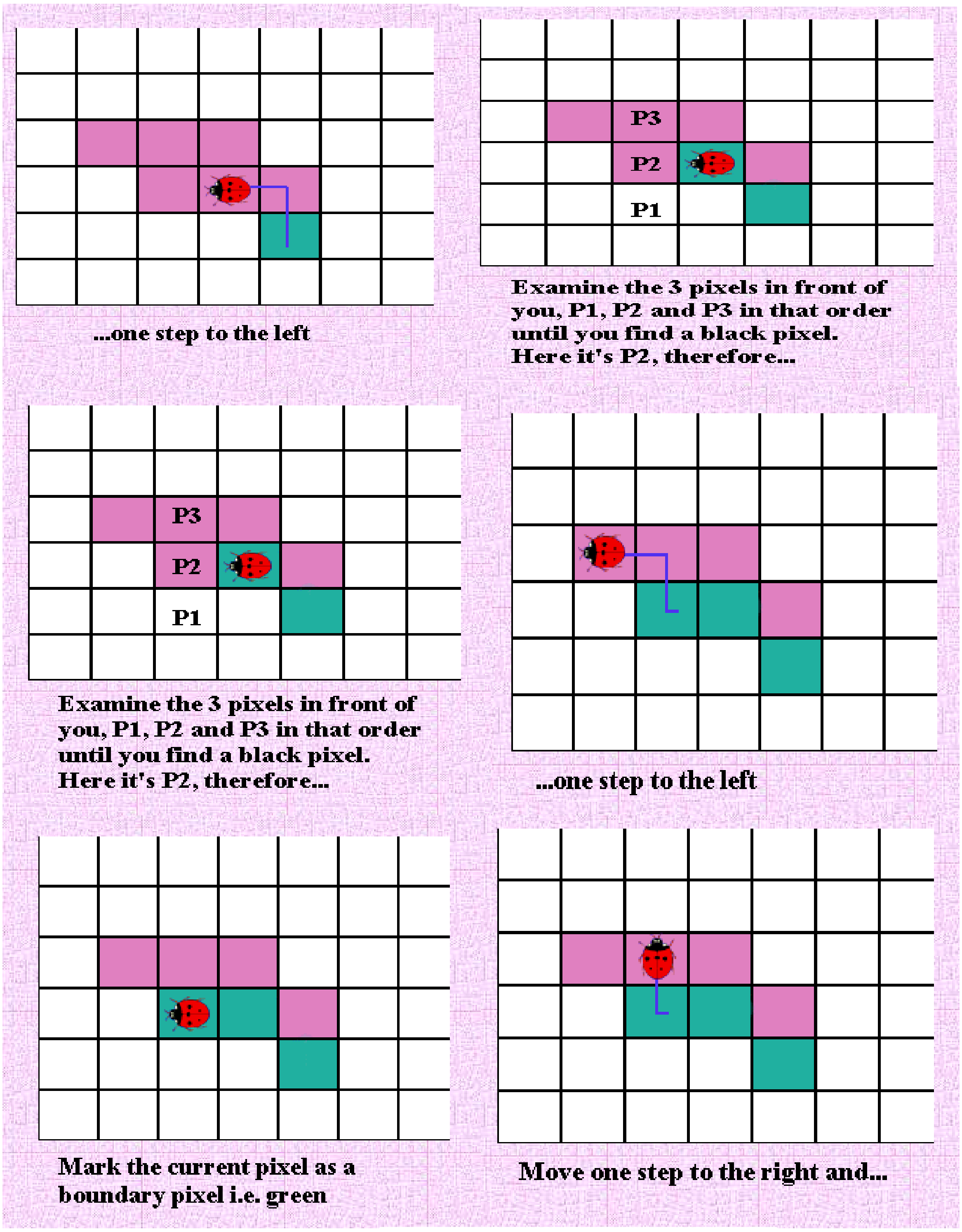


Figure 4

Demonstration:

The following is a demonstration of how Pavlidis' algorithm traces the contour of a given pattern. The bug (ladybird) walking over the pixels change their orientation as it turns left or right.




P3
P2
P1
...one step to the left
Examine the 3 pixels in front of you, P1, P2 and P3 in that order until you find a black pixel. Here it's P2, therefore...
P3
P2
P1
Examine the 3 pixels in front of you, P1, P2 and P3 in that order until you find a black pixel. Here it's P2, therefore...
...one step to the left
Mark the current pixel as a boundary pixel i.e. green
Move one step to the right and...

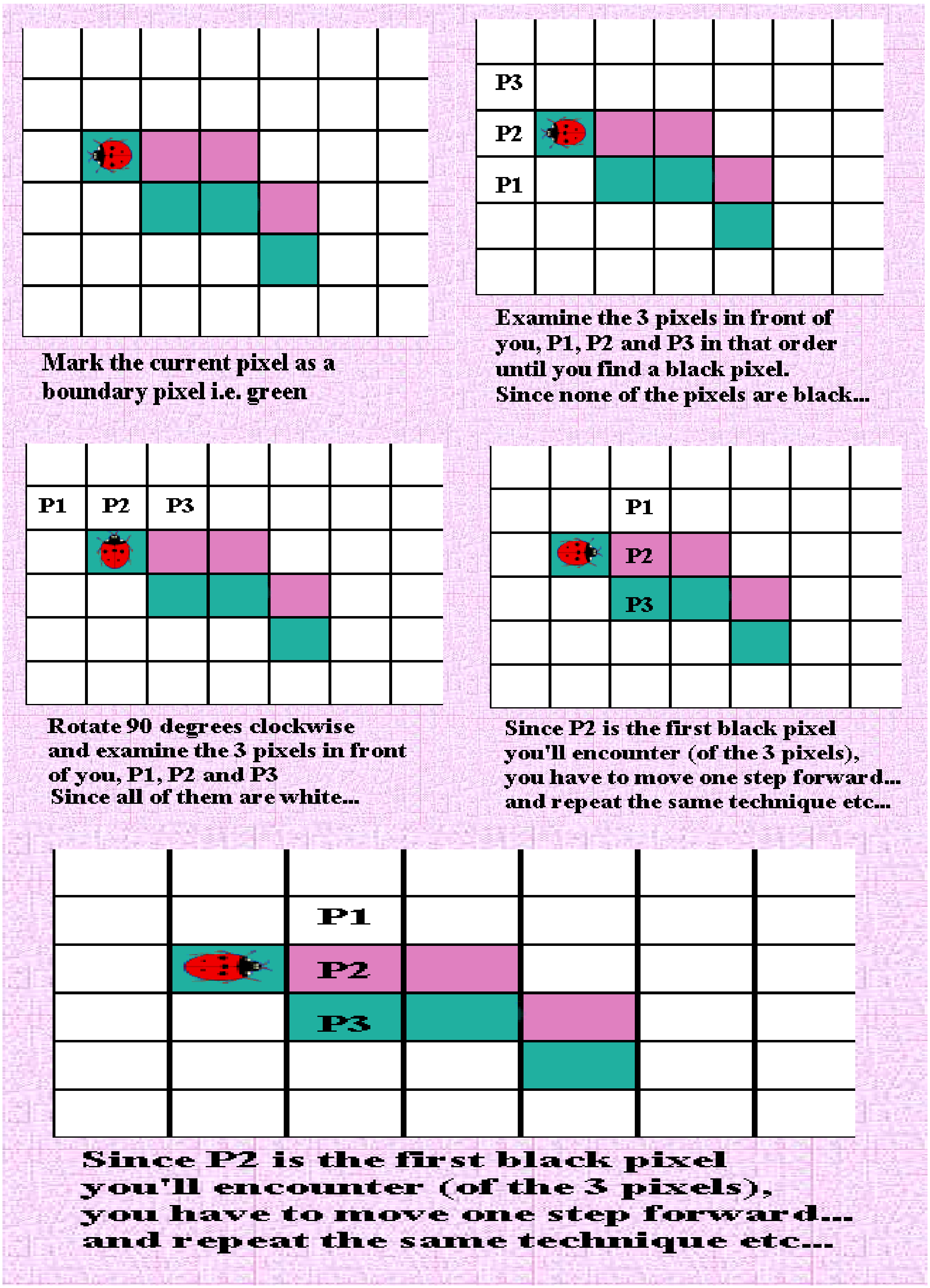


Fig14.

Analysis:

The Theo Pavlidis algorithm is more complex than, Moore-Neighbor tracing which has no special cases to take care of, yet it fails to extract the contour of a large family of patterns having a certain kind of connectivity.

The algorithm works very well on 4-connected patterns. However, its problem lies in tracing some 8-connected patterns that are not 4-connected.

The following is a demonstration of how Pavlidis' algorithm fails to extract the contour of an 8-connected pattern (that is not 4-connected) by missing a large portion of the boundary.

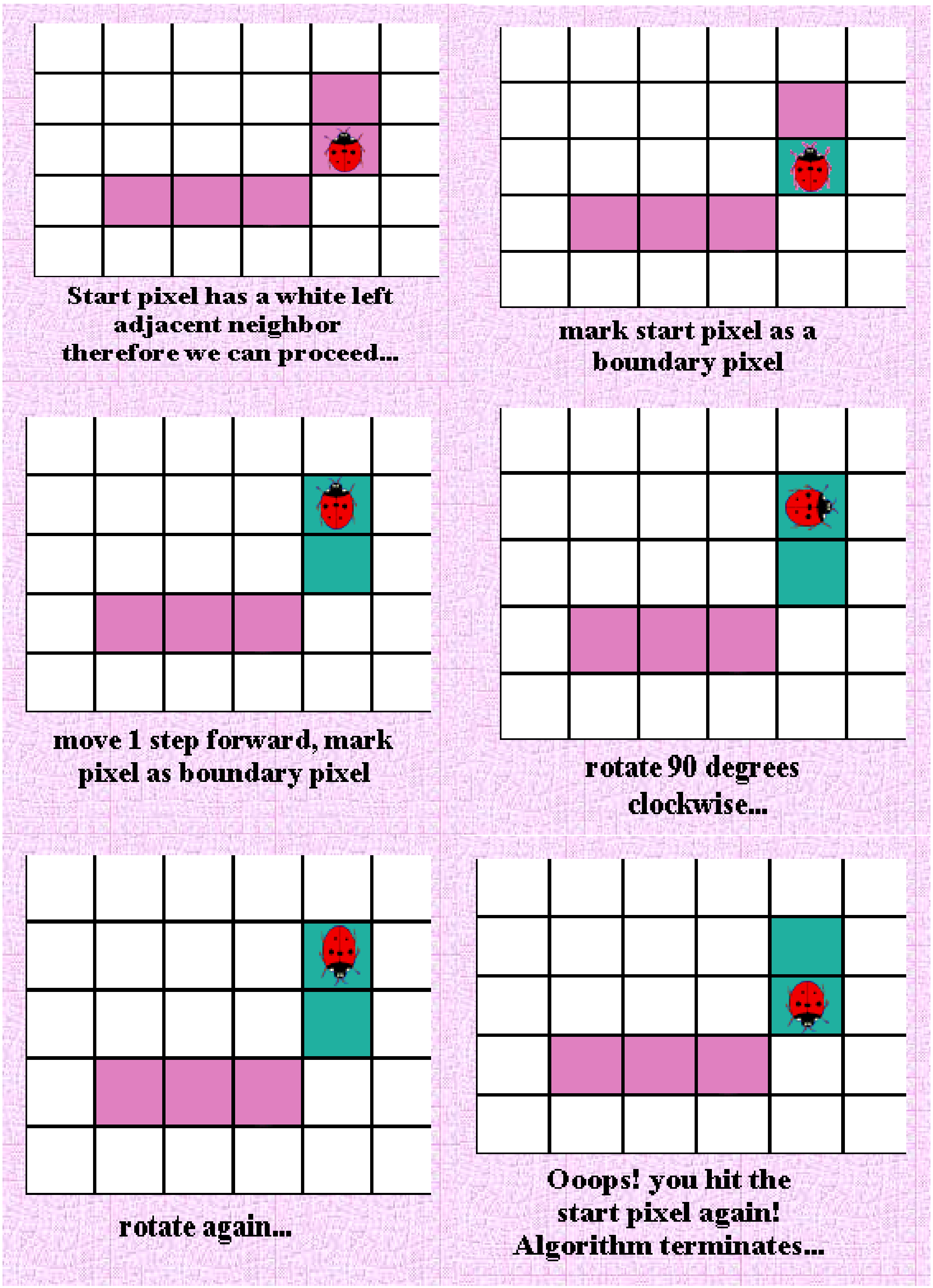


Fig15.

<u>Moore-Neighbor Tracing:</u>

Idea:

The idea behind Moore-Neighbor tracing is simple.

The Moore neighbour of a pixel, P, is the set of 8 pixels which share a vertex or edge with that pixel. These pixels are namely pixels P1, P2, P3, P4, P5, P6, P7 and P8 shown in the figure below.
The Moore neighborhood (or the *8-neighbors* or *indirect neighbors*) of a pixel is an important concept for edge detection or contour tracing.

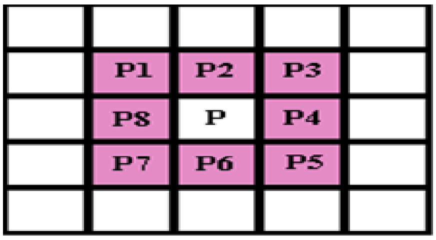
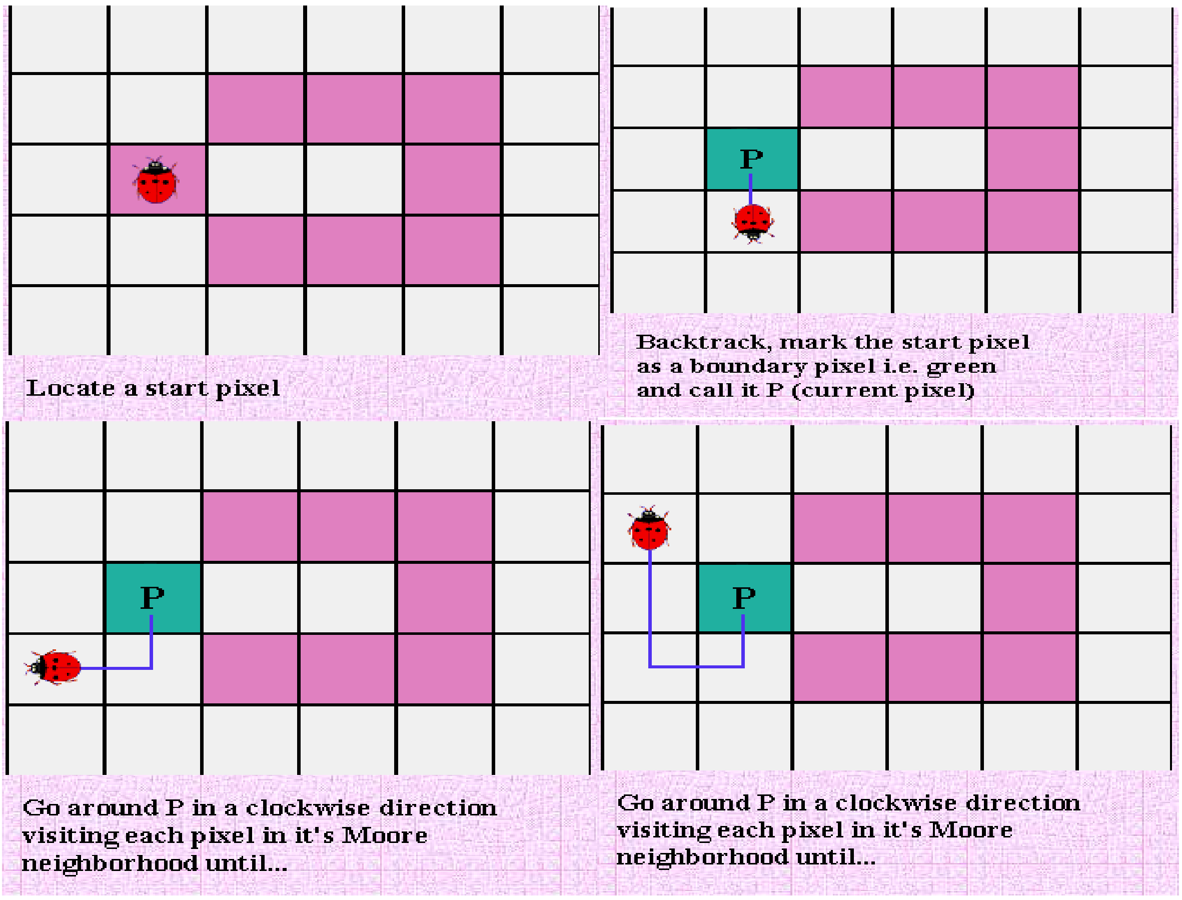


Figure 1

Demonstration:

The following images demonstrate how Moore-Neighbor tracing proceeds to trace the contour of a given pattern.



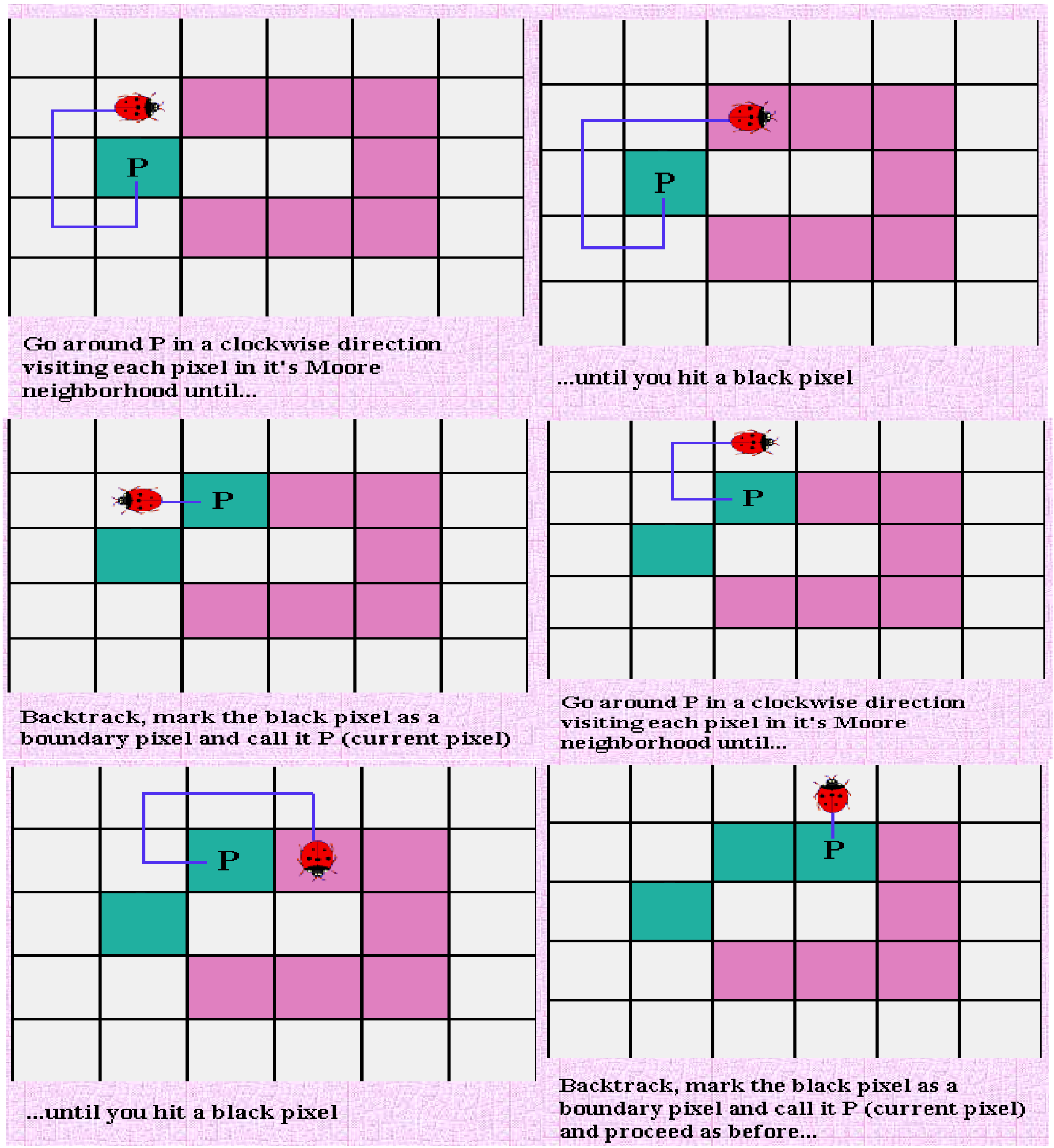


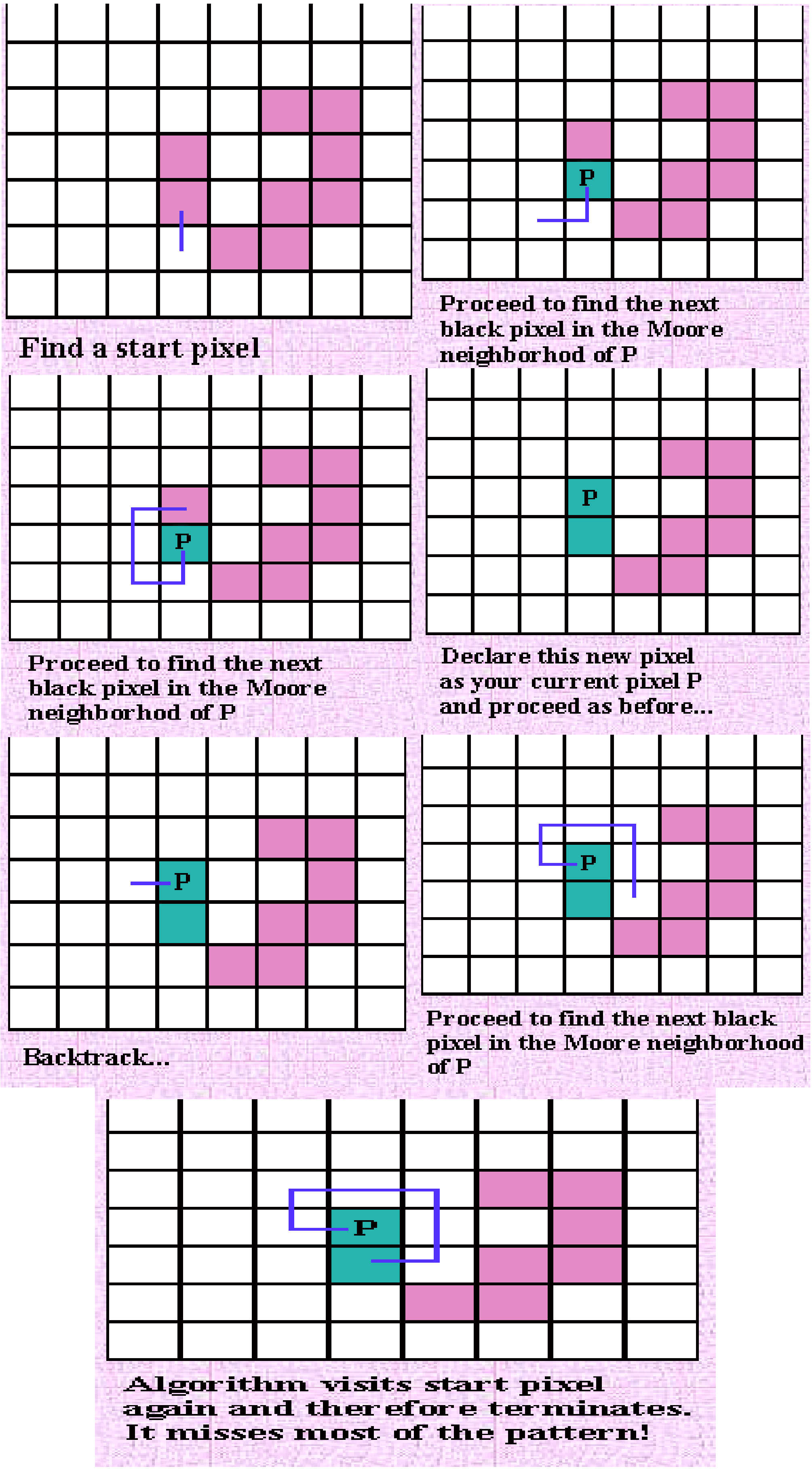
Fig16.

Analysis:

The main weakness of Moore-Neighbor tracing appears in the choice of the stopping criterion.

In the original description of the algorithm used in Moore-Neighbor tracing, the stopping criterion is to visit the start pixel for a second time. As seen for the Square Tracing algorithm, Moore-Neighbor tracing will fail to contour trace a large family of patterns if it depends on this criterion.

A demonstration explaining how Moore-Neighbor tracing fails to extract the contour of a pattern due to the bad choice of the stopping criterion is shown below:


Find a start pixel
P
Proceed to find the next
black pixel in the Moore
neighborhod of P
P
Proceed to find the next
black pixel in the Moore
neighborhod of P
P
Declare this new pixel
as your current pixel P
and proceed as before...
P
Backtrack...
P
Proceed to find the next black
pixel in the Moore neighborhood
of P
P
Algorithm visits start pixel
again and therefore terminates.
It misses most of the pattern!

Our Approach for Boundary Tracing:

We have used the Moore-Neighbor tracing algorithm to trace the boundary of an image.

As we can see, improving the stopping criterion would be a good start to improving the overall performance of Moore-Neighbor tracing. There are some effective alternatives to the existing stopping criterion. These are:

a) Stopping after visiting the **start** pixel ***n*** times, where n is at least 2,

OR

b) Stopping after entering the **start** pixel a second time **in the same manner it was entered initially.** This criterion was called ***Jacob's stopping criterion.***

Jacob's stopping criterion greatly improves the performance of Moore-Neighbor tracing.

The whole Moore-Neighborhood of a boundary pixel is checked in Moore-Neighbor tracing algorithm in order to find the next boundary pixel. The Square Tracing algorithm makes left or right turns and misses the "diagonal" pixels; in the Moore-Neighbor tracing algorithm, the outer boundary of any connected component is always extracted.

The algorithm usually terminates when the boundary pixel P is visited for a **second** time provided that the boundary pixel before P the second time around, is the **same** pixel which was before P when it was **first** visited.

When the stopping criterion is satisfied and the algorithm is not terminated, the Radial Sweep algorithm traces the **same** boundary for a second time.

The algorithm works on all four-connected patterns, but the problem lies in tracing some eight-connected patterns that are not four-connected.

Code for Boundary tracing algorithm:

```
%% Boundary Tracing
fprintf('Boundary Tracing\n====================\n');
input=imread('H:\4th Year Project_Jhilik\Images\baboon.jpeg');
pic=input;
input=rgb2gray(input);
s=size(input);
a=s(1)*2+s(2)*2-4;
boundary=zeros(a,2);
c=0;
for i=1:s(2)
    c=c+1;
    boundary(c,1)=1;
    boundary(c,2)=i;
end
for i=2:s(1)
    c=c+1;
    boundary(c,1)=i;
    boundary(c,2)=s(2);
end
for i=s(2)-1:-1:1
    c=c+1;
    boundary(c,1)=s(1);
    boundary(c,2)=i;
end
for i=s(1)-1:-1:2
    c=c+1;
    boundary(c,1)=i;
    boundary(c,2)=1;
end
disp(boundary);
```

OUTCOME:

A sequence points that determines the boundary pixels i.e. the contour of an image.

TASK 2: KEY POINT DETECTION

- ***AIM- TO DETERMINE THE KEY/CORNER POINTS***
- ***PURPOSE- TO STORE MESSAGE LENGTH***

Corner Point Detection:

Corner Point Detection is an image processing technique used to find the minimum number of points required to create an image, i.e. by joining the key points, we can get the outline of the image.

The corner points (or key points) are used to store the length of the message embedded in the image.

Moravec Corner Detector:

Idea:

This is one of the earliest corner detection algorithms and defines a *corner* to be a point with low self-similarity. The Moravec Corner Detector algorithm tests all the pixels in the image to see if a corner is present by considering how similar a patch centered on the pixel is to nearby, largely overlapping patches. The sum of squared differences (SSD) is used to measure the similarity between the corresponding pixels of two patches. A lower number indicates more similarity.

The nearby patches will look similar ff the pixel is present in a region of uniform intensity. If the pixel is on an edge, then the nearby patches in a direction perpendicular to the edge will look quite different, but nearby patches in a direction parallel to the edge will result in only in a small change. None of the nearby patches will look similar if the pixel is on a feature with variation in all directions.

As pointed out by Moravec, one of the major problems with this operator is that it is not isotropic if an edge and the smallest SSD will be large and the edge will be incorrectly chosen as an interest point.

Harris Corner Detector:

Idea:

Harris Corner Detector is commonly used in computer vision algorithms to extract corners and infer features of an image. Harris' corner detector takes the differential of the corner score into account instead of using shifting patches for every 45 degree angles. It has been proved to be more accurate in distinguishing between edges and corners. Since then, it has been improved and adopted in many algorithms to preprocess images for subsequent applications.

Our Approach for Key Point Detection:

We have used the Harris Corner Point Detection algorithm for finding out the corner points of an image.

Out of the Harris Corner Detection and Moravec Corner Detection, there are some shortcomings of Moravec Detection:

- Only tries 4 shifts.
- Uses a discrete rectangular window.

- Uses a simple min function.

But in Harris:

- We can consider “all” shifts.
- We can use a smooth circular (or later elliptical) window.
- We can characterize variation with respect to direction.

Code for Harris Corner Point Detector:

```
%% Key Point Detection
fprintf('Harris\n===========\n');
%img=getappdata(0,'image');
img=pic;
 im=img;
im=im(:,:,1);

sigma=1;
radius=1;
order=2*radius+1;
threshold=7000;
%derivatives in x and y direction
[dx,dy]=meshgrid(-1:1,-1:1);

Ix=conv2(double(im),dx,'same');
Iy=conv2(double(im),dy,'same');

% Implementing the Gaussian filter

dim=max(1,fix(6*sigma));
m=dim;
n=dim;
[h1,h2] = meshgrid(-(m-1)/2:(m-1)/2 , -(n-1)/2: (n-1)/2);
hg=exp(-(h1.^2+h2.^2)/(2*sigma^2));
[a b]=size(hg);
sum1=0;
for i=1:a
    for j=1:b
        sum1=sum1+hg(i,j);
    end
end
g=hg ./sum1;
%Calculate the entries of the M-matrix

Ix2=conv2(double(Ix.^2),g,'same');
Iy2=conv2(double(Iy.^2),g,'same');
Ixy=conv2(double(Ix.*Iy),g,'same');

%Harris measure
R=(Ix2.*Iy2 - Ixy.^2) ./ (Ix2+Iy2 + eps);

%Find local maxima
mx=ordfilt2(R,order^2,ones(order));
```

```
harris_points=(R==mx) & (R>threshold);
%disp(harris_points);
[harris_row,harris_col]=find(harris_points);

figure,imshow(im),hold on,
plot(harris_col,harris_row,'ys'), title('Harris_Corners');
%disp([harris_row,harris_col]);
```

OUTCOME:

A sequence of points that determines the corner pixels.

TASK 3: EDGE DETECTION

- ***AIM- To Find Maximum Number Of Possible Edges***
- ***PURPOSE- To Store The Message***

Edge detection methods for finding object boundaries in images:

In image processing the technique is used to find the boundaries of objects within images os known as Edge Detection.

It detects the discontinuities in brightness and is used for image segmentation and data extraction in image processing, computer vision, and machine vision.

The edge pixels are used in order to store the encrypted message.

Prewitt Operator:

Idea:

Prewitt operator is used for edge detection in an image.

It detects two types of edges:

- Horizontal edges
- Vertical Edges

We can perform this operation on an image using the Prewitt() method in Matlab.

Sobel Operator:

Idea:

The sobel operator is very similar to Prewitt operator. It is also a derivative mask and is used for edge detection.

Like Prewitt operator sobel operator is also used to detect two kinds of edges in an image:

- Vertical edges
- Horizontal edges

We can perform this operation on an image using the Sobel() method in Matlab.

Canny Edge Detection:

Idea:

Canny Edge Detection is used to detect the edges in an image.

It accepts a grayscale image as input and it uses a multistage algorithm.

We can perform this operation on an image using the Canny() method in Matlab.

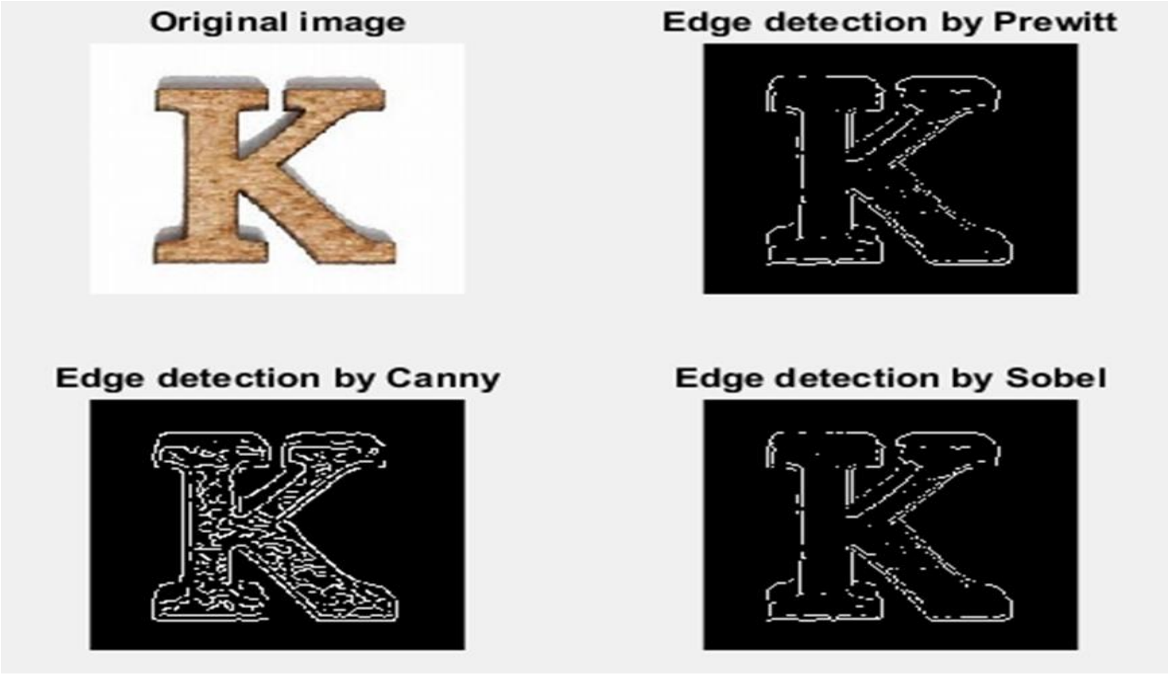


Fig17.

Our Approach for Edge Detection:

We have used Canny edge detection algorithm for detecting the edge pixels of an image.

Sobel and Prewitt operators have only one threshold value. Pixels having value higher than the threshold value are marked as edge pixels while those below are suppressed.

Meanwhile, Canny operator has two threshold values. Pixels above the upper threshold value are marked as strong edge pixels while those below the upper threshold but above the lower threshold are marked as weak edge pixels. Pixels below the lower threshold are suppressed.

Thus, Canny has more advantages since the edge of an object in the image with both strong and weak edges is fully displayed which is not possible in Sobel and Prewitt.

Code for Canny Edge Detection:

```
%% Edge Detection
fprintf('Canny\n===========\n');
%img=getappdata(0,'image');
img=pic;
img = rgb2gray(img);
%img=imresize(img,[256 256]);
img = double (img);

%Value for Thresholding
T_Low = 0.075;
T_High = 0.175;

%Gaussian Filter Coefficient
B = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9, 12, 9, 4;2, 4, 5,
4, 2 ];
B = 1/159.* B;
```

```
%Convolution of image by Gaussian Coefficient
A=conv2(img, B, 'same');

%Filter for horizontal and vertical direction
KGx = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
KGy = [1, 2, 1; 0, 0, 0; -1, -2, -1];

%Convolution by image by horizontal and vertical filter
Filtered_X = conv2(A, KGx, 'same');
Filtered_Y = conv2(A, KGy, 'same');

%Calculate directions/orientations
arah = atan2 (Filtered_Y, Filtered_X);
arah = arah*180/pi;

pan=size(A,1);
leb=size(A,2);

%Adjustment for negative directions, making all directions positive
for i=1:pan
    for j=1:leb
        if (arah(i,j)<0)
            arah(i,j)=360+arah(i,j);
        end
    end
end

arah2=zeros(pan, leb);

%Adjusting directions to nearest 0, 45, 90, or 135 degree
for i = 1  : pan
    for j = 1 : leb
        if ((arah(i, j) >= 0 ) && (arah(i, j) < 22.5) || (arah(i, j) >=
157.5) && (arah(i, j) < 202.5) || (arah(i, j) >= 337.5) && (arah(i, j) <=
360))
            arah2(i, j) = 0;
        elseif ((arah(i, j) >= 22.5) && (arah(i, j) < 67.5) || (arah(i, j)
>= 202.5) && (arah(i, j) < 247.5))
            arah2(i, j) = 45;
        elseif ((arah(i, j) >= 67.5 && arah(i, j) < 112.5) || (arah(i, j)
>= 247.5 && arah(i, j) < 292.5))
            arah2(i, j) = 90;
        elseif ((arah(i, j) >= 112.5 && arah(i, j) < 157.5) || (arah(i, j)
>= 292.5 && arah(i, j) < 337.5))
            arah2(i, j) = 135;
        end
    end
end

%figure, imagesc(arah2); colorbar;

%Calculate magnitude
magnitude = (Filtered_X.^2) + (Filtered_Y.^2);
magnitude2 = sqrt(magnitude);

BW = zeros (pan, leb);

%Non-Maximum Supression
for i=2:pan-1
    for j=2:leb-1
        if (arah2(i,j)==0)
```

```
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i,j+1), magnitude2(i,j-1)]));
        elseif (arah2(i,j)==45)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j-1), magnitude2(i-1,j+1)]));
        elseif (arah2(i,j)==90)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j), magnitude2(i-1,j)]));
        elseif (arah2(i,j)==135)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j+1), magnitude2(i-1,j-1)]));
        end
    end
end

BW = BW.*magnitude2;
%figure, imshow(BW);

%Hysteresis Thresholding
T_Low = T_Low * max(max(BW));
T_High = T_High * max(max(BW));

T_res = zeros (pan, leb);

for i = 1  : pan
    for j = 1 : leb
        if (BW(i, j) < T_Low)
            T_res(i, j) = 0;
        elseif (BW(i, j) > T_High)
            T_res(i, j) = 1;
        %Using 8-connected components
        elseif ( BW(i+1,j)>T_High || BW(i-1,j)>T_High || BW(i,j+1)>T_High
|| BW(i,j-1)>T_High || BW(i-1, j-1)>T_High || BW(i-1, j+1)>T_High ||
BW(i+1, j+1)>T_High || BW(i+1, j-1)>T_High)
            T_res(i,j) = 1;
        end
    end
end

 canny_row=zeros(pan);
 canny_col=zeros(leb);
edge_final = uint8(T_res.*255);
%disp(T_res);
f=0;
for i=1:pan
    for j=1:leb
        if(T_res(i,j)==1)
            f=f+1;
            canny_row(f)=i;
            canny_col(f)=j;
        end
    end
end
figure, imshow(edge_final);
```

OUTCOME:

A sequence of points that determines the edge pixels of an image.

<u>AIM 2</u>: MESSAGE EMBEDDING

TASK 1: EMBEDDING TECHNIQUE ADVANCEMENT

TECHNIQUES USED:

- *<u>Spatial Domain Methods:</u>*
  - *Least Significant Bit (LSB):* LSB is simple to use but susceptible to lossy compression and image manipulations. Some bits are changed directly in the image pixel values in hiding the data. Changes in the value of the LSB are imperceptible for human eyes.
  - *Pixel Value Differencing (PVD):* To embed the data in PVD two consecutive pixels are selected. Whether the pixels to be determined are from smooth area or an edge area. Payload is determined by calculating the difference between two regular pixels.
  - *Binary Pattern complexity (BPC):* The Binary Pattern complexity approach is used to measure the noise factor in the image complexity. The noisy portion is replaced by binary Pattern and it is mapped from the secret data.

- *<u>Vector Embedding:</u>* Vector embedding is a message embedding method that uses robust algorithm with codec standard (MPEG-1 and MPEG -2). This method embeds audio information to pixels of frames in host video.

- *<u>Distortion Techniques:</u>* The distortion method is used to store the secret data by distorting the signal. An encoder applies a sequence of modifications to the cover image and the decoder phase decodes the encrypted data to the original data with the secret data by using some secret key.

- *<u>Statistical Technique:</u>* In this technique, message is embedded by changing several properties of the cover. It involves the splitting of cover into blocks and then embedding one message bit in each block.

<u>Message embedding algorithm:</u>

*Step 1***:** Extracting the boundary pixels, corner pixels and the edge pixels of the cover image.

*Step 2***:** Extracting the characters of the encrypted text.

*Step 3***:** Calculating the length of the of the encrypted text.

Step 4: Converting the length to binary and calculating its number of bits, let it be ‘b’.

Step 5: Getting ‘b’ number of random values that will determine the corner points to be used.

*Step 6***:** Embedding the message length in binary form in those corner pixels using the LSB technique.

Step 7: Getting ‘b+(8*length of encrypted text)’ number of random values that will determine the corner points to use where the ‘b’ number of corner points will be stored along with the ‘(8*length of encrypted text)’ random values to store the edge points where each character of the encrypted text in binary form using LSB technique.

Step 8: Getting the edge points from the ‘(8*length of encrypted text)’ random values to store each character of the encrypted text in binary form using LSB technique.

Interface Demonstration and Background Coding:

```
%%Embedding the message in the cover image
z=size(harris_col);
s='howareyouiamfine';
l=strlength(s);
tot=l*8;
btot=dec2bin(tot);
btot_len=length(btot);
%Generating unique random keys
key=zeros(btot_len);
i=1;
    while(i<=btot_len)
        key(i)=randi(z(1));
        for k=1:i-1
            if(key(i)==key(k))
                i=i-1;
                break;
            end
        end
        i=i+1;
    end
key=sort(key);
%disp(btot_len);
% for i=1:btot_len
%     fprintf('Key%d: %d\n',i,key(i));
% end
%hps=mat2gray(harris_points);
%figure,imshow(hps);
for i=1:btot_len
    %disp(btot(i));
    ind=key(i);
    if(~(harris_row(i)==boundary(j,1) && harris_col(i)==boundary(j,2)))
        hp=dec2bin(pic(harris_row(ind),harris_col(ind)));
        lhp=length(hp);
        hp(lhp)=btot(i);
        pic(harris_row(ind),harris_col(ind))=bin2dec(hp);
    end
end
hps=mat2gray(pic);
figure,imshow(hps);
keyb=randi(z(1),btot_len,1);
b=dec2bin(btot_len);
lb=length(b);
for i=1:lb
    p=pic(boundary(i,1),boundary(i,2));
    bp=dec2bin(p);
    lbp=length(bp);
    bp(lbp)=b(i);
    pic(boundary(i,1),boundary(i,2))=bin2dec(bp);
end
%pic(boundary(1,1),boundary(1,2))=btot_len;
%disp(boundary_image(boundary(1,1),boundary(1,2)));
j=0;
i=lb+1;
while(j<btot_len)
    j=j+1;
    ind=key(j);
    disp(j);
    bhr=dec2bin(harris_row(ind),8);
```

```
        for jj=1:8
            h=pic(boundary(i,1),boundary(i,2));
            bh=dec2bin(h);
            lbh=length(bh);
            bh(lbh)=bhr(jj);
            pic(boundary(i,1),boundary(i,2))=bin2dec(bh);
            i=i+1;
        end
        bhc=dec2bin(harris_col(ind),8);
        for jj=1:8
            h=pic(boundary(i,1),boundary(i,2));
            bh=dec2bin(h);
            lbh=length(bh);
            bh(lbh)=bhr(jj);
            pic(boundary(i,1),boundary(i,2))=bin2dec(bh);
            i=i+1;
        end
    end
    z=size(canny_col);
    %Generating unique random edges
    edge=zeros(tot);
    i=1;
        while(i<=tot)
            edge(i)=randi(z(1));
            if(edge(i)==0)
                i=i-1;
            else
                for k=1:i-1
                    if(edge(i)==edge(k))
                        i=i-1;
                        break;
                    end
                end
            end
            i=i+1;
        end
    edge=sort(edge);
    % for i=1:tot
    %     fprintf('Edge%d: %d\n',i,edge(i));
    % end
    i=0;
    ii=1;
    while(ii<=l)
        ch=s(ii);
        bch=dec2bin(ch,8);
        for j=1:8
            i=i+1;
            %disp(i);
            e=edge(i);
            er=canny_row(e);
            ec=canny_col(e);
            bT_res=dec2bin(pic(er,ec));
            lbt=length(bT_res);
            bT_res(lbt)=bch(j);
            pic(er,ec)=bin2dec(bT_res);
        end
        ii=ii+1;
    end
    figure,imshow(pic);
```

OUTCOME: The encrypted message successfully embedded in the cover image.

AIM 3: MESSAGE HIDING

Data Hiding is the process of embedding information into digital content without causing perceptual degradation.

In data hiding, two famous techniques can be used. They are

- Cryptography.
- steganography

The method of storing and transmitting data in a particular form so that only those for whom it is intended can read and process it is known as Cryptography.

Steganography on the other hand hides an encrypted message so that no one suspects it exists. Steganography can embed secret data into cover media such as

- text
- image
- audio
- video.

TASK 1: EXISTING ALGO DESIGN

- *Substitution Cipher:* **Substitution Cipher** is an encryption method used to replace units of plain text with the cipher text, according to a fixed system. The "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth.

- *Transposition Cipher:* **Transposition Cipher** is an encryption method by which the positions held by units of plain text (which are commonly characters or groups of characters) are shifted according to a regular system, so that the cipher text constitutes a permutation of the plain text.

TASK 2: NEW ALGO DESIGN

Rail Fence Cipher:

Idea:

In a rail fence cipher, the order of the alphabets are rearranged to obtain the cipher-text.

- The plain-text is written downwards and diagonally on successive rails of an imaginary fence.

- When the bottom rail is reached, it traverses upwards moving diagonally. After reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zigzag manner.

- After each alphabet has been written, the individual rows are are combined to obtain the cipher-text.

Code for Rail Fence Cipher:

```
function [ cipher_text ] = Rail_Fence( plain_text )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
%prompt='Enter number of rails: ';
%num_rails=input(prompt);
len=length(plain_text);
num_rails=randi(floor(len/2));
disp(num_rails);
```

```
k=1;
i=1;
cipher_text=blanks(len);
while(i<=num_rails)
    if(i==1 || i==num_rails)
        j=i;
        while( j<=len)
            cipher_text(k)=plain_text(j);
            k=k+1;
            j=j+(2*(num_rails-1));
        end
    else
        cipher_text(k)=plain_text(i);
        k=k+1;
        x=0;
        j=i;
        while(j<=len)
            x=x+1;
            if(mod(x,2)==1)
                j=j+2*(num_rails-i);
            else
                j=j+i;
            end
            if(j<=len)
                cipher_text(k)=plain_text(j);
                k=k+1;
            end
            j=j+1;
        end
    end
    i=i+1;
end
end
```

Spiral Matrix:

Idea:

Printing a matrix in spiral order is just a way to traverse the matrix. The matrix can be represented by a 2-D array. A string is given as input and the output received will be a matrix where the characters are input in spiral order.

Code for Spiral Matrix Encryption:

```
function [ st ] = Spiral_Encryp( s )
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
str='';
l=length(s);
a=zeros(100);
b=zeros(100);
for i=1:l
    if(s(i:i)~=' ')
        str=strcat(str,s(i:i));
    end
end
l=length(str);
i=2;
f=0;
while(i<=l/2)
```

```
        if(mod(l,i)~=0)
            f=1;
            break;
        end
        i=i+1;
    end
    if(f==0)
        str=strcat(str,'.');
        l=length(str);
    end
    j=1;
    i=2;
    for i=2:l/2
        if(mod(l,i)==0)
            a(j)=i;
            b(j)=l/i;
            j=j+1;
        end
    end
    min=9999;
    p=0;
    for i=1:j-1
        if((a(i)-b(i))==0)
            min=a(i)-b(i);
            p=i;
            break;
        elseif((a(i)-b(i))<min)
            min=abs(a(i)-b(i));
            p=i;
        end
    end
    r=a(p);
    c=b(p);
    ar=repmat('a',r,c);
    k = 1;
    l = 1;
    t=1;
    m=r;
    n=c;
    st='';
    while (k <= r && l <= c)
        for i=l:c
            ar(k,i)=str(t:t);
            t=t+1;
        end
        k=k+1;
        for i=k:r
            ar(i,c)=str(t:t);
            t=t+1;
        end
        c=c-1;
        if(k<r)
            for i=c:-1:l
                ar(r,i)=str(t:t);
                disp(ar(r,i));
                t=t+1;
            end
            r=r-1;
        end
         if(l<c)
            for i=r:-1:k
```

```
                ar(i,l)=str(t:t);
                t=t+1;
            end
            l=l+1;
         end
end
for i=1:n
    for j=1:m
            st=strcat(st,ar(j,i));
    end
end
st=strcat(int2str(m),';',int2str(n),';',st);
end
```

Output:

```
Enter the message:JHILIKORCHIRIYAROY

J H I L I K
Y A R O Y O
I R I H C R
```

Encryption using LSB replacement:

Idea:

A cover text is taken as input by the user and both the cover text and the message are converted to their respective values in binary. Each bit of the message is then encrypted within the LSB of a character of the cover text and the total number of bits in the message is stored within the first character of the cover text.

Code for LSB Replacement Encryption:

```
function [ c ] = Encryp2( n,s )
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here
% prompt='Enter a message ';
% n=input(prompt);
% prompt='Enter the cover text ';
% s=input(prompt,'s');
%s='HOWAREYOUIAMFINEWHOAREYOU';
%n=str2double(n1);
disp(n);
l=strlength(s);
b1=string(l);
ctr=0;
c=blanks(l);
as=zeros(l);
b=dec2bin(n);
ct=length(b);
if(l<4*ct)
    %var=4*ct;
   msgbox(sprintf('The length of the cover text must be at least
%d',4*ct));
else
%     for i=1:l
```

```
%         b1(i)="";
%     end
    for i=1:l
        c(i)=s(i:i);
        as(i)=double(c(i));
        b1(i)=dec2bin(as(i));
        ctr=strlength(b1(i));
        for j=1:8-ctr
            b1(j)=strcat("0",b1(j));
        end
    end
    rand1=zeros(ct);
    for i=1:ct
        rand1(i)=randi(l-ct);
        for k=1:i
            if(rand1(i)==rand1(k))
                i=i-1;
                break;
            end
        end
    end
    rand2=sort(rand1);
    ct1=ct;
    rnd=string(ct);
    b2=dec2bin(ct1);
    for j=1:8-ctr
            b2=strcat("0",b2);
    end
    for i=1:ct
        r=rand2(i);
        rnd(i)=dec2bin(r);
        for j=1:8-ctr
            rnd(j)=strcat("0",rnd(j));
        end
    end
    b1(1)=b2;
    for i=2:ct+1
        b1(i)=rnd(i-1);
    end
    for tp=1:ct
        b1(ct+rand2(tp))=strcat(extractAfter(b1(ct+rand2(tp)),1),b(tp:tp));
    end
    for i=1:l
        as(i)=bin2dec(b1(i));
        if(i<=ct+1)
            as(i)=as(i)+65;
        end
        c(i)=char(as(i));
    end
    fprintf('The Encrypted Message is: ');
    for i=1:l
        fprintf('%c',c(i));
    end
    msgbox('Message Successfully Encrypted!!!');
end
```

GUI Interface:

Interface Page 1:

The first page of the GUI Interface. In this page, an encryption technique is randomly chosen and a message taken as input from the user is encrypted using the chosen encryption technique. It consists of different working parts as follows.

Spin Wheel:

The spin wheel consists of several radio buttons and a pushbutton. Each radio button corresponds to an encryption technique. The encryption techniques chosen are given below.

Radio buttons:

Each radio button has a different encryption technique. They are as follows:

- Spiral Encryption Technique
- Rail Fence Encryption Technique
- LSB Number Encryption Technique

The Josephus Problem is implemented to select one of the encryption techniques based on a random number. The last remaining button or the 'winner' button with its corresponding encryption technique is used to encrypt the message for added security.

Josephus Problem:

In computer science and mathematics, the **Josephus problem** (or **Josephus permutation**) is a theoretical problem related to a certain counting-out game.

People stand in a circle waiting to be executed. Counting starts at a specified point in the circle and proceeds around the circle in a specified direction. A person is executed after a specified number of people are skipped. The procedure repeats with the remaining people, starting with the next person, going in the same direction and skipping the same number of people, until only one person remains, and is freed.

The problem — given the number of people, starting point, direction, and number to be skipped — is to choose the position in the initial circle to avoid execution.

Pushbutton:

The pushbutton "Spin The Wheel" contains the code to choose a radio button containing an encryption technique. The Josephus Permutation runs here in the background to choose a radio button from a random number. The chosen radio button and its corresponding encryption technique is displayed in a textbox. Once it has been selected, a textbox is enabled where the user can input the message they want to send. If the encryption using LSB replacement is chosen, another textbox is also enabled where the user has to input a cover text.

Message Encryption:

When the "Encrypt the Message" push button is clicked, the message taken as input from the user is encrypted according to the technique selected. The encrypted text is then shown in a new textbox.

Message Embedding:

The "Embed the message" push button opens the next page in the GUI interface and sends the relevant information (such as the embedded message, embedding technique, etc) to it. Once all the data is sent, the current page is closed.

Interface Page 2:

The second page of the interface is where the user uploads an image from their device and embeds the encrypted text with the image. It also consists of several features.

Image Browsing:

The user uploads the image they want with the "Browse an Image" pushbutton. It lets the user choose an image of the required format from their device and upload it to the interface. The selected image is then displayed within the axes.

Embedding Message Within the Image:

The information of the selected image (number of boundary, key point and edge pixels) are gathered when the user clicks on the "Encrypt" pushbutton. The information is then stored in the following parts of the image:

The message is embedded within the edge pixels of the cover image by LSB modification.

The length of the message stored is encrypted in the Key/Corner points of the image.

The boundary pixels stores the location of the key point and edge pixels containing the relevant information.

The modified image is then sent to the concerned party.

Decrypting the Message From the Image:

The image containing the secret message is received and stored in an appropriate location. It is smoothed with the help of filters in order to remove the noise from the image. The boundary pixels are then read to find the location of the key point pixels containing the length of the image. The length is then used to find the number of pixels used to store the message and their locations and subsequently the encrypted message. The message is then decrypted by reversing the encryption technique used to receive the original message.

<u>Sender's Part:</u>

In the sender's part, the sender has to click on the **'Spin The Wheel'** button to get an encryption technique to use. Then this encryption technique will be shown in a text box. Based on the encryption technique chosen, the message entered will be encrypted on clicking the **'Encrypt The Message'** button. Then in order to embed the encrypted text in an image the sender has to click on the **'Embed The Message'** button which will take the sender to the next page where the embedding will be done. On opening the second page he encrypted technique chosen by the spin wheel will be shown on the top of the page. Then the sender has to browse an image using the button **'Browse an Image'.** Browsing an image, the image chosen will be shown on the page. The sender then has to click on the **'Hide The Message'** which will embed the encrypted text in the image. Following this the sender has to click on the **'Decrypt'** button to send the stego image to the receiver.

<u>Spin The Wheel:</u>

Code:

```
sz=9;
n=randi(50);
a=zeros(sz);
for i=1:sz
    a(i)=i;
end
i=1;
while(1)
    i=i+n;
    while(i>sz)
        i=i-sz;
    end
    if(i~=sz)
        for j=i+1:sz
            a(j-1)=a(j);
        end
    end
    i=i-1;
    sz=sz-1;
    if(sz==1)
        break;
    end
end
p=a(1);
if(mod((p-1),3)==0)
    set(handles.text6,'String','Spiral Encryption Technique');
    set(handles.spiral,'enable','on');
    set(handles.encryp2,'enable','off');
    set(handles.rail,'enable','off');
    set(handles.spiral, 'Value', 1);
    set(handles.msgtxt,'enable','on');
    set(handles.text9,'enable','on');
    set(handles.text10,'enable','off');
    set(handles.cvrtxt,'enable','off');
elseif(mod((p-2),3)==0)
    set(handles.text6,'String','Rail Fence Encryption Technique');
    set(handles.spiral,'enable','off');
    set(handles.encryp2,'enable','off');
    set(handles.rail,'enable','on');
    set(handles.rail, 'Value', 1);
    set(handles.msgtxt,'enable','on');
    set(handles.msgtxt,'enable','on');
```

```
    set(handles.text9,'enable','on');
    set(handles.text10,'enable','off');
    set(handles.cvrtxt,'enable','off');
else
    set(handles.text6,'String','LSB Encryption Technique');
    set(handles.spiral,'enable','off');
    set(handles.rail,'enable','off');
    set(handles.encryp2,'enable','on');
    set(handles.encryp2, 'Value', 1);
    set(handles.msgtxt,'enable','on');
    set(handles.msgtxt,'enable','on');
    set(handles.text9,'enable','on');
    set(handles.text10,'enable','on');
    set(handles.cvrtxt,'enable','on');
end
```

Encrypt The Message:

Code:

```
u=get((get(handles.uibuttongroup2,'SelectedObject')),'Tag');
if(strcmp(u,'spiral'))
    s=get(handles.msgtxt,'String');
    st=Spiral_Encryp(s);
    set(handles.text12,'String',st);
    msgbox('Message Successfully Encrypted!!!');
elseif(strcmp(u,'rail'))
    plain_text=get(handles.msgtxt,'String');
    cipher_text=Rail_Fence(plain_text);
    set(handles.text12,'String',cipher_text);
    msgbox('Message Successfully Encrypted!!!');
else
    n=get(handles.msgtxt,'String');
    s=get(handles.cvrtxt,'String');
    c=Encryp2(n,s);
    set(handles.text12,'String',c);
end
```

Embed The Message:

Code:

```
a=get(handles.text6,'String');
setappdata(0,'spin',a);
b=get(handles.text12,'String');
setappdata(0,'encryp',b);
Encryption();
close(untitled);
```

Browse an Image:

Code:

```
[filename pathname]=uigetfile('*','Choose an Image');
img1=imread([pathname,filename]);
axes(handles.axes1);
imshow(img1);
setappdata(0,'image',img1);
```

Hide The Message:

Code:

```
%% Boundary Tracing
fprintf('Boundary Tracing\n===================\n');
input=getappdata(0,'image');
pic=input;
input=rgb2gray(input);
```

```
s=size(input);
a=s(1)*2+s(2)*2-4;
boundary=zeros(a,2);
c=0;
for i=1:s(2)
    c=c+1;
    boundary(c,1)=1;
    boundary(c,2)=i;
end
for i=2:s(1)
    c=c+1;
    boundary(c,1)=i;
    boundary(c,2)=s(2);
end
for i=s(2)-1:-1:1
    c=c+1;
    boundary(c,1)=s(1);
    boundary(c,2)=i;
end
for i=s(1)-1:-1:2
    c=c+1;
    boundary(c,1)=i;
    boundary(c,2)=1;
end

%% Edge Detection
fprintf('Canny\n===========\n');
img=pic;
img = rgb2gray(img);
img = double (img);

%Value for Thresholding
T_Low = 0.075;
T_High = 0.175;

%Gaussian Filter Coefficient
B = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9, 12, 9, 4;2, 4, 5,
4, 2 ];
B = 1/159.* B;

%Convolution of image by Gaussian Coefficient
A=conv2(img, B, 'same');

%Filter for horizontal and vertical direction
KGx = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
KGy = [1, 2, 1; 0, 0, 0; -1, -2, -1];

%Convolution by image by horizontal and vertical filter
Filtered_X = conv2(A, KGx, 'same');
Filtered_Y = conv2(A, KGy, 'same');

%Calculate directions/orientations
arah = atan2 (Filtered_Y, Filtered_X);
arah = arah*180/pi;

pan=size(A,1);
leb=size(A,2);

%Adjustment for negative directions, making all directions positive
for i=1:pan
    for j=1:leb
```

```
        if (arah(i,j)<0)
            arah(i,j)=360+arah(i,j);
        end
    end
end

arah2=zeros(pan, leb);

%Adjusting directions to nearest 0, 45, 90, or 135 degree
for i = 1  : pan
    for j = 1 : leb
        if ((arah(i, j) >= 0 ) && (arah(i, j) < 22.5) || (arah(i, j) >=
157.5) && (arah(i, j) < 202.5) || (arah(i, j) >= 337.5) && (arah(i, j) <=
360))
            arah2(i, j) = 0;
        elseif ((arah(i, j) >= 22.5) && (arah(i, j) < 67.5) || (arah(i, j)
>= 202.5) && (arah(i, j) < 247.5))
            arah2(i, j) = 45;
        elseif ((arah(i, j) >= 67.5 && arah(i, j) < 112.5) || (arah(i, j)
>= 247.5 && arah(i, j) < 292.5))
            arah2(i, j) = 90;
        elseif ((arah(i, j) >= 112.5 && arah(i, j) < 157.5) || (arah(i, j)
>= 292.5 && arah(i, j) < 337.5))
            arah2(i, j) = 135;
        end
    end
end

%Calculate magnitude
magnitude = (Filtered_X.^2) + (Filtered_Y.^2);
magnitude2 = sqrt(magnitude);

BW = zeros (pan, leb);

%Non-Maximum Supression
for i=2:pan-1
    for j=2:leb-1
        if (arah2(i,j)==0)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i,j+1), magnitude2(i,j-1)]));
        elseif (arah2(i,j)==45)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j-1), magnitude2(i-1,j+1)]));
        elseif (arah2(i,j)==90)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j), magnitude2(i-1,j)]));
        elseif (arah2(i,j)==135)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j+1), magnitude2(i-1,j-1)]));
        end
    end
end

BW = BW.*magnitude2;

%Hysteresis Thresholding
T_Low = T_Low * max(max(BW));
T_High = T_High * max(max(BW));

T_res = zeros (pan, leb);
```

```
for i = 1  : pan
    for j = 1 : leb
        if (BW(i, j) < T_Low)
            T_res(i, j) = 0;
        elseif (BW(i, j) > T_High)
            T_res(i, j) = 1;
        %Using 8-connected components
        elseif ( BW(i+1,j)>T_High || BW(i-1,j)>T_High || BW(i,j+1)>T_High
|| BW(i,j-1)>T_High || BW(i-1, j-1)>T_High || BW(i-1, j+1)>T_High ||
BW(i+1, j+1)>T_High || BW(i+1, j-1)>T_High)
            T_res(i,j) = 1;
        end
    end
end

 canny_row=zeros(pan);
 canny_col=zeros(leb);
edge_final = uint8(T_res.*255);

f=0;
for i=1:pan
    for j=1:leb
        if(T_res(i,j)==1)
            f=f+1;
            canny_row(f)=i;
            canny_col(f)=j;
        end
    end
end

%% Key Point Detection
fprintf('Harris\n===========\n');
img=pic;
 im=img;
im=im(:,:,1);

sigma=1;
radius=1;
order=2*radius+1;
threshold=7000;
%derivatives in x and y direction
[dx,dy]=meshgrid(-1:1,-1:1);

Ix=conv2(double(im),dx,'same');
Iy=conv2(double(im),dy,'same');

% Implementing the Gaussian filter

dim=max(1,fix(6*sigma));
m=dim;
n=dim;
[h1,h2] = meshgrid(-(m-1)/2:(m-1)/2 , -(n-1)/2: (n-1)/2);
hg=exp(-(h1.^2+h2.^2)/(2*sigma^2));
[a b]=size(hg);
sum1=0;
for i=1:a
    for j=1:b
        sum1=sum1+hg(i,j);
    end
end
g=hg ./sum1;
```

```
%Calculate the entries of the M-matrix

Ix2=conv2(double(Ix.^2),g,'same');
Iy2=conv2(double(Iy.^2),g,'same');
Ixy=conv2(double(Ix.*Iy),g,'same');

%Harris measure
R=(Ix2.*Iy2 - Ixy.^2) ./ (Ix2+Iy2 + eps);

%Find local maxima
mx=ordfilt2(R,order^2,ones(order));

harris_points=(R==mx) & (R>threshold);
 [harris_row,harris_col]=find(harris_points);


%% Embedding the message in the cover image
z=size(harris_col);
s=getappdata(0,'encryp');
disp(s);
l=strlength(s);
tot=l*8;
btot=dec2bin(tot,8);
btot_len=length(btot);
%Generating unique random keys
key=zeros(btot_len);
i=1;
    while(i<=btot_len)
        key(i)=randi(z(1));
        for k=1:i-1
            if(key(i)==key(k))
                i=i-1;
                break;
            end
        end
        i=i+1;
    end
key=sort(key);
for i=1:btot_len
    ind=key(i);
    if(~(harris_row(i)==boundary(j,1) && harris_col(i)==boundary(j,2)))
        hp=dec2bin(pic(harris_row(ind),harris_col(ind)));
        lhp=length(hp);
        hp(lhp)=btot(i);
        pic(harris_row(ind),harris_col(ind))=bin2dec(hp);
    end
end
hps=mat2gray(pic);
keyb=randi(z(1),btot_len,1);
b=dec2bin(btot_len,8);
lb=length(b);
for i=1:lb
    p=pic(boundary(i,1),boundary(i,2));
    bp=dec2bin(p);
    lbp=length(bp);
    bp(lbp)=b(i);
    pic(boundary(i,1),boundary(i,2))=bin2dec(bp);
end
j=0;
i=lb+1;
while(j<btot_len)
```

```
    j=j+1;
    ind=key(j);

    bhr=dec2bin(harris_row(ind),9);
    for jj=1:9
        h=pic(boundary(i,1),boundary(i,2));
        bh=dec2bin(h);
        lbh=length(bh);
        bh(lbh)=bhr(jj);
        pic(boundary(i,1),boundary(i,2))=bin2dec(bh);
        i=i+1;
    end
    bhc=dec2bin(harris_col(ind),9);
    for jj=1:9
        h=pic(boundary(i,1),boundary(i,2));
        bh=dec2bin(h);
        lbh=length(bh);
        bh(lbh)=bhc(jj);
        pic(boundary(i,1),boundary(i,2))=bin2dec(bh);
        i=i+1;
    end
end
ii=i;
z=size(canny_col);
%Generating unique random edges
edge=zeros(tot);
i=1;
    while(i<=tot)
        edge(i)=randi(z(1));
        if(edge(i)==0)
            i=i-1;
        else
            for k=1:i-1
                if(edge(i)==edge(k))
                    i=i-1;
                    break;
                end
            end
        end
        i=i+1;
    end
edge=sort(edge);

%Storing the random ege points in boundary pixels
j=0;
i=ii;
ss=size(pic);
while(j<tot)
    j=j+1;
    ind=edge(j);
    bhr=dec2bin(canny_row(ind),9);

    for jj=1:9


            h=pic(boundary(i,1),boundary(i,2));
            bh=dec2bin(h);
            lbh=length(bh);
            bh(lbh)=bhr(jj);
            pic(boundary(i,1),boundary(i,2))=bin2dec(bh);
```

```
            i=i+1;
        end
        bhc=dec2bin(canny_col(ind),9);
        for jj=1:9
                h=pic(boundary(i,1),boundary(i,2));
                bh=dec2bin(h);
                lbh=length(bh);
                bh(lbh)=bhc(jj);
                pic(boundary(i,1),boundary(i,2))=bin2dec(bh);

            i=i+1;
        end
    end
    i=0;
    ii=1;
    while(ii<=l)
        ch=s(ii);
        bch=dec2bin(ch,8);
        for j=1:8
            i=i+1;
            %disp(i);
            e=edge(i);
            er=canny_row(e);
            ec=canny_col(e);
            bT_res=dec2bin(pic(er,ec));
            lbt=length(bT_res);
            bT_res(lbt)=bch(j);
            pic(er,ec)=bin2dec(bT_res);
        end
        ii=ii+1;
    end
    axes(handles.axes2);
    imshow(pic);
    setappdata(0,'image1',pic);
```

Decrypt:

Code:

```
spin1=getappdata(0,'spin');
setappdata(0,'spin2',spin1);
Decryption();
close(Encryption);
```

Receiver's Part:

In the receiver's part, on opening he receiver's page the encrypted technique chosen by the spin wheel will be shown on the top of the page. The receive will receive the stego image by clicking on the **'Receive'** button. The stego image will then be shown on the page. After that, the receiver has to click on the **'Retrieve'** button to get the encrypted message. The encrypted message will then be shown in the text box. Then there is a **'Decrypt'** button clicking on which will decrypt the message and show it in the text box.

Receive:

Code:

```
img1=getappdata(0,'image1');
axes(handles.axes1);
imshow(img1);
```

Retrieve:

Code:

```
%% Boundary Tracing
fprintf('Boundary Tracing\n====================\n');
input=getappdata(0,'image1');
pic=input;
input=rgb2gray(input);
s=size(input);
a=s(1)*2+s(2)*2-4;
boundary=zeros(a,2);
c=0;
for i=1:s(2)
    c=c+1;
    boundary(c,1)=1;
    boundary(c,2)=i;
end
for i=2:s(1)
    c=c+1;
    boundary(c,1)=i;
    boundary(c,2)=s(2);
end
for i=s(2)-1:-1:1
    c=c+1;
    boundary(c,1)=s(1);
    boundary(c,2)=i;
end
for i=s(1)-1:-1:2
    c=c+1;
    boundary(c,1)=i;
    boundary(c,2)=1;
end
fprintf('Canny\n===========\n');

img=pic;
img = rgb2gray(img);
img = double (img);

%Value for Thresholding
T_Low = 0.075;
T_High = 0.175;

%Gaussian Filter Coefficient
B = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9, 12, 9, 4;2, 4, 5,
4, 2 ];
B = 1/159.* B;
```

```
%Convolution of image by Gaussian Coefficient
A=conv2(img, B, 'same');

%Filter for horizontal and vertical direction
KGx = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
KGy = [1, 2, 1; 0, 0, 0; -1, -2, -1];

%Convolution by image by horizontal and vertical filter
Filtered_X = conv2(A, KGx, 'same');
Filtered_Y = conv2(A, KGy, 'same');

%Calculate directions/orientations
arah = atan2 (Filtered_Y, Filtered_X);
arah = arah*180/pi;

pan=size(A,1);
leb=size(A,2);

%Adjustment for negative directions, making all directions positive
for i=1:pan
    for j=1:leb
        if (arah(i,j)<0)
            arah(i,j)=360+arah(i,j);
        end
    end
end

arah2=zeros(pan, leb);

%Adjusting directions to nearest 0, 45, 90, or 135 degree
for i = 1  : pan
    for j = 1 : leb
        if ((arah(i, j) >= 0 ) && (arah(i, j) < 22.5) || (arah(i, j) >=
157.5) && (arah(i, j) < 202.5) || (arah(i, j) >= 337.5) && (arah(i, j) <=
360))
            arah2(i, j) = 0;
        elseif ((arah(i, j) >= 22.5) && (arah(i, j) < 67.5) || (arah(i, j)
>= 202.5) && (arah(i, j) < 247.5))
            arah2(i, j) = 45;
        elseif ((arah(i, j) >= 67.5 && arah(i, j) < 112.5) || (arah(i, j)
>= 247.5 && arah(i, j) < 292.5))
            arah2(i, j) = 90;
        elseif ((arah(i, j) >= 112.5 && arah(i, j) < 157.5) || (arah(i, j)
>= 292.5 && arah(i, j) < 337.5))
            arah2(i, j) = 135;
        end
    end
end


%Calculate magnitude
magnitude = (Filtered_X.^2) + (Filtered_Y.^2);
magnitude2 = sqrt(magnitude);

BW = zeros (pan, leb);

%Non-Maximum Supression
for i=2:pan-1
    for j=2:leb-1
        if (arah2(i,j)==0)
```

```
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i,j+1), magnitude2(i,j-1)]));
        elseif (arah2(i,j)==45)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j-1), magnitude2(i-1,j+1)]));
        elseif (arah2(i,j)==90)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j), magnitude2(i-1,j)]));
        elseif (arah2(i,j)==135)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j+1), magnitude2(i-1,j-1)]));
        end
    end
end

BW = BW.*magnitude2;

%Hysteresis Thresholding
T_Low = T_Low * max(max(BW));
T_High = T_High * max(max(BW));

T_res = zeros (pan, leb);

for i = 1  : pan
    for j = 1 : leb
        if (BW(i, j) < T_Low)
            T_res(i, j) = 0;
        elseif (BW(i, j) > T_High)
            T_res(i, j) = 1;
        %Using 8-connected components
        elseif ( BW(i+1,j)>T_High || BW(i-1,j)>T_High || BW(i,j+1)>T_High
|| BW(i,j-1)>T_High || BW(i-1, j-1)>T_High || BW(i-1, j+1)>T_High ||
BW(i+1, j+1)>T_High || BW(i+1, j-1)>T_High)
            T_res(i,j) = 1;
        end
    end
end

 canny_row=zeros(pan);
 canny_col=zeros(leb);
edge_final = uint8(T_res.*255);
f=0;
for i=1:pan
    for j=1:leb
        if(T_res(i,j)==1)
            f=f+1;
            canny_row(f)=i;
            canny_col(f)=j;
        end
    end
end


%% %%%%%%%% Decryption %%%%%%%%%%% %%

%Extracting the length of the binary of the total length
s=getappdata(0,'encryp1');
l=strlength(s);
keylen=blanks(8);
for i=1:8
    p=pic(boundary(i,1),boundary(i,2));
```

```
        bp=dec2bin(p);
        lbp=length(bp);
        keylen(i)=bp(lbp);
    end
    deckey=bin2dec(keylen);

    %Extracting the random key points
    j=0;
    i=9;
    ptr=blanks(deckey);
    ptc=blanks(deckey);
    keypts=zeros(deckey,2);
    while(j<8)
        j=j+1;
        for jj=1:9
            h=pic(boundary(i,1),boundary(i,2));
            bh=dec2bin(h);
            lbh=length(bh);
            ptr(jj)=bh(lbh);
            i=i+1;
        end
        bhr=bin2dec(ptr);
        for jj=1:9
            h=pic(boundary(i,1),boundary(i,2));
            bh=dec2bin(h);
            lbh=length(bh);
            ptc(jj)=bh(lbh);
            i=i+1;
        end
        bhc=bin2dec(ptc);
        keypts(j,1)=bhr;
        keypts(j,2)=bhc;
    end
    ii=i;
    %Extracting the total length from the key points
    btot1=blanks(deckey);
    for i=1:deckey
        p=pic(keypts(i,1),keypts(i,2));
        bp=dec2bin(p);
        lbp=length(bp);
        btot1(i)=bp(lbp);
    end
    tot1=bin2dec(btot1);

    %Extracting the random edge points
    j=0;
    i=ii;
    ptr=blanks(8);
    ptc=blanks(8);
    edgepts=zeros(tot1,2);
    while(j<tot1)
        j=j+1;
        for jj=1:9
                h=pic(boundary(i,1),boundary(i,2));
                bh=dec2bin(h);
                lbh=length(bh);
                ptr(jj)=bh(lbh);
            i=i+1;
        end

        bhr=bin2dec(ptr);
```

```
    for jj=1:9
            h=pic(boundary(i,1),boundary(i,2));
            bh=dec2bin(h);
            lbh=length(bh);
            ptc(jj)=bh(lbh);
        i=i+1;
    end
    bhc=bin2dec(ptc);
    edgepts(j,1)=bhr;
    edgepts(j,2)=bhc;
end

%Extracting the message from the edges
bch=blanks(8);
msg=blanks(l);
ii=1;
i=0;
while(ii<=l)
    for j=1:8
        i=i+1;
        er=edgepts(i,1);
        ec=edgepts(i,2);
        bT_res=dec2bin(pic(er,ec));
        lbt=length(bT_res);
        bch(j)=bT_res(lbt);
    end
    msg(ii)=bin2dec(bch);
    ii=ii+1;
end
disp(msg);
set(handles.text6,'String',msg);
```

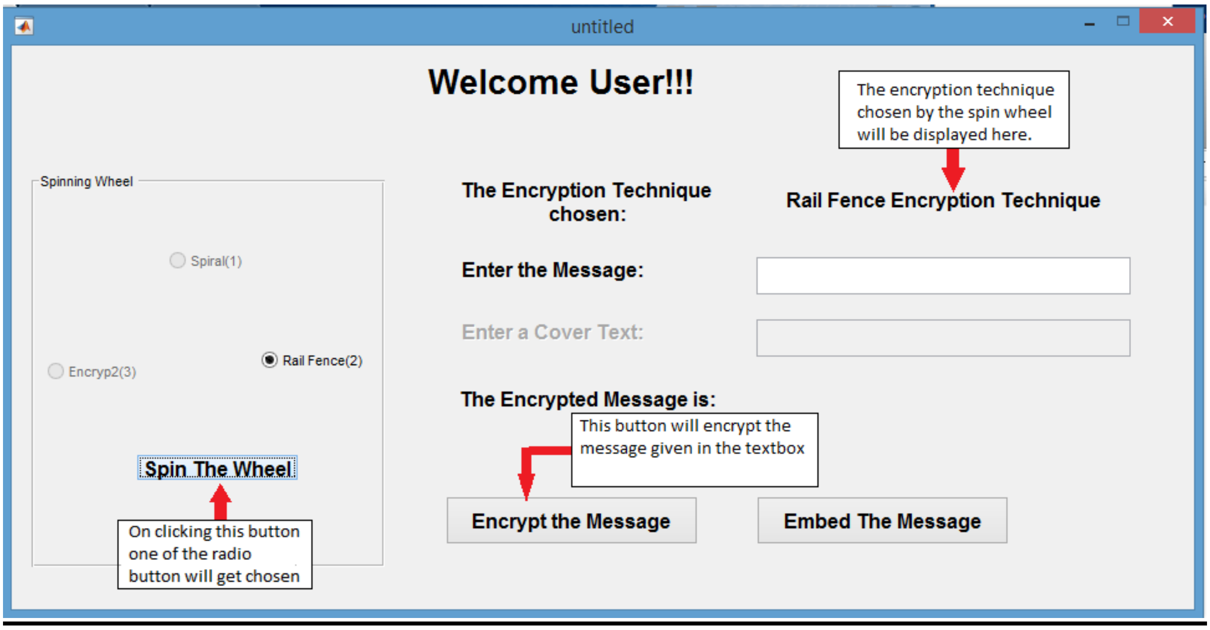
__Decrypt__:
Code:

```
msg=get(handles.text6,'String');
u=get(handles.text3,'String');
if(strcmp(u,'Spiral Encryption Technique'))
    st=Spiral_Decryp(msg);
    disp(st);
    set(handles.text6,'String',st);
    msgbox('Message Successfully Decrypted!!!');
elseif(strcmp(u,'Rail Fence Encryption Technique'))
    st=Rail_Decryp(msg);
    disp(st);
    set(handles.text6,'String',st);
    msgbox('Message Successfully Decrypted!!!');
else
    st=Decryp2(msg);
    set(handles.text3,'String',msg);
    msgbox('Message Successfully Decrypted!!!');
 end
```

Chapter 5:

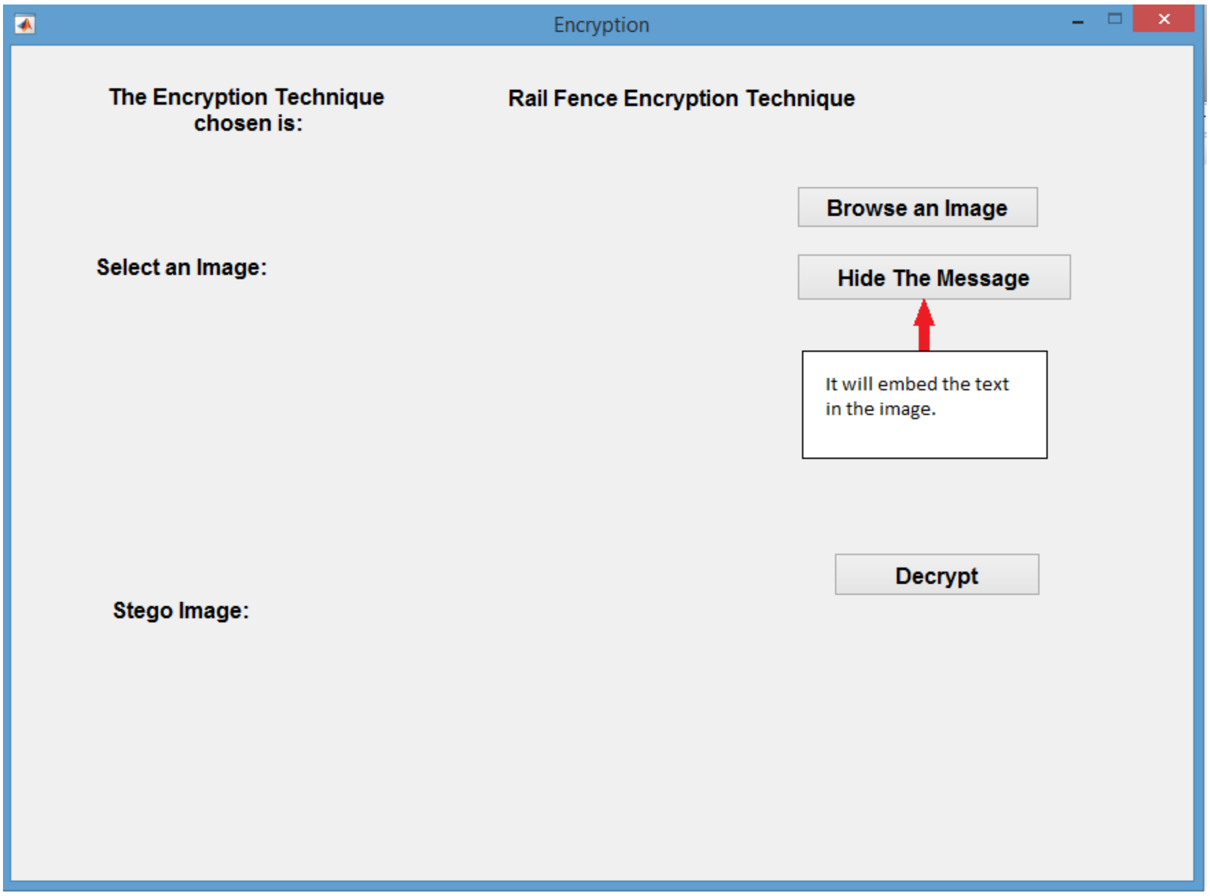
RESULT ANALYSIS

SYSTEM WORKFLOW:




untitled
Welcome User!!!
Spinning Wheel
Message Successfully Encrypted!!!
OK
Encryp2(3)
Rail Fence(2)
Spin The Wheel
The Encryption Technique chosen:
Rail Fence Encryption Technique
Enter the Message:
HowAreYou
Enter a Cover Text:
The Encrypted Message is:
HuoowYAer
This button will take the user to the embedding page.
Encrypt the Message
Embed The Message

Fig18.


Encryption
The Encryption Technique chosen is:
Rail Fence Encryption Technique
Browse an Image
Select an Image:
Hide The Message
It will embed the text in the image.
Decrypt
Stego Image:

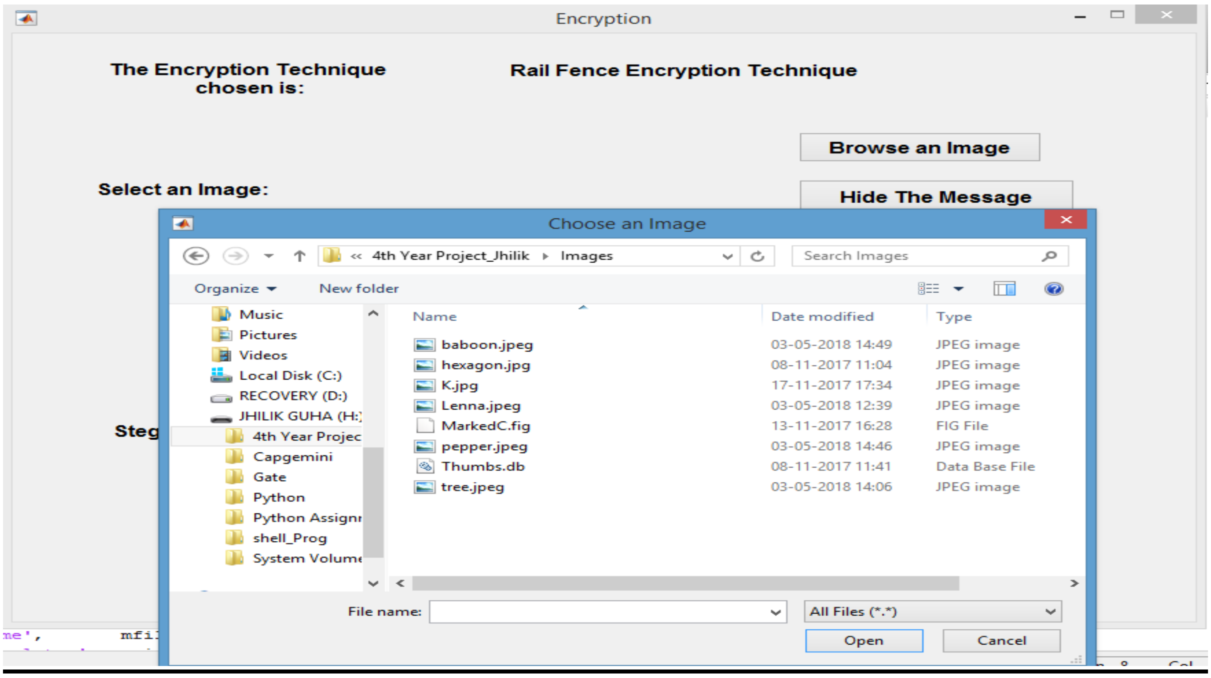

Encryption
The Encryption Technique chosen is:
Rail Fence Encryption Technique
Browse an Image
Select an Image:
Hide The Message
Choose an Image
4th Year Project_Jhilik
Images
Search Images
Organize
New folder
Music
Pictures
Videos
Local Disk (C:)
RECOVERY (D:)
Capgemini
Gate
Python
shell_Prog
Name
Date modified
Type
baboon.jpeg
03-05-2018 14:49
JPEG image
hexagon.jpg
08-11-2017 11:04
JPEG image
K.jpg
17-11-2017 17:34
JPEG image
Lenna.jpeg
03-05-2018 12:39
JPEG image
MarkedC.fig
13-11-2017 16:28
FIG File
pepper.jpeg
03-05-2018 14:46
JPEG image
Thumbs.db
08-11-2017 11:41
Data Base File
tree.jpeg
03-05-2018 14:06
JPEG image
File name:
All Files (*.*)
Open
Cancel

Fig19.

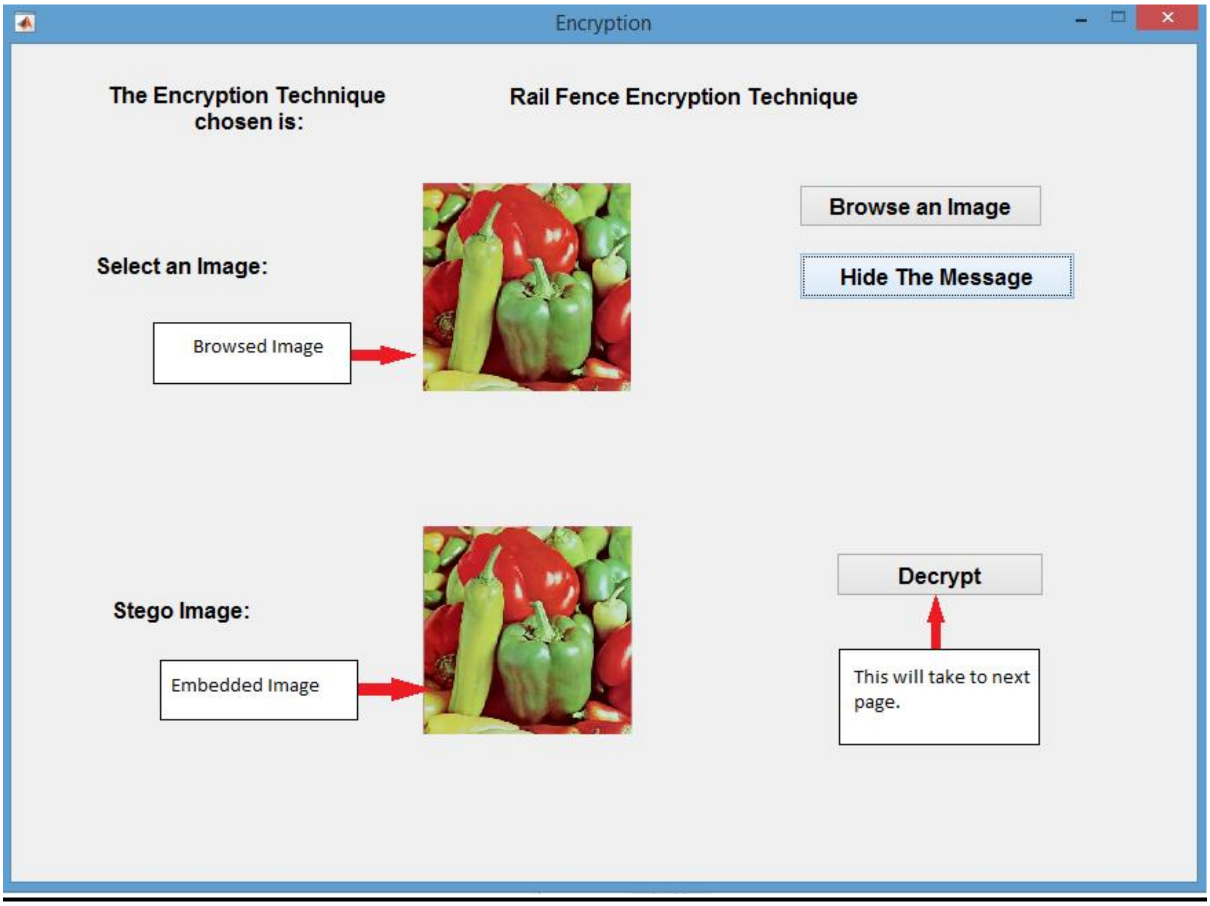

Encryption
The Encryption Technique chosen is:
Rail Fence Encryption Technique
Browse an Image
Select an Image:
Hide The Message
Browsed Image
Stego Image:
Decrypt
Embedded Image
This will take to next page.

Fig20.

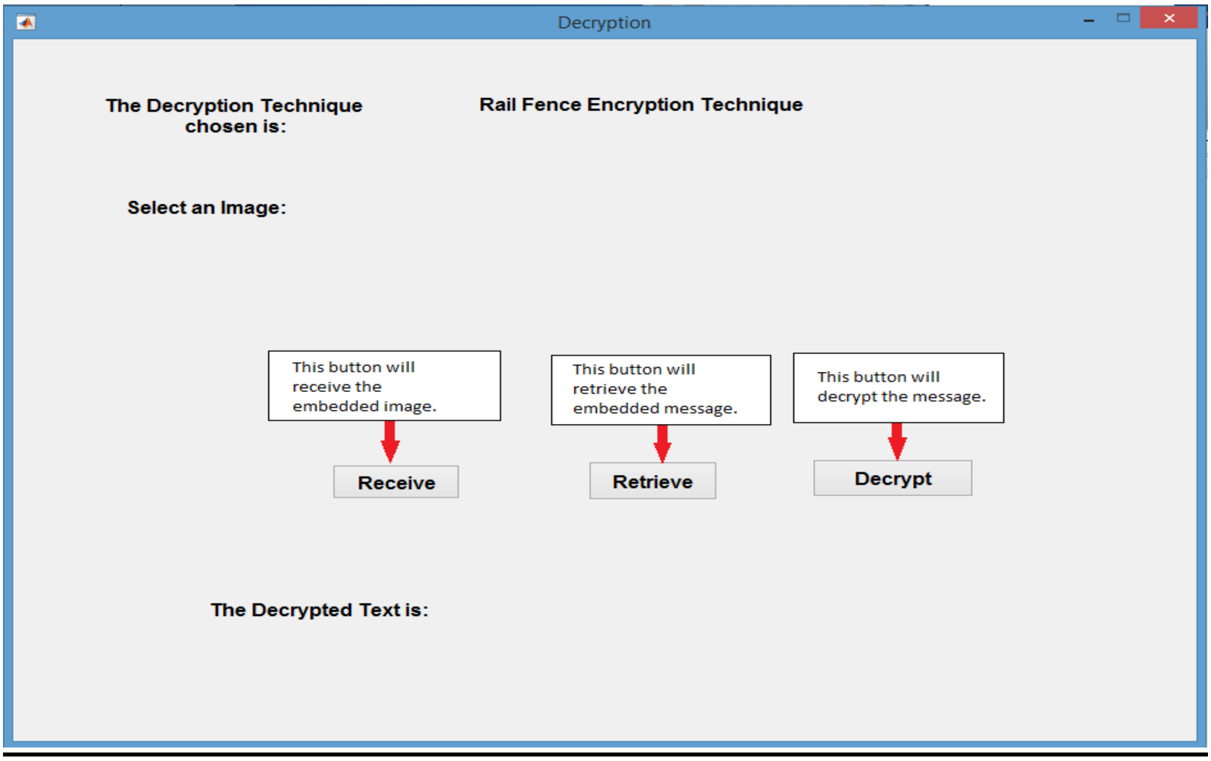
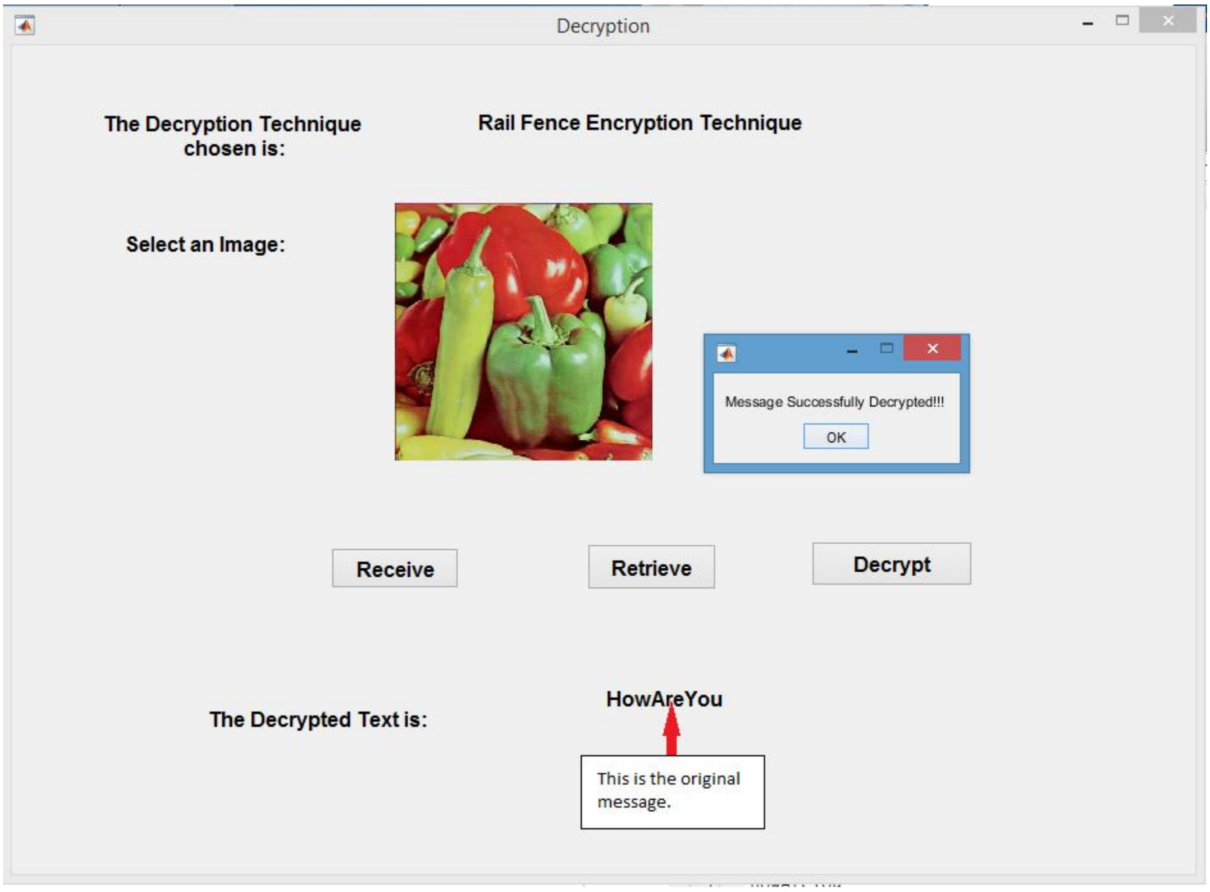

Decryption
The Decryption Technique chosen is:
Rail Fence Encryption Technique
Select an Image:
This button will receive the embedded image.
This button will retrieve the embedded message.
This button will decrypt the message.
Receive
Retrieve
Decrypt
The Decrypted Text is:

Fig21.


Decryption
The Decryption Technique chosen is:
Rail Fence Encryption Technique
Select an Image:
Message Successfully Decrypted!!!
OK
Receive
Retrieve
Decrypt
The Decrypted Text is:
HowAreYou
This is the original message.

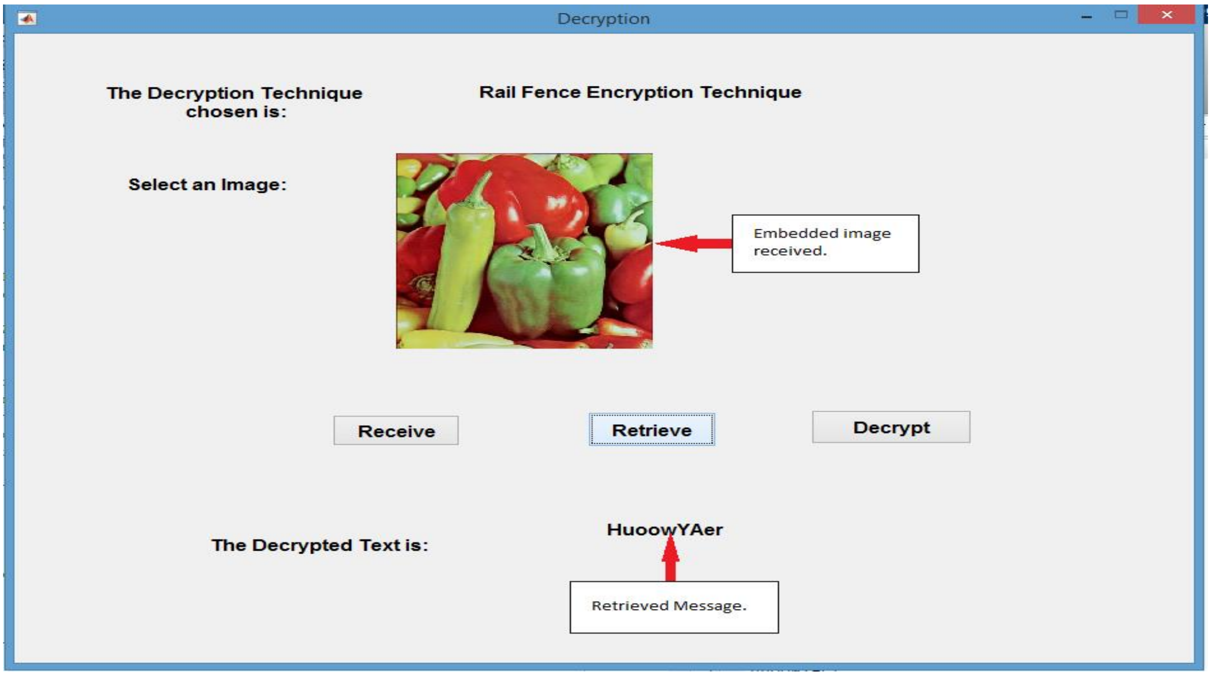
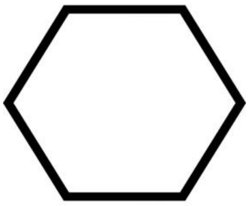

Decryption
The Decryption Technique chosen is:
Rail Fence Encryption Technique
Select an Image:
Embedded image received.
Receive
Retrieve
Decrypt
The Decrypted Text is:
HuoowYAer
Retrieved Message.

Fig22.

ANALYSIS

Sl. No.	Image	Message	Accuracy	Methods	Conclusion
1.		1. Howareyou 2. OrchiSaha 3. RiyaRoy1 4. abcd	Message number 1, 2, 3 are worked 100% in Rail Fence Encryption & Decryption Method. Message number 4 worked 100% in Spiral Encryption but worked partially in Spiral Decryption.	1. Rail Fence Encryption 2. Spiral Encryption(4)	Our project is fully working on these three texts
2.		1. Howareyou 2. OrchiSaha 3. RiyaRoy1 4. abcd	Message number 1, 2, 3 are worked 100% in Rail Fence Encryption & Decryption Method. Message number 4 worked 100% in Spiral Encryption but worked partially in Spiral Decryption.	1. Rail Fence Encryption 2. Spiral Encryption(4)	Our project is fully working on these three texts
3.		1. Howareyou 2. OrchiSaha 3. RiyaRoy1 4. abcd	Message number 1, 2, 3 are worked 100% in Rail Fence Encryption & Decryption Method. Message number 4 worked 100% in Spiral Encryption but worked partially in Spiral Decryption.	1. Rail Fence Encryption 2. Spiral Encryption(4)	Our project is fully working on these three texts
4.		1. Howareyou 2. OrchiSaha 3. RiyaRoy1 4. abcd	Message number 1, 2, 3 are worked 100% in Rail Fence Encryption & Decryption Method. Message number 4 worked 100% in Spiral Encryption but worked partially in Spiral Decryption.	1. Rail Fence Encryption 2. Spiral Encryption(4)	Our project is fully working on these three texts

Chapter 6:

FUTURE SCOPE

- To perform image steganography for messages of any length
- To perform image steganography of any size of images.
- To implement improved edge-detection and corner-point detection techniques.
- Implementation of complete client server based application program.
- Vulnerity checking in presence of noise in the channel.
- Implementation of error correction and detection methods at receiver end.
- Implementation of Encryption Wheel with user interactive interface.

Chapter 7:

REFERENCES

1. Chan C-K, Cheng L-M: Hiding data in images by simple LSB substitution. *Pattern Recognit* 2004, 37(3):469-474. 10.1016/j.patcog.2003.08.007

2. Ker A: Improved detection of LSB steganography in grayscale images. In *Lecture Notes in Computer Science: 6th International Workshop on Information Hiding, Toronto, Canada*. Edited by: Fridrich J. Springer Berlin; 2004:97-115.

3. Ker AD: Steganalysis of LSB matching in grayscale images. *IEEE Signal Process. Lett*2005, 12(6):441-444.

4. Pevnỳ T, Filler T, Bas P: Using high-dimensional image models to perform highly undetectable steganography. In *Lecture Notes in Computer Science: 12th International Conference on Information Hiding, Calgary, AB, Canada*=. Edited by: Safavi-Naini R, Böhme R, Fong PWL. Springer Berlin; 2010:161-177.

5. Ker AD: Steganalysis of embedding in two least-significant bits. *IEEE Trans. Inf. Forensics Security* 2007, 2(1):46-54.

6. Fridrich JJ, Pevný T, Kodovský J: Statistically undetectable jpeg steganography: dead end challenges, and opportunities. Paper presented at the 9th workshop on multimedia & security (MM&Sec), Dallas, TX, USA, 20–21 Sep 2007, pp. 3–14

7. Ker AD: A general framework for the structural steganalysis of LSB replacement. In *Lecture Notes in Computer Science: 7th International Workshop on Information Hiding, Barcelona*. Edited by: Barni M, Herrera-Joancomartí J, Katzenbeisser S, Pérez-González F. Springer Berlin; 2005:296-311.

8. Provos N, Honeyman P: Hide and seek: an introduction to steganography. *IEEE Secur. Privacy* 2003, 1(3):32-44. 10.1109/MSECP.2003.1203220

9. Westfeld A, Pfitzmann A: Attacks on steganographic systems - breaking the steganographic utilities EzStego, Jsteg, Steganos, and S-Tools - and some lessons learned. In *Lecture Notes in Computer Science: 3rd International Workshop on Information Hiding, Dresden, Germany*. Edited by: Pfitzmann A. Springer Berlin; 1999:61-76.

10. Dumitrescu S, Wu X, Wang Z: Detection of LSB steganography via sample pair analysis. In *Lecture Notes in Computer Science: 5th International Workshop on Information Hiding, Noordwijkerhout, The Netherlands*. Edited by: Petitcolas FAP. Springer Berlin; 2002:355-372.

11. Fridrich JJ, Goljan M, Hogea D, Soukal D: Quantitative steganalysis of digital images: estimating the secret message length. *Multimedia Syst* 2003, 9(3):288-302. 10.1007/s00530-003-0100-9

12. Fridrich JJ, Goljan M: On estimation of secret message length in LSB steganography in spatial domain. Paper presented at the SPIE security, steganography, and watermarking of multimedia contents VI, San Jose, CA, USA, 18–22 Jan 2004, vol. 5306, pp. 23–34

13. Pevný T, Bas P, Fridrich JJ: Steganalysis by subtractive pixel adjacency matrix. *IEEE Trans. Inf. Forensics Security* 2010, 5(2):215-224.

14. Fridrich JJ, Kodovský J: Rich models for steganalysis of digital images. *IEEE Trans. Inf. Forensics Security* 2012, 7(3):868-882.

15. Kodovský J, Fridrich JJ, Holub V: Ensemble classifiers for steganalysis of digital media. *IEEE Trans. Inf. Forensics Security* 2012, 7(2):432-444.

16. Kouider S, Chaumont M, Puech W: Adaptive steganography by oracle (ASO). Paper presented at the IEEE international conference on multimedia and expo (ICME), San Jose, CA, USA, 15–19 July 2013, pp. 1–6

17. Holub V, Fridrich J, Denemark T: Universal distortion function for steganography in an arbitrary domain. *EURASIP J. Inform. Secur* 2014., 2014(1): doi:10.1186/1687-417X-2014-1

18. Filler T, Fridrich JJ: Design of adaptive steganographic schemes for digital images. Paper presented at the media watermarking, security, and forensics XIII, part of IS&T SPIE electronic imaging symposium, San Francisco, CA, USA, 24 Jan 2011, vol. 7880, pp. 1–1

19. Fridrich JJ, Kodovský J: Multivariate Gaussian model for designing additive distortion for steganography. Paper presented at the IEEE international conference on acoustics, speech and signal processing (ICASSP), Vancouver, BC, Canada, 26–31 May 2013, pp. 2949–2953

20. Wu D-C, Tsai W-H: A steganographic method for images by pixel-value differencing. Pattern Recogn. Lett 2003, 24(9-10):1613-1626. 10.1016/S0167-8655(02)00402-6

21. Zhang X, Wang S: Vulnerability of pixel-value differencing steganography to histogram analysis and modification for enhanced security. Pattern Recogn. Lett 2004, 25(3):331-339. 10.1016/j.patrec.2003.10.014

22. Yang C-H, Weng C-Y, Wang S-J, Sun H-M: Adaptive data hiding in edge areas of images with spatial LSB domain systems. IEEE Trans. Inf. Forensics Security 2008, 3(3):488-497.

23. Luo W, Huang F, Huang J: Edge adaptive image steganography based on LSB matching revisited. IEEE Trans. Inf. Forensics Security 2010, 5(2):201-214.

24. Hempstalk K: Hiding behind corners: using edges in images for better steganography. Paper presented at the second computing women congress (CWC), Hamilton, New Zealand, 11–19 Feb 2006, pp. 1–4

25. Dumitrescu S, Wu X, Memon N: On steganalysis of random LSB embedding in continuous-tone images. Paper presented at the international conference on image processing, Rochester, NY, USA, 22–25 Sept 2002, vol. 3, pp. 641–644

26. Ker AD, Böhme R: Revisiting weighted stego-image steganalysis. Paper presented at the SPIE electronic imaging, security, forensics, steganography, and watermarking of multimedia xontents X, Orlando, FL, USA, 27–31 Jan 2008, vol. 6819, pp. 5–1517

27. Tan S, Li B: Targeted steganalysis of edge adaptive image steganography based on LSB matching revisited using b-spline fitting. IEEE Signal Process. Lett 2012, 19(6):336-339.

28. Duda RO, Hart PE, Stork DG: Pattern Classification. Wiley, New York, NY; 2001.

29. Gul G, Kurugollu F: A new methodology in steganalysis: breaking highly undetectable steganography (HUGO). In Lecture Notes in Computer Science: 13th International Conference on Information Hiding, Prague, Czech Republic. Edited by: Filler T, Pevný T, Craver S, Ker A. Springer Berlin; 2011:71-84.

30. Holub V, Fridrich J: Digital image steganography using universal distortion. Paper presented at the first ACM workshop on information hiding and multimedia security, Montpellier, France, 17–19 June 2013, pp. 59–68

31. Holub V, Fridrich J: Designing steganographic distortion using directional filters. Paper presented at the IEEE international workshop on information forensics and security (WIFS), Tenerife, Spain, 2–5 Dec 2012, pp. 234–239

32. Filler T, Judas J, Fridrich J: Minimizing additive distortion in steganography using syndrome-trellis codes. IEEE Trans. Inf. Forensics Security 2011, 6 (3):920-935.