Chapter 3

METHODOLOGY

- Copy-Paste region detection
- Suspected Partition detection
CHAPTER-3: METHODOLOGY

3.1 Copy-Paste Detection

The prime objective of the study was to device a program or software, which takes as an input the digital image file, assumed to be forged, and determines or spots the areas of forgery from it.

The modern familiar programming techniques used to perform different operations on image files are:

1. Java Development kit (JDK 1.2)
2. MATLAB® 7.5.0.342 (R2007b)

The basic programming operations required on a given suspected image file available in JPEG or BMP or PNG format are as follows:

a. Recording each pixel value of an image in form of an array data structure.

b. Comparison of every picture element (pixel) with all other elements, in order to determine the similarity between the set of pixels, if exists.

c. Determining the statistics of similar pixels found in the same image.

d. Identifying the extent of image forgery, and spot out the areas of forgery within an image.

In order to test the methodology suggested, 10 sample images with different details of scenes are selected. These sample images act as an input to the program, whereas the execution of program resembles the processing part. In a whole, the following notion is used to solve the problem of image forgery detection.

![Figure 3.1: Notion applied towards Copy-Paste region Detection.](image-url)
Based on the above requisite, the program using Java was developed as below:

```java
import java.awt.*;
import java.applet.*;
import java.awt.image.*;

public class forgtest extends Applet {
    Image ori, nw; // ori is test image, nw is resultant image.
    int pxl[] = new int [15000]; // 100x150 pixel image.
    PixelGrabber p=new PixelGrabber (ori,0,0,300,50,pxi,0,300);
    // Image is stored into array
    try { p.grabPixels(); }
    catch( InterruptedException e) { }
    for (int i=0; i<15000; i++) // Compare every value.
    {
        int p = pxl[i];
        for (int j=0; j<15000; j++)
        {
            int q = pxl[j]; // Check if they are same,
            if (q==p) pxl[i] = 0; // Make BLACK.
        }
    }
    nw = createlmage (new MemorylmageSource(300,100,pxl, 0,300) ;
    // Applet designed.
    public void paint (Graphics g)
    {
        g.drawImage( ori, 10, 10, this); // Original.
        g.drawImage(nw, 400, 10, this); // Result.
    }
}
```

However, by using Java source code mentioned above, the image forgery was not identified as per the desired results.

Hence, a program using MATLAB 7.2 was devised, considering the below mentioned advantages of MATLAB:

**Recording of the processing used**
MATLAB is a general purpose programming language. When it is used to process images one generally writes function files, or script files to perform the operations. These files form a formal record of the processing used and ensures that the final results can be tested and replicated by others should the need arise.

**Access to implementation details**
MATLAB provides many functions for image processing and other tasks. Most of these functions are written in the MATLAB language and are publicly readable as plain text files. Thus the implementation details of these functions are accessible and open to scrutiny. The defense can examine the processing used in
complete detail, and any challenges raised can be responded to in an informed way by the prosecution. This makes MATLAB very different from applications, such as Photoshop.

Some MATLAB functions cannot be viewed. These are generally lower level functions that are computationally expensive and are hence provided as 'built-in' functions running as native code. These functions are heavily used and tested and can be relied on with considerable confidence.

Numerical accuracy
Another advantage of MATLAB is that it allows one to ensure maximal numerical precision in the final result.

In general, image files store data to 8 bit precision. This corresponds to a range of integer values from 0-255. A pixel in a colour image may be represented by three 8 bit numbers, each representing the red, green and blue components as an integer value between 0 and 255. Typically this is ample precision for representing normal images.

However as soon as one reads this image data into memory and starts to process it is very easy to generate values that lie outside the range 0-255. For example, to double the contrast of an image one multiplies the intensity values by 2. An image value of 200 will become 400 and numerical overflow will result. How this is dealt with will vary between image processing programs. Some may truncate the results to an integer in the range 0-255, others may perform the mathematical operations in floating point arithmetic and then rescale the final results to an integer in the range 0-255.

It is here that numerical precision, and hence image fidelity, may be lost. Some image processing algorithms result in some pixel values with very large magnitudes (positive or negative). Typically these large values occur at points in the image where intensity discontinuities occur, the edges of the image are common sources of this problem. When this image with widely varying values is rescaled to integers in the range 0-255 much of this range may be used just to represent the few pixels with the large values. The bulk of the image data may then have to be represented within a small range of integer values, say from 0-50. Clearly this represents a considerable loss of image information. If another process is then applied to this image the problems can then accumulate. Trying to establish the extent of this problem, if any, is hard if one is using proprietary software.
Chapter 3: Methodology

Being a general programming language it is possible to have complete control of the precision with which one represents data in MATLAB. An image can be read into memory and the data cast into double precision floating point values. All image processing steps can then be performed in double precision floating point arithmetic, and at no intermediate stage does one need to rescale the results to integers in the range 0-255. Only at the final point when the image is to be displayed and/or written to file does it need to be rescaled. Here one can use histogram truncation to eliminate extreme pixel values so that the bulk of the image data is properly represented.

Advanced algorithms
MATLAB is a scientific programming language and provides strong mathematical and numerical support for the implementation of advanced algorithms. It is for this reason that MATLAB is widely used by the image processing and computer vision community. New algorithms are very likely to be implemented first in MATLAB, indeed they may only be available in MATLAB.

The following algorithm is applied for detection of Copy-Paste regions in an image:

\textbf{Algorithm 3.1: Detection of Copy-Paste regions}

\textbf{Step 1.} Initialize the following parameters of an image.
- \( I \): Image array of dimension \( M \times N \) comprising of color value.
- \( P \): An \( M \times N \) Boolean value array indicating whether the positioned pixel comparison is completed or not.
- \( x \) and \( y \): Current pixel position coordinates.
- \( a \) and \( b \): Likely to be similar pixel coordinates.
- \( c \): Similarity color coefficient, set to zero in case of same color.

\textbf{Step 2.} Initialize \( P_{x,y} = \text{false} \), indicating the comparison operation not performed.

\textbf{Step 3.} Process Step-4 and Step-5 for \( x, a = 1 \) to \( M \) and \( y, b = 1 \) to \( N \)

\textbf{Step 4.} If \( (I_{x,y} = I_{a,b} \pm c) \) and \( (I_{x+1,y} = I_{a+1,b} \pm c) \) and \( (P_{x,y} = \text{true}) \)
then
\( I_{a,b} = \text{BLACK} \) to set the detected similar pixels with desired color.

\textbf{Step 5.} Set \( P_{x,y} = \text{true} \), indicating that comparison is over.
Chapter 3: Methodology

The code developed to implement this algorithm, for Copy-Paste detection in an image using MATLAB is as shown below:

Program 3.1: Implementation of algorithm 3.1 in MATLAB

```matlab
% Identification of Copy-Paste Area in BMP image
i=imread('D:\RESEARCH\PhD\tl.png','png');
% Save Bitmap 24 bit format in Array I
z=zeros(220,220);
% To check that positioned pixel is already processed or not
for x=1:95
    for y=1:40
        % Compare each pixel with every other.
        for a=x:95
            for b=1:40
                if (z(a,b)==0) % If not processed then ..
                    if ( (i(x,y,1)==i(a,b,1)) && (i(x,y,2)==i(a,b,2)) && (i(x,y,3)==i(a,b,3)) && (~((x==a) && (y==b))) )
                        % Checked that - (i) Pixel value of adjacent is same
                        % - (ii) Its not compared with itself
                        i(x,y,1)=0; % Make all 3 values 0 ,
                        i(x,y,2)=0;
                        % Indicating that same pixels are DETECTED and Black.
                        i(x,y,3)=0;
                        %i(a,b,1)=100;
                        %i(a,b,2)=100; % i(a,b) represents another set of copy.
                        i(a,b,3)=100;
                    end % of inner if
                end % of outer if
            end % of var. b
        end % of var. a
    end % of var. y
end % of var. x
imshow(i) % Display detected portions, with black.
```

Brijesh Jajal, Ph.D. Thesis (Computer Science, January 2012)
3.2 Suspected Partition Detection

In an experiment to detect the suspected partitions of an image, an algorithm is devised to spot such forgery or manipulated areas. The logical sequence for implementation of an idea is exhibited in figure-3.2.

![Figure 3.2: Notion used in suspected partition detection.](image1)

When the copy-paste operation is assumed not to be performed in the same image, we can assume that region can be suspected to be from some other image. In such cases, there are chances of the traces at the position at which the paste operation is carried out. Such locations can be easily detected, since there are high changes in neighboring pixel values, compared to the other positions in an image. The changes can be detected in horizontal (as in figure-3.3) or vertical axis.

![Figure 3.3: Pixel positions in an image.](image2)

When a portion of image is copied from another image, we can assume to identify a vertical or horizontal line of reference, of length L, where all neighboring pixels carry high amount of difference in color value i.e. $I_{x,y} - I_{x+1,y} = F$. Based on this assumption, algorithm 3.2 is devised.
**Algorithm 3.2:** Detection of Suspected Partitions.

**Step 1.** Let a and b are height and width of an RGB image, 
I is the image pixel value representation matrix of size a x b. 
F is the determining factor for the pixel difference.

**Step 2.** Determine the difference between two pixels. 
For each value of i = 1 to a, and j = 1 to b 
Check for the following conditions to be evaluated to true:

- a) \( \text{abs} (I_{a,b} - I_{a,b+1}) > F \)
- b) \( \text{abs} (I_{a,b} - I_{a,b-1}) < F \)
- c) \( \text{abs} (I_{a-1,b} - I_{a+1,b+1}) > F \)
- d) \( \text{abs} (I_{a-1,b-1} - I_{a+1,b+1}) < F \)

**Step 3.** If all above four conditions (a, b, c and d) are satisfied, it indicates that two pixels along the same axis contain the split color values (highly different) or assumed to be forgery prone.

**Step 4.** Considering the required level of detection L being no. of columns or height of forgery, Set partition line to indicate black color. 
\( I_{a,b} = 0, \text{ where } x = a - (L/2) \text{ to } a + (L/2) \).

The algorithm 3.2 is implemented using equivalent MATLAB code, as below:

**Program 3.2:** Implementation of algorithm 3.2 in MATLAB

```matlab
% Detection of a major variation in image row/column due to forgery.
% Considering im2 as merged image.
% Find difference between pixel values of same image.

il = imread('d:\sample1.bmp', 'bmp');
x = 1;
for a = 2:109
    for b = 2:109
        % Next Column is different
        nc1 = abs(il(a,b,1) - il(a,b+1,1));
        nc2 = abs(il(a,b,2) - il(a,b+1,2));
        nc3 = abs(il(a,b,3) - il(a,b+1,3));
        % Previous column is similar
        pc1 = abs(il(a,b,1) - il(a,b-1,1));
        pc2 = abs(il(a,b,2) - il(a,b-1,2));
        pc3 = abs(il(a,b,3) - il(a,b-1,3));
```
% Above next is different
an1 = abs(il(a-1,b,1) - il(a-1,b+1,1));
an2 = abs(il(a-1,b,2) - il(a-1,b+1,2));
an3 = abs(il(a-1,b,3) - il(a-1,b+1,3));

% Above Previous is similar
ap1 = abs(il(a-1,b-1,1) - il(a-1,b,1));
ap2 = abs(il(a-1,b-1,2) - il(a-1,b,2));
ap3 = abs(il(a-1,b-1,3) - il(a-1,b,3));

if (nc1>5||nc2>5||nc3>5)&&(an1>5||an2>5||an3>5)&&(pcl<5 && pc2<5 && pc3<5 && ap1<5 && ap2<5 && ap3<5 )
    il(a,b,1)=0; il(a-1,b,1)=0; %Pixel and its above.
il(a,b,2)=0; il(a-1,b,2)=0;
il(a,b,3)=0; il(a-1,b,3)=0;
end
end
end

i2 = imread('Merge.bmp','bmp');

% Check for 7 consecutive occurrences, for L = 7
for a = 4:109
    for b= 1:109
        if (il(a,b)==0 && il(a-2,b)==0 && il(a-1,b)==0 &&
il(a+1,b)==0 && il(a+2,b)==0 && il(a+3,b)==0 &&
il(a+3,b)==0)
            i2(a-3,b,1)=0; i2(a-3,b,2)=0;i2(a-3,b,3)=0;
i2(a-2,b,1)=0;i2(a-2,b,2)=0;i2(a-2,b,3)=0;
i2(a-1,b,1)=0;i2(a-1,b,2)=0;i2(a-1,b,3)=0;
i2(a,b,1)=0;i2(a,b,2)=0;i2(a,b,3)=0; % Set to BLACK
            i2(a+1,b,1)=0; i2(a+1,b,2)=0; i2(a+1,b,3)=0;
i2(a+2,b,1)=0;i2(a+2,b,2)=0;i2(a+2,b,3)=0;
i2(a+3,b,1)=0;i2(a+3,b,2)=0;i2(a+3,b,3)=0;
        end % of IF statement
    end % Inner loop
end % Outer loop
imshow(i2);