Chapter 3 Academic Research

In the previous chapter we discussed the generics of an Image processing application especially the understanding of Machine vision applications. The project is about object recognition and this chapter will tell us how to get to that objective. We will start of by first understanding a few details about image processing i.e. convolution, RGB etc. There onwards we will move on to filter, kernel Masks etc. At the end of this chapter we should have a clear understanding of the current systems in the market, and where this system is different. Moreover all of these underlined requirements will then place us in a position to undergo the analysis stage where the solution will be devised.

3.1 Concepts and Processing

Image acquisition and compression are not the major axis points of this project, but what is important are the image enhancement and image analysis operations. By far the most common and known feature of image processing is the image enhancement because it visually enhances the image to a form where the computer can understand image information easily.

Before we dive right in to the enhancement arena we should first go through the common concepts of image processing and i.e. an image is a rectangular array of pixels. Each element of the array is physically located on the screen in terms of the Cartesian coordinates. The image-starting axis at 0,0 that is located on the left upper corner. Pixel coordinates ascend from left to right where as line coordinates ascend from top to bottom of the image.

Another important concept of pixel data is the RGB. These are the regular tones of color any display system would have RED, BLUE, GREEN for RGB respectively. Red, green, and blue can be combined in various proportions to obtain any color in the visible spectrum. Levels of R, G, and B can each range from 0 to 100 percent of full intensity. The range of decimal numbers represents each level from 0 to 255. We will be discussing the importance of RGB further on in detail.

Moreover another important highlight is the usage of resolution through out this document that is 800x600 pixels.

Image processing is typically done in two ways either pixel point processing or group pixels processing. We will talk about each of them, as they are an essential to this project.

3.2 Pixel Point Processing

Is the most raw and basic form of image processing. It takes the input as x and y coordinates and processes them one by one to come up to the expected result. Each pixel is modified according to a specific spatial location in the resulting output image.

A general equation for a point process is given by the equation

$O(x, y) = M[I(x, y)]$

Where M is the mapping function. This function inputs brightness to output brightness.

I (x, y) is the input coordinate location where as the O (x, y) will be the output image. Any specific pixel manipulation will perform its processing in such a way that the x, y coordinates of one will always reflect the x, y of the output image. The main process is to initiate the image in to an array of pixels and start processing them and then after the processing is done the pixel is re-initialized with the new resultant on the pixel x, y.

A very simple operation is that of a *"Complement Image"* this is one of the simplest operations that exist and most of the authors will take this as their first example to understanding pixel processing.

Suppose we wish to make a positive image from a negative one. Just like the whites in the input image will be blacks in the output image. And all other colors will take their respective reverse colors.

Before taking on the processing unit we will first go through a basic algorithm of image reading, which will help us in understanding the process of image manipulation.

Code Insight 3.1

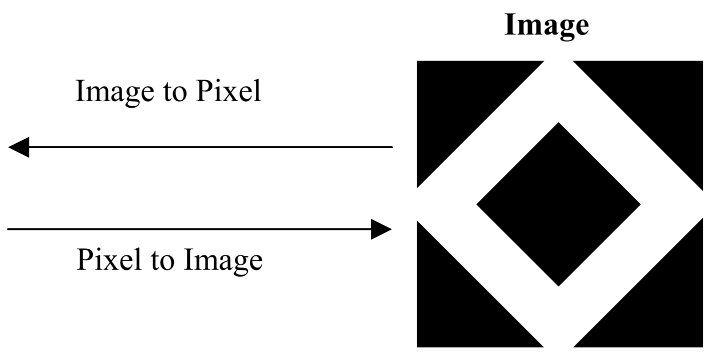
```
/**
Reading an image in a 2D array
*/
1.  Begin
2.  Input Image
3.  Get Pixels in RGB
4.  Convert R, G and B into 2 Dimensional Arrays
5.  For I = 0 to image height of image
6.          Next step
7.          For j = 0 to width of image
8.                  Next Step
9.                  Get R [I][j]          {Get R can be replaced with Get G and also Get B}
10.                 Get R [I][j]=Process ([I][j])
11.         End for
12. End for
13. 2 Dimensional Arrays to Image
14. Display new Image
15. End
```

The algorithm shows that the image is read into an array of pixels i.e. three 2D arrays R, G and B respectively. It is to be noted that if the processing is in color the R G and B will be separately processed else if it is in gray scale that is ranging from 0 to 255 then the any of the arrays can be processed and copied on to the others. For instance as on line no 9 only the R array is processed which can then be further on copied to G and B which is not shown in this algorithm and will be discussed later.
In our example the pixel image could be something as follows.

Figure 3.1

Pixels

0	0	255	0	0
0	255	255	255	0
255	0	0	0	255
0	255	0	255	0
0	0	255	0	0

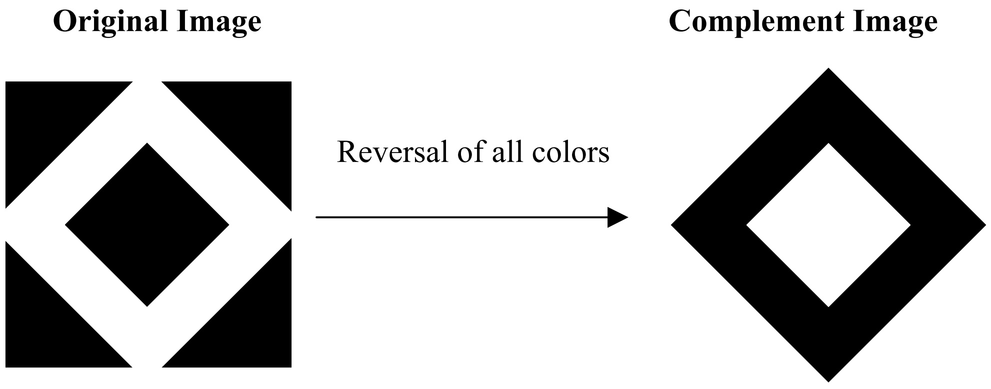


In the above example we can see the data represented by the image in form of pixels i.e. grey scale which means all R G and B will have the same value. Hat we need to understand here is that the data is plotted in a 2D array, which makes our lives easier.☺

3.2.1 Complement image: *(Reference Point 1)*

Coming back to our first pixel point operation the Complement Image Operation, remember that we were discussing the point where all blacks will become white and all whites will become black. More over all grey colors will have been reversed. But to understand fully I am showing the result of the figure 3.1 when it is complemented as follows.

Figure 3.2



Complement operations are often useful in making the subtle brightness details in bright areas of an image more visible. As we discussed in our previous chapter that subtle changes in the brightness regions are not very clear for the human eye, thus making them darker will make things easier.

A graphical view of this representation will help us more to understand the change. It only shows 5 pixel related data. The pixel values are set on the white y-axis and the pixels are set on the x-axis. The curve denotes the values of pixels at different points. Following figure shows the original image and the next shows the values after the pixels have been complemented.

Figure 3.3. The original image pixel representation of first 5 pixels

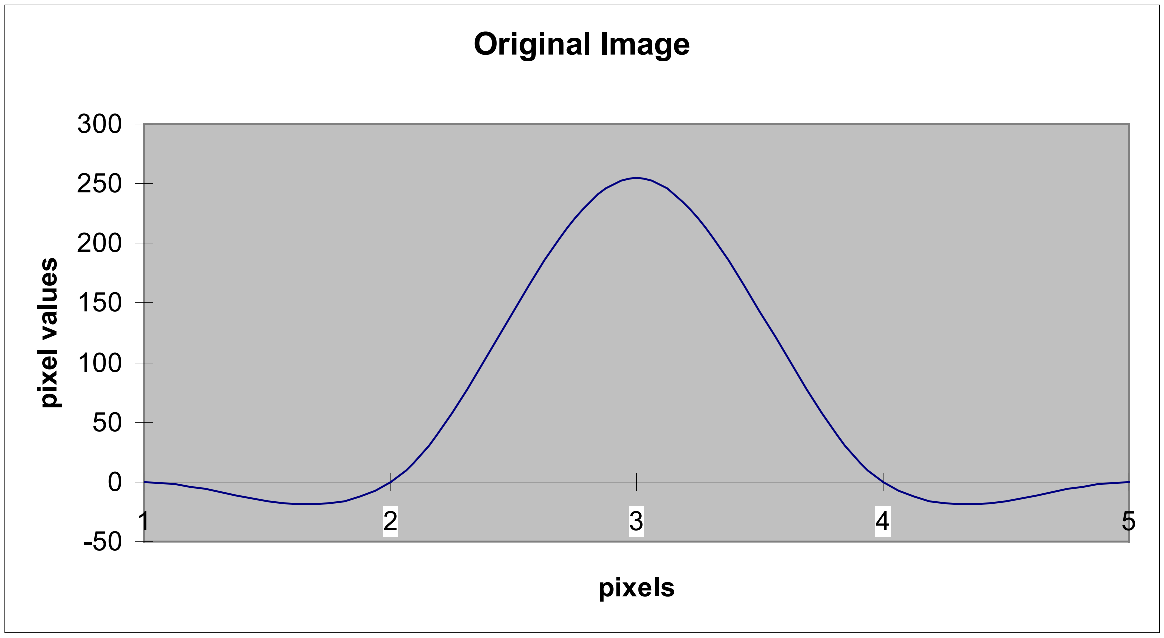
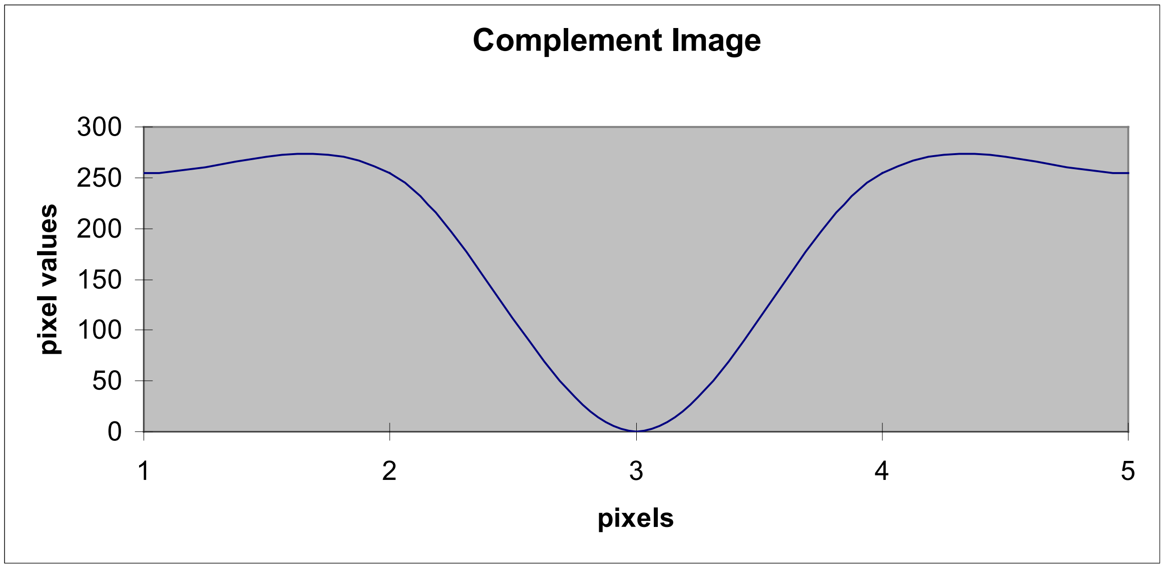


Figure 3.4



Complement image is the basic form of pixel-by-pixel processing. We will now move on to a few more processing techniques used in our system using Pixel Point Processing.

3.2.2 Gray Level Classification: *(Reference Point 2)*

This pixel point operation is very useful for the system. It parameterizes the pixels into groups of different levels. If the pixels are between the specified thresholds they are given a certain predefined brightness value. This process will make a lot of sense when the image is clearer if not pixels can be misclassified.

Implementation:

```
If (x, y) >lowValue && less then midValue
        Then x, y = 0
Else if (x, y) > middleValue && less then HighValue
        Then x, y = 64
Else
        x, y = 255
```

Results:

Original Image **Gray Level Classification**

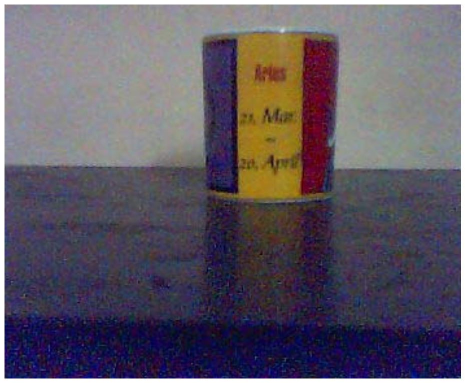


Figure 3.5

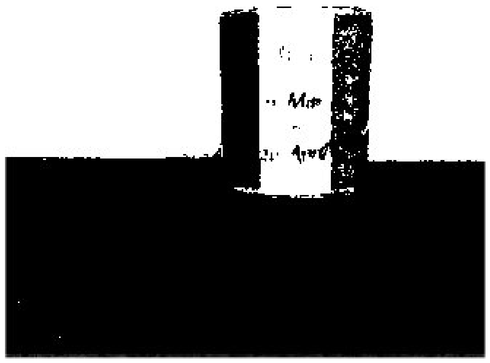


Figure 3.6

It is up to the programmers or the application developer to implement as many conditions between the thresholds they want to give their application what it needs. However in the example only two conditions where made to put all the pixels above 100 as white and the rest as black.

3.2.3 Gray Scale Images: *(Reference Point 3)*

Converting image into gray scales is one of the preliminary operations when we think of this projects main objective. It is a way through which all the R, G and B will become equal and within the range of gray of 0 to 255. This makes it easier for us to process image in a more simplistic way.

Implementation:

```
//load colors
int R = (rgb >> 16) & 0xff;
int G = (rgb >> 8) & 0xff;
int B = (rgb >> 0) & 0xff;

// vars for YIQ color model
float Y, I, Q;

// convert RGB to YIQ
Y = (float) (0.299*R + 0.587*G + 0.114*B);
I = (float) (0.596*R - 0.275*G - 0.321*B);
Q = (float) (0.212*R - 0.523*G + 0.311*B);

// covert color image to greyscale
R = (int) Y;
G = (int) Y;
B = (int) Y;
return (rgb & 0xff000000)  | (R<<16) | (G << 8) | (B <<0);
```

Results:

Original Image

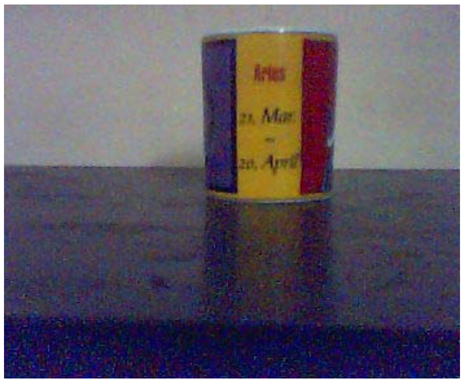


Figure 3.5

Gray Scale

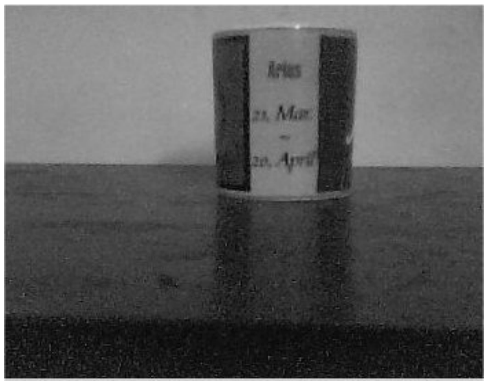


Figure 3.7

The implementation suggests that x and y positions of pixels are passed to the function where each pixel is separately processed and then returned back with modifications as *"Grey Scale"*.

These were a few pixel-by-pixel operations. We will be discussing them further on when we talk about image analysis and boundary segmentations. However for now we will discuss the second type of image processing on pixels and that is the Pixel Group Processing.

3.3 Pixel Group Processing

Through out the previous section it is quite much noticeable that all the operations that took place with a pixel-by-pixel approach were all independent of their neighbor pixels. For simple image processing applications it might be easy to go either of the two approaches pixel or pixel group. But for Intelligent applications that are well suited for computer vision are to be dealt with severely when talking about boundaries and segmentation. These are the concepts which will actually need a lot of processing in terms of theirs neighboring pixels. Pixel processing is no doubt an important segment of every application because it helps in a lot more other situations rather then just complementing and gray scaling. However the pixel group processing introduces a lot of other dimensions to help guide the application for a better information arena.

Pixel group processing will operate on a group of surrounding pixels with a center pixel. The surrounding pixels or the bordering pixels will have some brightness information that will be valuable for our processes. This is done through *"Spatial Frequency"*.

An Image is formed with two levels of frequency distribution. *"High Frequency"* represents sharp edges or where a lot of points are present. Slowly changing edges and brightness transitions will represent Low Spatial Frequency. Using the concept of "Spatial Convolution" we can operate upon these Spatial Frequencies.

The Spatial Convolution uses a weighted average formula based on the calculation of a Kernel. Kernel is a dimensional Square that has an odd number of mask values. It can take the shape of 3x3, 5x5 and etc. The larger size of the kernel we produce will add more freedom to the spatial filters. To process images within this domain we will use *"Convolution Masks"* which is an array of numeric values (coefficients). These masks can take any value but it is important that they are processed in a form where values do not exceed 255.

The numerical method of processing is for instance we have an image of 300 x 300 line image this would mean that for a 3 x 3 kernel mask we would need a 9 x9 multiplications and nine additions per pixel. The calculations are quite clear they would top the million marks. But the answer lies is it really that much worth? Does it take a lot of time? We will discuss all that later but for now lets see a few important masks used in this project.

The basic implementation of such operations is as follows

Code Insight 3.2

```
/**
start
Acquire Image
Copy to buffer
 Build a Kernel
Start processing
{
x-1,y-1          x,y-1          x+1,y-1
x-1,y            x,y            x+1,y
x-1,y+1    x,y+1          x+1,y+1
}
buffer to image
end
**/.
```

We will now discuss all the image operations researched and studied in this project.

3.3.1 Low Pass Filter: *(Reference Point 4)*

The basic objective of a low pass filter is to smooth down the edges of an image by satisfying high frequency details. The result can be brightness scaled down where points at the image selected to vary on different frequency domains.

This filter is good in reducing noise and also efficient in its purpose where it leaves the low frequency details as they are and smoothens the high ones. I have used this through out the object detection process where a different combination of filters is applied.

Implementation

The mask use in the project is as follows.

```
 float[] lowPass={
                1.0f/16.0f , 2.0f/16.0f , 1.0f/16.0f,
                2.0f/16.0f , 4.0f/16.0f , 2.0f/16.0f,
                1.0f/16.0f , 2.0f/16.0f , 1.0f/16.0f
               };
```

Results

Original Image

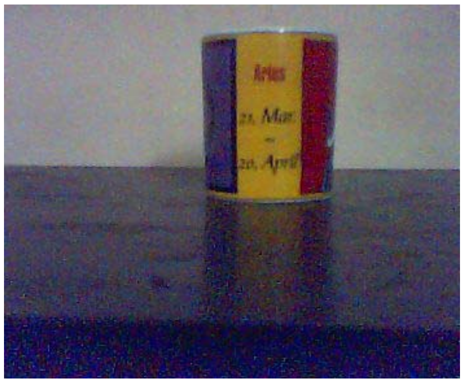


Figure 3.5

Low Pass

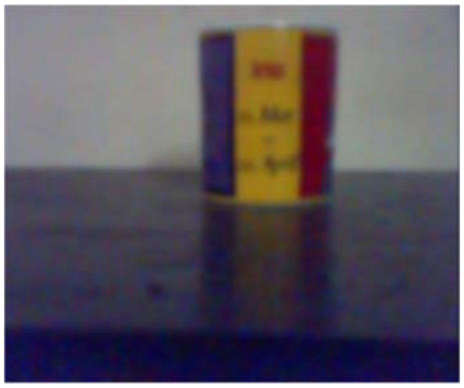


Figure 3.8

3.3.2 High Pass filter: *(Reference Point 5)*

High pass filter is quite much the opposite of the low pass filter. It highlights the high frequency details in the image. In the overall high pass filter will increase the brightness of the image and making the white like pixels very much prominent. All the masks have different effects and we have implemented three in our project. High pass filter was very much useful in this project as it was used in combination with the low pass filter to make details more prominent and accurate. Examples of these will be discussed later.

Implementation

The kernel masks are as follows

float[] hp1={
-1.0f, -1.0f, -1.0f,
-1.0f, 8.0f, -1.0f,
-1.0f, -1.0f, -1.0f
};
	float[] hp1={
	0.0f ,-1.0f , 0.0f,
	-1.0f , 5.0f , -1.0f,
	0.0f ,-1.0f , 0.0f
	};
		float[] hp1={
		-1.0f ,-1.0f , -1.0f,
		-1.0f , 9.0f , -1.0f,
		-1.0f ,-1.0f , -1.0f
		};

Results

Original Image

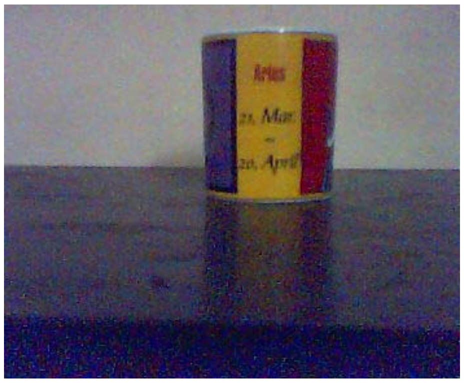


Figure 3.5

High Pass 1

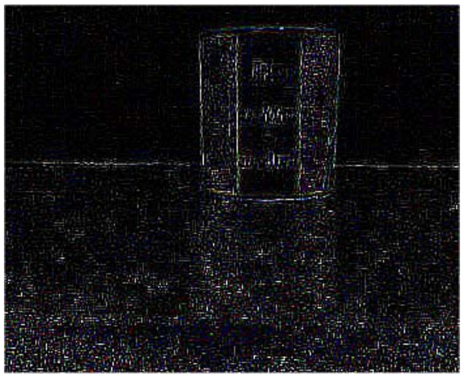


Figure 3.9

High Pass 2

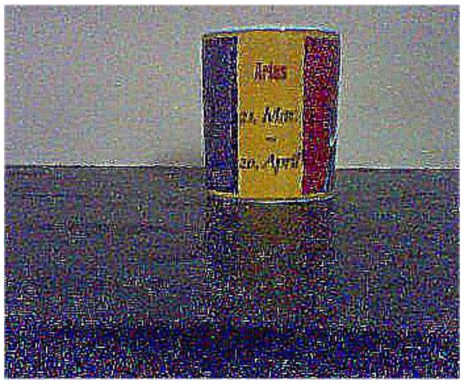


Figure 3.9

High Pass 3

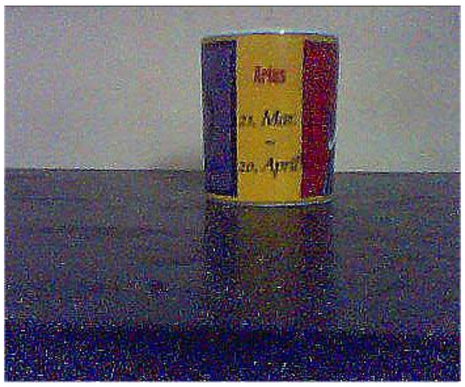


Figure 3.10

The high pass filter will usually produce negative values and for that particular reasons we remember the discussion of pixel group processing mentioned before where we had decided that any negative values will be given a range from 0 – 255. Generally these filters will have a larger values in the center surrounded by a minimal as neighbors.

3.3.3 Edge Enhancement Filters

3.3.3.1 Sobel Edge Enhancement: *(Reference Point 6)*

The sobel edge enhancement operation will processes regardless of any direction. It sums two directional edge enhancement operations and the result appears as an outline of the object. Constant regions will become black where as non-constant or moving regions/pixels will become white.

Implementation:

Define two masks vertical and horizontal

Vertical Mask	Horizontal Mask

float data_v[] = new float[] {
-1.0F, 0.0F, -1.0F,
0.0F, 0.0F, 0.0F,
1.0F, 2.0F, -1.0F
};
	float data_h[] = new float[] {
	1.0F, 0.0F, -1.0F,
	0.0F, 0.0F, 0.0F,
	1.0F, 0.0F, -1.0F
	};

Results:

Original Image **Sobel Enhance**

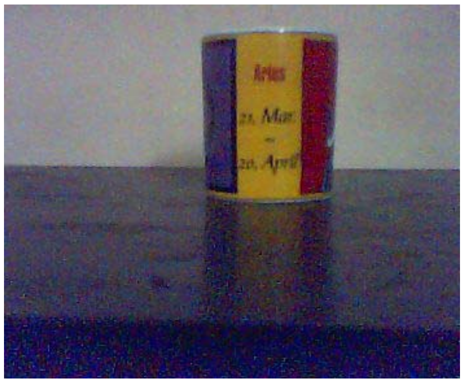


Figure 3.5

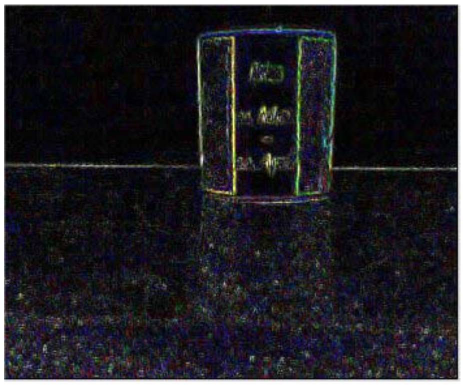


Figure 3.11

3.3.3.2 Prewitt Gradient Edge: *(Reference Point 7)*

This is a directional gradient edge enhancement. Where it operates on North, North East, East, South East, and West. The resulting image will also have the directional effects. However in this project I have used the Kernel Masks in a different way that will be represented as following. I have used two techniques for getting prewitt edges. By using only two masks Horizontal and Vertical and then keeping them as positive and negative values.

The positive values are named as prewitt Sensitive and the common as prewitt Edge:

Implementation:

Prewitt Edge

Vertical Mask	Horizontal Mask

float data_v[] = new float[] {
1.0F, 0.0F, -1.0F,
1.0F, 0.0F, -1.0F,
1.0F, 0.0F, -1.0F
};
	float data_h[] = new float[] {
	-1.0F, -1.0F, -1.0F,
	0.0F, 0.0F, 0.0F,
	1.0F, 1.0F, 1.0F
	};

Prewitt Sensitive

Vertical Mask	Horizontal Mask

float data_v[] = new float[] {
-1.0F, 1.0F, 1.0F,
-1.0F, -2.0F, 1.0F,
-1.0F, 1.0F, 1.0F
};
	float data_v[] = new float[] {
	1.0F, 1.0F, 1.0F,
	-1.0F, -2.0F, 1.0F,
	-1.0F, -1.0F, 1.0F
	};

It can be seen that I have not gone to the details of east, west, etc. But these Masks full fill the requirements so far.

Results can be seen on the next page.

Results

Original Image **Prewitt Edge**

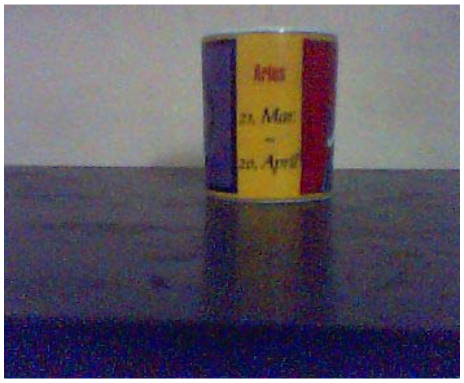


Figure 3.5

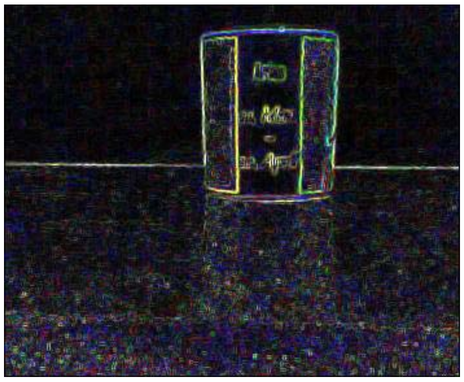


Figure 3.12

Prewitt Sensitive

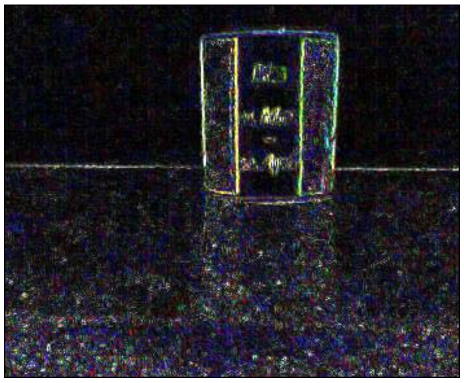


Figure 3.13

This experiment yielded quite an amazing result and different tones of colors where highlighted that were all different from every gradient filter used so far. It can bee seen that the prewitt edge filter shows a lot of blue; this means that all small edges are even highlighted. Prewitt sensitive just looks like sobel enhance to a lay user, here but we will talk about it in detail when we start off with object Detection.

3.3.3.3 Robert Cross Edge: *(Reference Point 8)*

The Robert cross edge performs a quick process on 2D images and highlights region of high Spatial Frequency. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point.

Roberts Cross is very fast as only four pixels are needed to process it and only to operations are performed subtractions or additions.

Implementation:

Horizontal Mask	Vertical Mask

float data_h[] = new float[] {
00F, 0.0F, -1.0F,
0.0F, 1.0F, 0.0F,
0.0F, 0.0F, 0.0F
};
	float data_v[] = new float[] {
	-1.0F, 0.0F, 0.0F,
	0.0F, 1.0F, 0.0F,
	0.0F, 00F, 0.0F
	};

Results

Original Image **Roberts Cross**

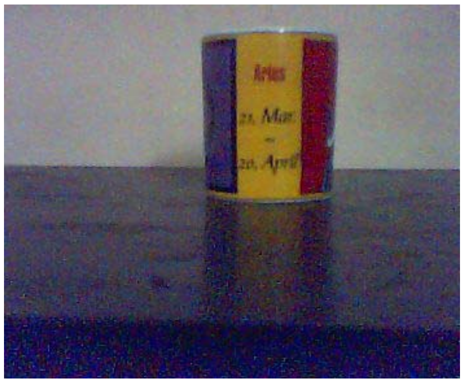


Figure 3.5

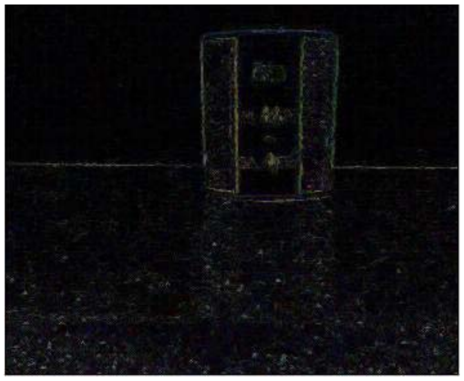


Figure 3.14

3.3.3.4 Shift And Difference Edge: *(Reference Point 9)*

The shift and difference operation is dependant on the subtraction operation. Each pixel is subtracted from its adjacent neighbors. The directions can be horizontal, Vertical and diagonal.

It can produce results that are less then zero.

Implementation:

Vertical	Horizontal	Diagnol
0.0F, -1.0F, 0.0F, 0.0F,1.0F,0.0F, 0.0F, 0.0F, 0.0F	0.0F,0.0F,0.0F, -1.0F,1.0F,0.0F, 0.0F, 0.0F, 0.0F	-1.0F, 0.0F, 0.0F, 0.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F

Results

Original Image

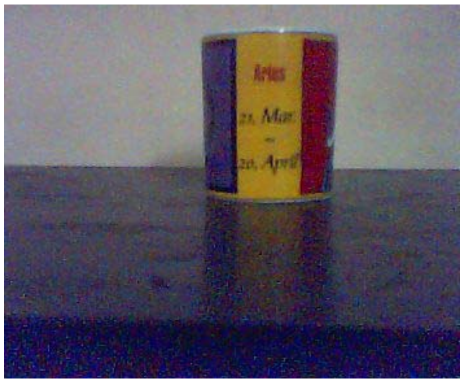


Figure 3.5

Shift And Difference

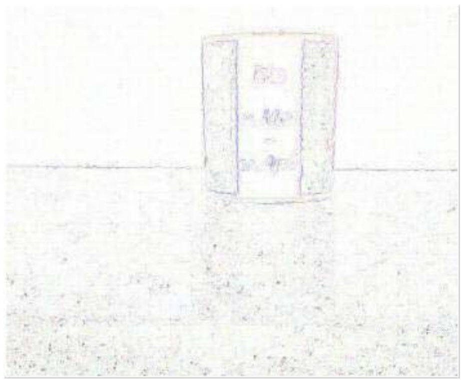


Figure 3.15

The above image has been inverted so that it is visible easily. The boundaries can now be seen.

3.3.3.5 Laplacian Edge Enhance: *(Reference Point 10)*

The Laplacian edge is just like the sobel edge enhance which does not have any direction and the results are omni directional. Constant brightness regions will become black and changing will become black. It can also return negative results.

The masks are as follows

Implementation:

Mask1	Mask2	Mask3
-1.0F, -1.0F, -1.0F, -1.0F, 8.0F, -1.0F, -1.0F, -1.0F, -1.0F	0.0F, -1.0F, 0.0F, -1.0F, 4.0F, -1.0F, 0.0F, -1.0F, 0.0F	1.0F, -2.0F, 1.0F, -2.0F, 4.0F, -2.0F, 1.0F, -2.0F, 1.0F

Results

Original Image

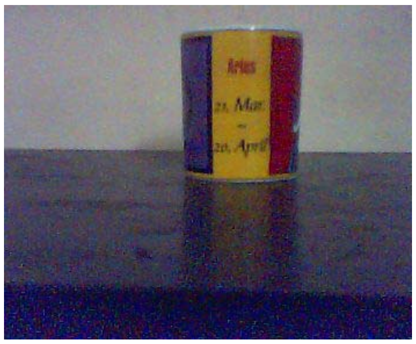


Figure 3.5

Laplacian Mask1

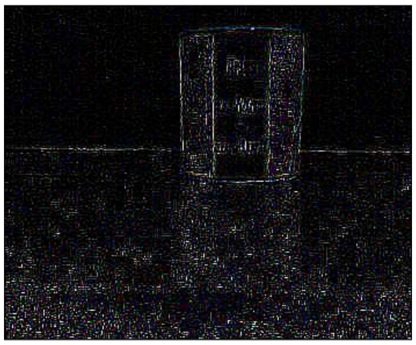


Figure 3.16

3.4 Image Analysis

We have now discussed the major five image edge detection operations. Edge detection operations play a vital role in detecting objects and creating a computer vision application. However they need to be supported by various other operations that we will discuss now onwards. These operations are called Image Analysis operations and they make the basis of this project. However before we move on forward we will look at the image Analysis in brief as to how such objectives are achieved.

The first step in any *image segmentation* application is to remove as much of noise possible. We have done that quite effectively by using the Low Pass filter or the blur filter.

The next step is the *image preprocessing* which means eliminating all needless features of the image. For example using pixel point operations to make the image into gray scale etc.

The third step is the *object Discrimination* that then identifies the information that will be processed further for analysis. These operations where covered in our previous section where we talked about edge enhancement with a particular criteria of detecting high Spatial Frequency.

The fourth stage of image Analysis then pertains to *Object boundary clean up*. For this we refer to the *Morphological image operations*. Image morphology is a means to work on the image boundaries and a lot of binary operations will be used in our project to perform under this category.

Image morphology can be of two types *gray scale morphology* or the *binary morphological processing*. Gray scale is an enhancement to the Binary processing where operations are completely on binary images.

We will now look at the binary morphological operations used within this research.

3.4.1 Binary Erosion, Dilation, Open and Close: *(Reference Point 11)*

The binary erosion operations reduce the size of white pixels on a black background where as the dilation operation increases the size of white pixels on a black background.

The erosion is used to remove small disturbances such as speckle or spur. Multiple application of erosion will remove such disturbances but will also reduce boundaries or might create differences or gaps in the boundaries if they are very thin. However dilation operations will help in adjoining pixels with broken links etc.

Implementation:

The kernel sets used are as follows

Horizontal	Vertical	Square	Cross

kh[][] = {
{0, 0, 0},
{1, 1, 1},
{0, 0, 0}
};
	kv[][] = {
	{0, 1, 0},
	{0, 1, 0},
	{0, 1, 0}
	};
		kSquare[][] = {
		{1, 1, 1},
		{1, 1, 1},
		{1, 1, 1}
		};
			kCross[][] = {
			{0, 1, 0},
			{1, 1, 1},
			{0, 1, 0}
			};

The results can bee seen on the next page. However a brief mention of the Binary Opening and Closing is also needed here. These processes help in forming the binary opening and closing. While using the same kernel we can make a combination of opening and closing operations. In the opening operation first the erosion operation eliminates the small disturbances in the image such as speckle. It also reduces the object sizes and then the dilation is followed which brings the object back to its original size and the disturbances are removed.

On the other hand the Binary Closing operations will first dilate and then erode. In this case the dilation eliminates the holes in an image and then erosion brings back the objects to their normal size.

The results on the next page make things much more clearer.

Original

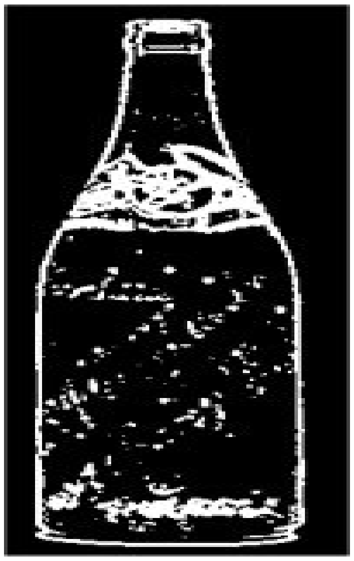


Figure 3.17

Erode

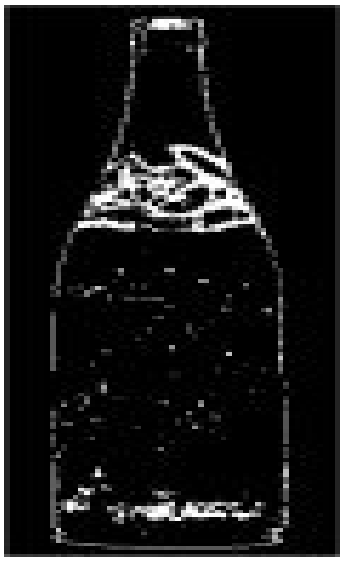


Figure 3.18

Dilate

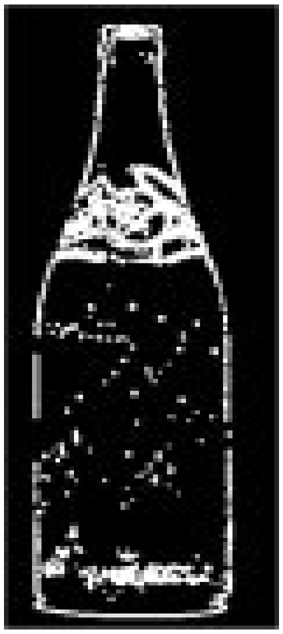


Figure 3.19

Open

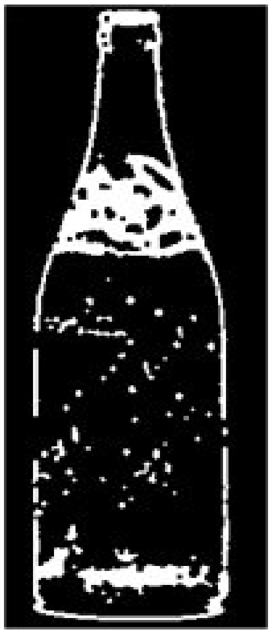


Figure 3.20

Close

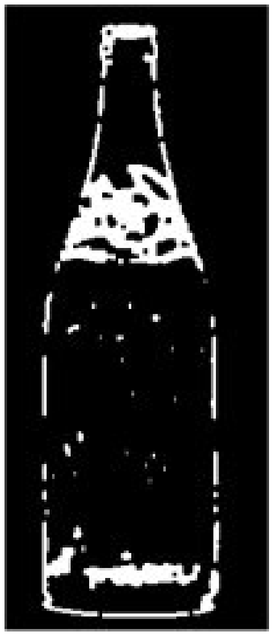


Figure 3.21

3.4.2 Thinning / Outlining: *(Reference Point 12)*

Thinning in its purist form is by taking the set theoretical difference between the eroded image and the opening of the eroded image.

While reading **Zhang and Suen** they apply the algorithm based on two passes through the image for all iteration. In each pass a 3 x 3 window extracts pixel from input image. This then becomes the basis for thinning in the project. In the first pass we scan for deletion of the extra pixels not forming consistency in between green blue and red. And then we place 1 and 0 for different 0 to 255 tones. Later we subtract the image as a difference from erosion.

Results:

Grey Level Classification

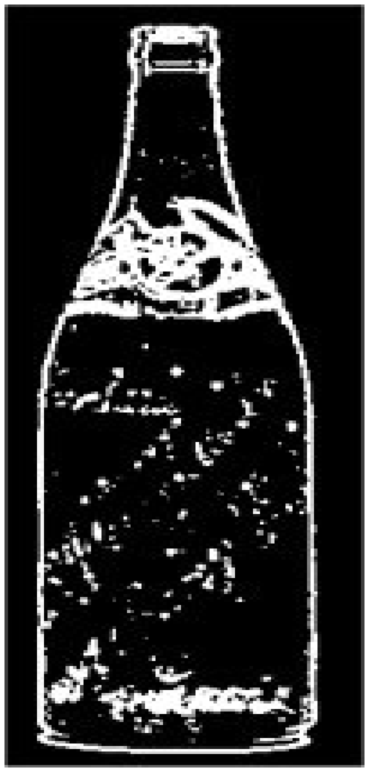


Figure 3.22

Thin Image

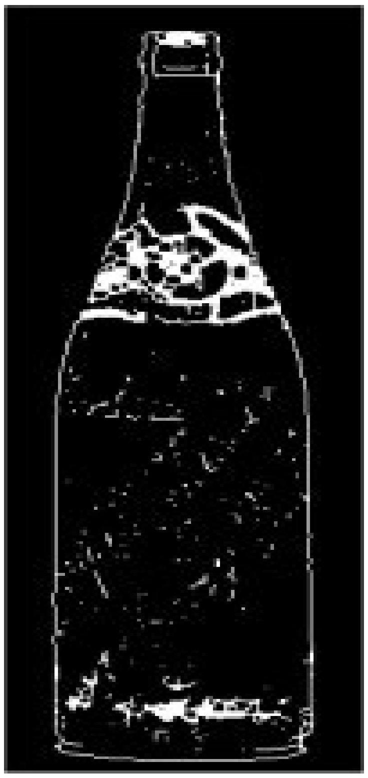


Figure 3.23

3.4.3 Contours*(Reference Point 13)*

These are extensions to the Outlining operations. They can take the following forms

- Inside Contour which is the difference between the image and the erosion.
- Outside Contour is the difference between the dilation and the image itself.
- Middle Contour is the difference between the dilation and the erosion.

Results

Original **Inside** **Middle** **Outside**

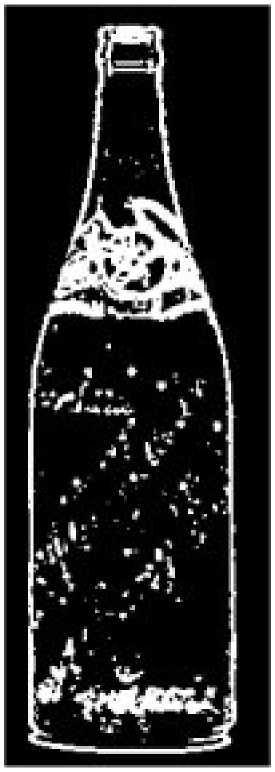


Figure 3.24

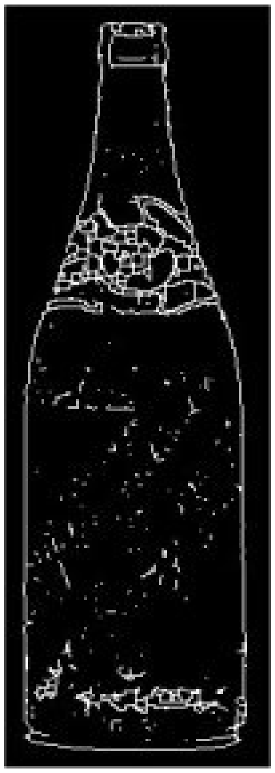


Figure 3.25

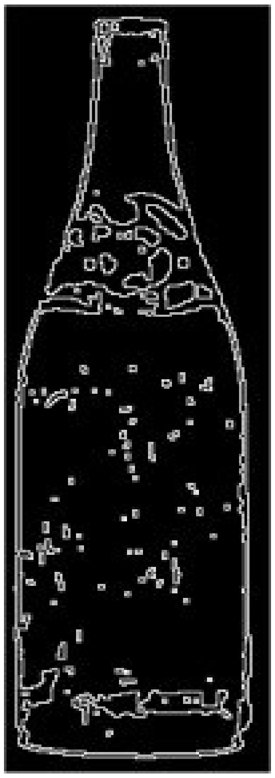


Figure 3.26

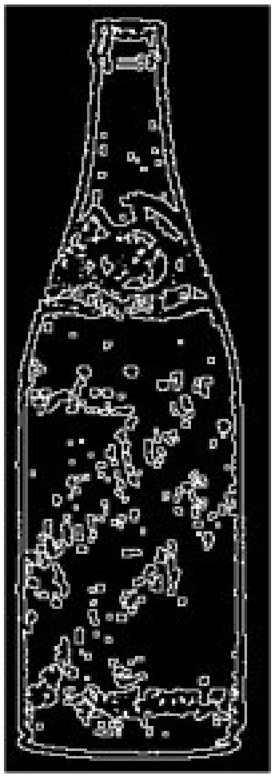


Figure 3.27

By now we have discussed almost all the operations that are involved in attaining the objective of this project. However we have not discussed the ones that are in the software itself. I did quite a lot research on them as well. However I will mention those operations in brief.

- Add, Subtract, Divide, Multiply
- XOR, OR, NOR
- Blur
- Sharpen
- Diffuse
- Invert
- Gaussian Blur
- Median Ranks
- Skeleton (distance transformation)
- Line Segment (Horizontal, Vertical, Left to Right, Right to Left)
- Re sampling.

This is where we end our chapter for the academic research done. However as this was a research oriented project much of the analysis also covers the research set. We have discussed the major operations needed however what we have not discussed is the Object Recognition system. How does the computer see? And how will this system classify the objects, measure them etc. All of this will be continued in the chapter of Analysis. Where we will discuss algorithms and practices in the practical context.