Quality Control for ICs production using camera with C# program (Machine Vision)

Table of contents:

1.0 INTRODUCTION

Machine vision is the current technology used to provide imaging-based automatic inspection and analysis for applications. This includes automatic inspection, process control, and robot guidance in industry. The scope of this technology is wide. The primary uses for machine vision are automatic inspection for product manufacturing and industrial robot control. Commonly, machine vision are applied in applications such as quality assurance, sorting, material handling, robot guidance, and optical gauging.

In this experiment job scope, students are required to design a machine vision software that can inspect product quality through the use of image processing. Images are processed by comparing to the reference image of accepted quality. These can be done by using several methods and techniques that later will be discussed in details in methodology part.

1.1 OBJECTIVES

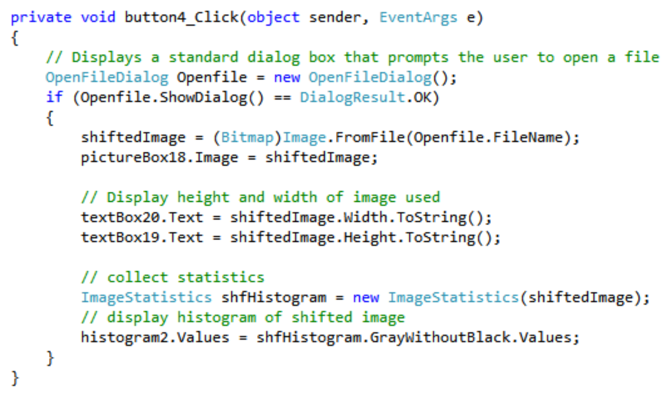
Trimester 1 15/16, under the subject of Machine Vision ERT 3046 we were given an assignment covering all chapters from the subject. The assignment is done via Visual C# .NET. In addition, 3rd party library such as Aforge.NET, Accord.NET, EmguCV and etc. are referred in order to complete our assignment. The topic is algorithm development in nature. Below are the objectives to meet:

- Apply binary image processing and feature extraction techniques. (a)
- Apply segmentation and contour extraction. (b)
- Analyse texture patterns. (c)
- Evaluate the performance of classification process. (d)
- Locate four side edges and form a rectangle. (e)
- Calculate the different in position and angle shift respect to reference image. (f)
- Perform contrast analysis and shape recognition on bright round area. (g)

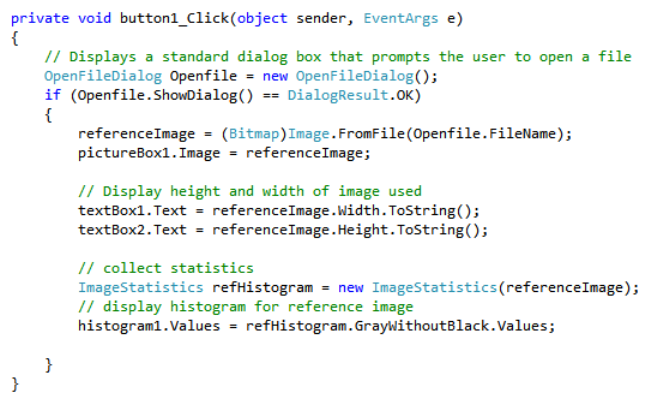
2.0 METHODOLOGY

In this portion, the codes and the way of our codes work will be explained in brief.

1. The UPLOAD buttons



```
private void button4_Click(object sender, EventArgs e)
{
    // Displays a standard dialog box that prompts the user to open a file
    OpenFileDialog Openfile = new OpenFileDialog();
    if (Openfile.ShowDialog() == DialogResult.OK)
    {
        shiftedImage = (Bitmap)Image.FromFile(Openfile.FileName);
        pictureBox18.Image = shiftedImage;

        // Display height and width of image used
        textBox20.Text = shiftedImage.Width.ToString();
        textBox19.Text = shiftedImage.Height.ToString();

        // collect statistics
        ImageStatistics shfHistogram = new ImageStatistics(shiftedImage);
        // display histogram of shifted image
        histogram2.Values = shfHistogram.GrayWithoutBlack.Values;
    }
}
```

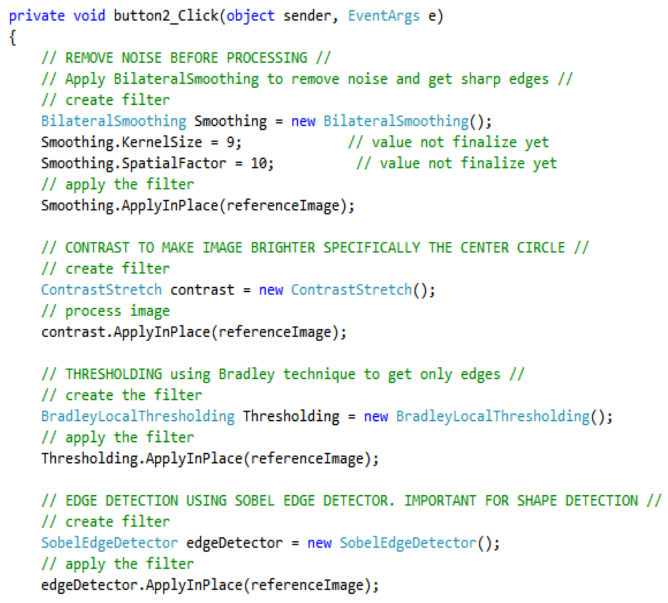


```
private void button1_Click(object sender, EventArgs e)
{
    // Displays a standard dialog box that prompts the user to open a file
    OpenFileDialog Openfile = new OpenFileDialog();
    if (Openfile.ShowDialog() == DialogResult.OK)
    {
        referenceImage = (Bitmap)Image.FromFile(Openfile.FileName);
        pictureBox1.Image = referenceImage;

        // Display height and width of image used
        textBox1.Text = referenceImage.Width.ToString();
        textBox2.Text = referenceImage.Height.ToString();

        // collect statistics
        ImageStatistics refHistogram = new ImageStatistics(referenceImage);
        // display histogram for reference image
        histogram1.Values = refHistogram.GrayWithoutBlack.Values;

    }
}
```

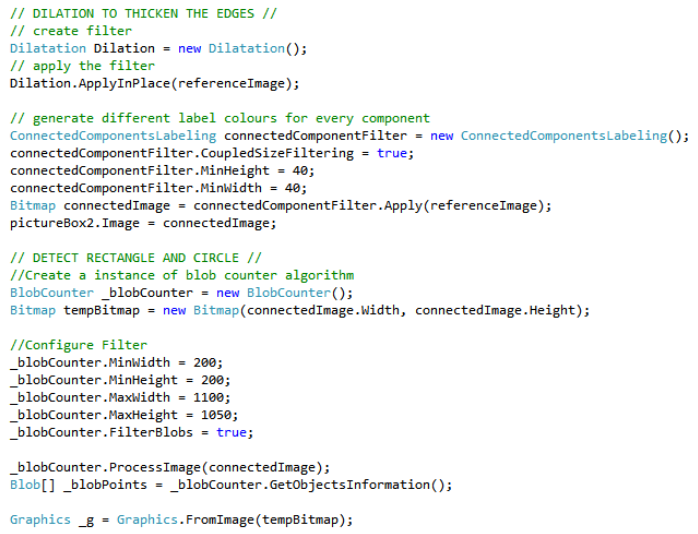
Figure 2.1 Upload button

Both of the image refer to the UPLOAD button execution. The first image describes the UPLOAD button for Reference Image and the second image is for Shifted Image. After each buttons are pressed, user will be asked to select images from folders and after selecting the images, histogram of GrayWithoutBlack will appear for both images. The histograms are used for indication of the grayscale images intensity without black colour. We tried to avoid using histogram of gray and black colour together because black colour happened to produce unwanted spikes on our histogram and hence prevents us from seeing the real representation of our images.

2. The PROCESS IMAGE buttons



```
private void button2_Click(object sender, EventArgs e)
{
    // REMOVE NOISE BEFORE PROCESSING //
    // Apply BilateralSmoothing to remove noise and get sharp edges //
    // create filter
    BilateralSmoothing Smoothing = new BilateralSmoothing();
    Smoothing.KernelSize = 9;            // value not finalize yet
    Smoothing.SpatialFactor = 10;         // value not finalize yet
    // apply the filter
    Smoothing.ApplyInPlace(referenceImage);

    // CONTRAST TO MAKE IMAGE BRIGHTER SPECIFICALLY THE CENTER CIRCLE //
    // create filter
    ContrastStretch contrast = new ContrastStretch();
    // process image
    contrast.ApplyInPlace(referenceImage);

    // THRESHOLDING using Bradley technique to get only edges //
    // create the filter
    BradleyLocalThresholding Thresholding = new BradleyLocalThresholding();
    // apply the filter
    Thresholding.ApplyInPlace(referenceImage);

    // EDGE DETECTION USING SOBEL EDGE DETECTOR. IMPORTANT FOR SHAPE DETECTION //
    // create filter
    SobelEdgeDetector edgeDetector = new SobelEdgeDetector();
    // apply the filter
    edgeDetector.ApplyInPlace(referenceImage);
```



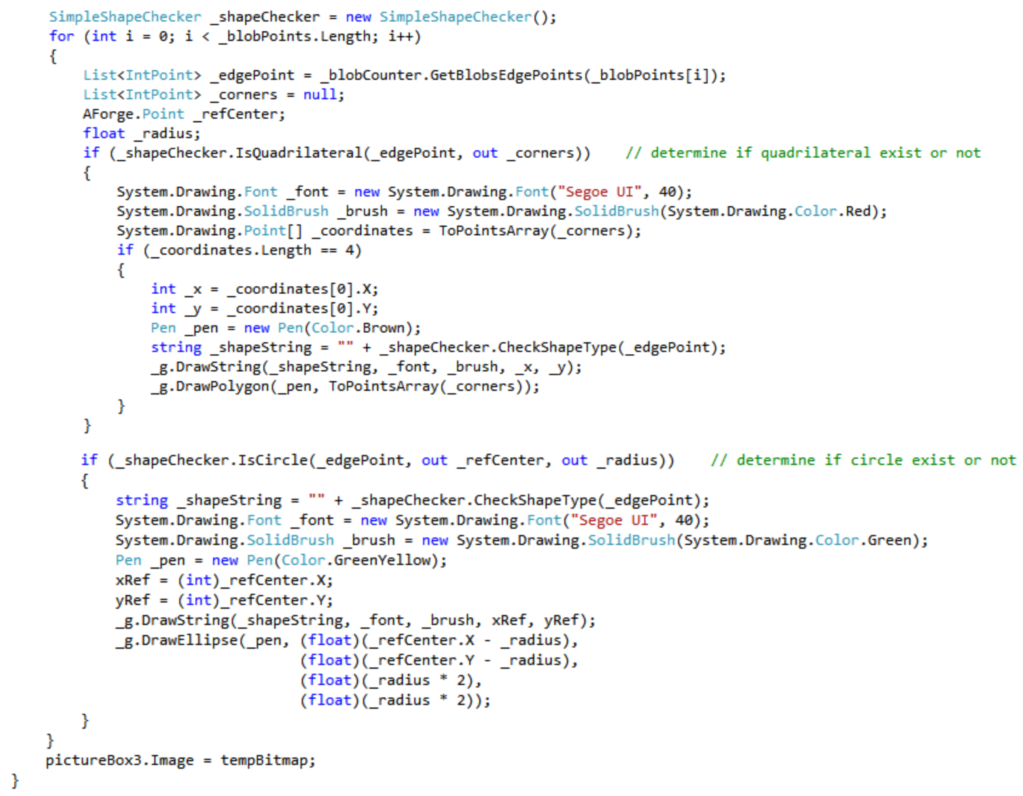
```
    // DILATION TO THICKEN THE EDGES //
    // create filter
    Dilatation Dilation = new Dilatation();
    // apply the filter
    Dilation.ApplyInPlace(referenceImage);

    // generate different label colours for every component
    ConnectedComponentsLabeling connectedComponentFilter = new ConnectedComponentsLabeling();
    connectedComponentFilter.CoupledSizeFiltering = true;
    connectedComponentFilter.MinHeight = 40;
    connectedComponentFilter.MinWidth = 40;
    Bitmap connectedImage = connectedComponentFilter.Apply(referenceImage);
    pictureBox2.Image = connectedImage;

    // DETECT RECTANGLE AND CIRCLE //
    //Create a instance of blob counter algorithm
    BlobCounter _blobCounter = new BlobCounter();
    Bitmap tempBitmap = new Bitmap(connectedImage.Width, connectedImage.Height);

    //Configure Filter
    _blobCounter.MinWidth = 200;
    _blobCounter.MinHeight = 200;
    _blobCounter.MaxWidth = 1100;
    _blobCounter.MaxHeight = 1050;
    _blobCounter.FilterBlobs = true;

    _blobCounter.ProcessImage(connectedImage);
    Blob[] _blobPoints = _blobCounter.GetObjectsInformation();

    Graphics _g = Graphics.FromImage(tempBitmap);
```

Figure 2.2 Process Image buttons

After uploading images, we proceed to process the images to fulfill the requirements listed. The algorithm in Figure 1.2 starts from:

1. Remove noises using ***Bilateral Smoothing Filter***. Bilateral filter conducts "selective" Gaussian smoothing of areas of same colour (domains) which removes noise and contrast artifacts while preserving sharp edges. We use KernelSize of 9 and SpatialFactor of 10 to get what the results that we wanted. *Achieve (a)
2. Make the images brighter using ***Contrast Stretch Filter***. It is a simple image enhancement technique that attempts to improve the contrast in an image by 'stretching' the range of intensity values it contains to span a desired range of values. *Achieve (g)
3. Threshold the images using ***Bradley Local Thresholding*** technique. Every image's pixel is set to black if its brightness is *t* percent lower than the average brightness of surrounding pixels in the window of the specified size, otherwise it is set to white. We use this as it threshold the line to be the object and that is enough for us to do the processing later. *Achieve (a)
4. Detect edges of shape in images using ***Sobel Edge Detection*** method and form shapes. The filter searches for objects' edges by applying Sobel operator. Each pixel of the result image is calculated as approximated absolute gradient magnitude for corresponding pixel of the source image. Besides, the *Sobel Edge Detection* gives us thicker edges formation. *Achieve (b) and (e)
5. Make the lines of shape thicker using ***Dilation*** filter. The filter assigns maximum value of surrounding pixels to each pixel of the result image. Surrounding pixels, which should be processed, are specified by structuring element: 1 - to process the neighbor, -1 - to skip it. *Achieve (b)
6. Separate each component and label them with different colours using ***Connected Component Labeling*** method. The filter performs labeling of objects in the source image. It colors each separate object using different color. The image processing filter treats all none black pixels as objects' pixels and all black pixel as background. *Achieve (b)
7. Extract standalone objects in images using ***BlobCounter*** filter. We extract only the biggest rectangle from images by specifying the boundary needed. *Achieved (e) and (g)
8. From the extracted objects, perform quadrilateral and circle detection using ***SimpleShapeChecker*** class for provided set of points (shape's edge points). During the check the class goes through the list of all provided points and checks how accurately they fit into assumed shape. *Achieved (e) and (g)

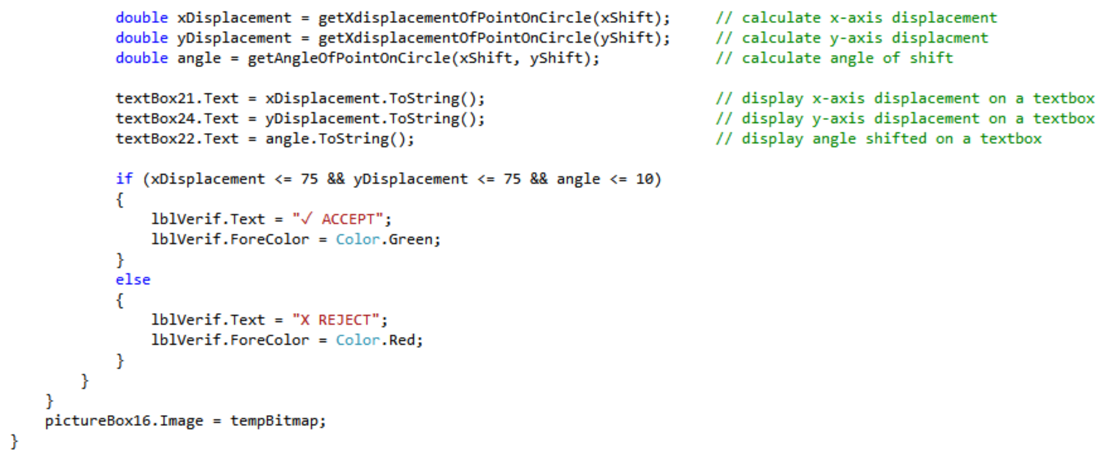
9. Using the centre coordinates of circles detected from reference and shifted images, perform image position shifting and angle shifting calculation. *Achieved (f)
10. Extract the images features using ***GrayLevelCooccurrenceMatrix*** Class. The class used can capture properties of a texture. Features extracted are the values of entropy, energy, homogeneity and contrast. *Achieve (c) and (d)



```
    SimpleShapeChecker _shapeChecker = new SimpleShapeChecker();
    for (int i = 0; i < _blobPoints.Length; i++)
    {
        List<IntPoint> _edgePoint = _blobCounter.GetBlobsEdgePoints(_blobPoints[i]);
        List<IntPoint> _corners = null;
        AForge.Point _refCenter;
        float _radius;
        if (_shapeChecker.IsQuadrilateral(_edgePoint, out _corners))    // determine if quadrilateral exist or not
        {
            System.Drawing.Font _font = new System.Drawing.Font("Segoe UI", 40);
            System.Drawing.SolidBrush _brush = new System.Drawing.SolidBrush(System.Drawing.Color.Red);
            System.Drawing.Point[] _coordinates = ToPointsArray(_corners);
            if (_coordinates.Length == 4)
            {
                int _x = _coordinates[0].X;
                int _y = _coordinates[0].Y;
                Pen _pen = new Pen(Color.Brown);
                string _shapeString = "" + _shapeChecker.CheckShapeType(_edgePoint);
                _g.DrawString(_shapeString, _font, _brush, _x, _y);
                _g.DrawPolygon(_pen, ToPointsArray(_corners));
            }
        }

        if (_shapeChecker.IsCircle(_edgePoint, out _refCenter, out _radius))    // determine if circle exist or not
        {
            string _shapeString = "" + _shapeChecker.CheckShapeType(_edgePoint);
            System.Drawing.Font _font = new System.Drawing.Font("Segoe UI", 40);
            System.Drawing.SolidBrush _brush = new System.Drawing.SolidBrush(System.Drawing.Color.Green);
            Pen _pen = new Pen(Color.GreenYellow);
            xRef = (int)_refCenter.X;
            yRef = (int)_refCenter.Y;
            _g.DrawString(_shapeString, _font, _brush, xRef, yRef);
            _g.DrawEllipse(_pen, (float)(_refCenter.X - _radius),
                                 (float)(_refCenter.Y - _radius),
                                 (float)(_radius * 2),
                                 (float)(_radius * 2));
        }
    }
    pictureBox3.Image = tempBitmap;
}
```

Figure 2.3 Shape Detection Algorithm

From Figure 1.3, shape detection is performed to determine quadrilateral shape and circle shape. Both UPLOAD buttons are using the same algorithm. However for shifted image processing, we include the calculation for determining the position and angle shifted.

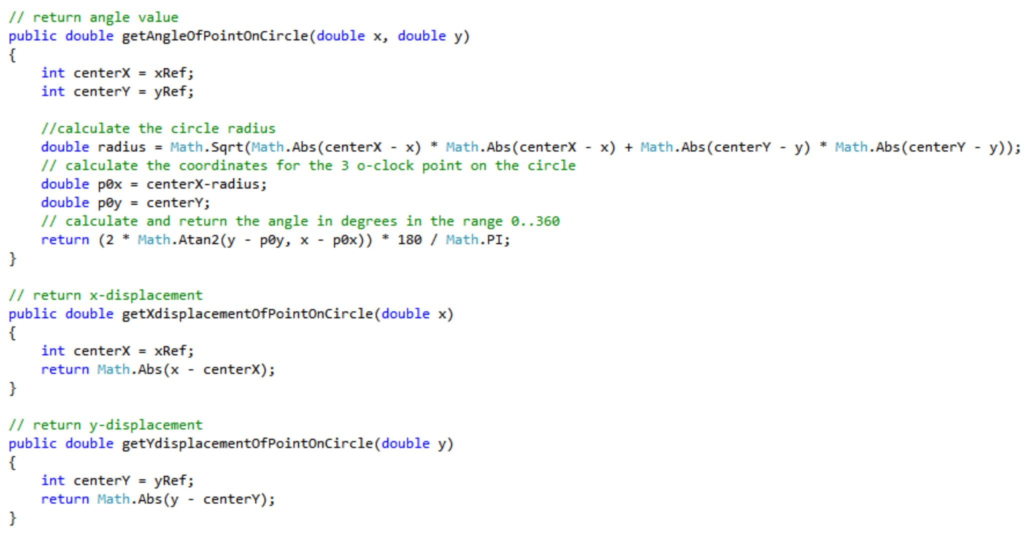


```
            double xDisplacement = getXdisplacementOfPointOnCircle(xShift);     // calculate x-axis displacement
            double yDisplacement = getXdisplacementOfPointOnCircle(yShift);     // calculate y-axis displacment
            double angle = getAngleOfPointOnCircle(xShift, yShift);             // calculate angle of shift

            textBox21.Text = xDisplacement.ToString();                          // display x-axis displacement on a textbox
            textBox24.Text = yDisplacement.ToString();                          // display y-axis displacement on a textbox
            textBox22.Text = angle.ToString();                                  // display angle shifted on a textbox

            if (xDisplacement <= 75 && yDisplacement <= 75 && angle <= 10)
            {
                lblVerif.Text = "√ ACCEPT";
                lblVerif.ForeColor = Color.Green;
            }
            else
            {
                lblVerif.Text = "X REJECT";
                lblVerif.ForeColor = Color.Red;
            }
        }
    }
    pictureBox16.Image = tempBitmap;
}
```

Figure 2.4 Position and Angle Calculation

Figure 1.4 show that calculation is being done and it will be displayed using textboxes. We check if the x - y axis and angle shifting not more than 75 pixels, 75 pixels and 10 degrees, the image can be accepted. It depends on manufacturers to adjust the conditions that they need in order to match their product requirements.

3.0 Calculation Formula



```
// return angle value
public double getAngleOfPointOnCircle(double x, double y)
{
    int centerX = xRef;
    int centerY = yRef;

    //calculate the circle radius
    double radius = Math.Sqrt(Math.Abs(centerX - x) * Math.Abs(centerX - x) + Math.Abs(centerY - y) * Math.Abs(centerY - y));
    // calculate the coordinates for the 3 o-clock point on the circle
    double p0x = centerX-radius;
    double p0y = centerY;
    // calculate and return the angle in degrees in the range 0..360
    return (2 * Math.Atan2(y - p0y, x - p0x)) * 180 / Math.PI;
}

// return x-displacement
public double getXdisplacementOfPointOnCircle(double x)
{
    int centerX = xRef;
    return Math.Abs(x - centerX);
}

// return y-displacement
public double getYdisplacementOfPointOnCircle(double y)
{
    int centerY = yRef;
    return Math.Abs(y - centerY);
}
```

Figure 2.5 Formula for Position and Angle Shifting

Figure 1.5 illustrated the simple formula that we used in order to determine the position and angle shifting. The x and y reference point we took it from the centre coordinate of reference image circle. The formula that we used are as follows:

For angle shifted calculation:

1. $radius = \sqrt{(|x_{ref} - x_{shift}|)^2 + (|y_{ref} - y_{shift}|)^2}$
2. Start from 3 o'clock axis, we calculate
    a. $x = radius - x_{ref}$
    b. $y = y_{ref}$
3. $Radian = Atan2(y_{shift} - y, x_{shift} - x)$
4. $Angle\ shifted = Radian\left(\frac{360}{\pi}\right)$

For position shifted calculation:

1. $x_{displacement} = x_{shift} - x_{ref}$
2. $y_{displacement} = y_{shift} - y_{ref}$

4. Texture Analysis

```
private void button7_Click(object sender, EventArgs e)      // display reference image properties on textboxes
{
    double entropy, energy, contrast, homogeneity;

    AnalyseBitmapTexture(referenceImage, out entropy, out energy, out contrast, out homogeneity);
    ImageStatistics stats = new ImageStatistics(referenceImage);

    textBox3.Text = entropy.ToString();
    textBox4.Text = energy.ToString();
    textBox5.Text = homogeneity.ToString();
    textBox6.Text = contrast.ToString();
    textBox7.Text = xRef.ToString();
    textBox8.Text = yRef.ToString();
    textBox10.Text = stats.Gray.Values.Average().ToString();
}

private void button6_Click(object sender, EventArgs e)      // display shifted image properties on textboxes
{
    double entropy, energy, contrast, homogeneity;

    AnalyseBitmapTexture(shiftedImage, out entropy, out energy, out contrast, out homogeneity);
    ImageStatistics stats = new ImageStatistics(shiftedImage);

    textBox18.Text = entropy.ToString();
    textBox17.Text = energy.ToString();
    textBox16.Text = homogeneity.ToString();
    textBox15.Text = contrast.ToString();
    textBox14.Text = xShift.ToString();
    textBox13.Text = yShift.ToString();
    textBox11.Text = stats.Gray.Values.Average().ToString();
}
```

Figure 2.6 Texture Properties Displayed in Textboxes

Figure 1.7 shows some formula that we used to calculate the value of entropy, energy, homogeneity and contrast. These formula are: *Achieve (d)

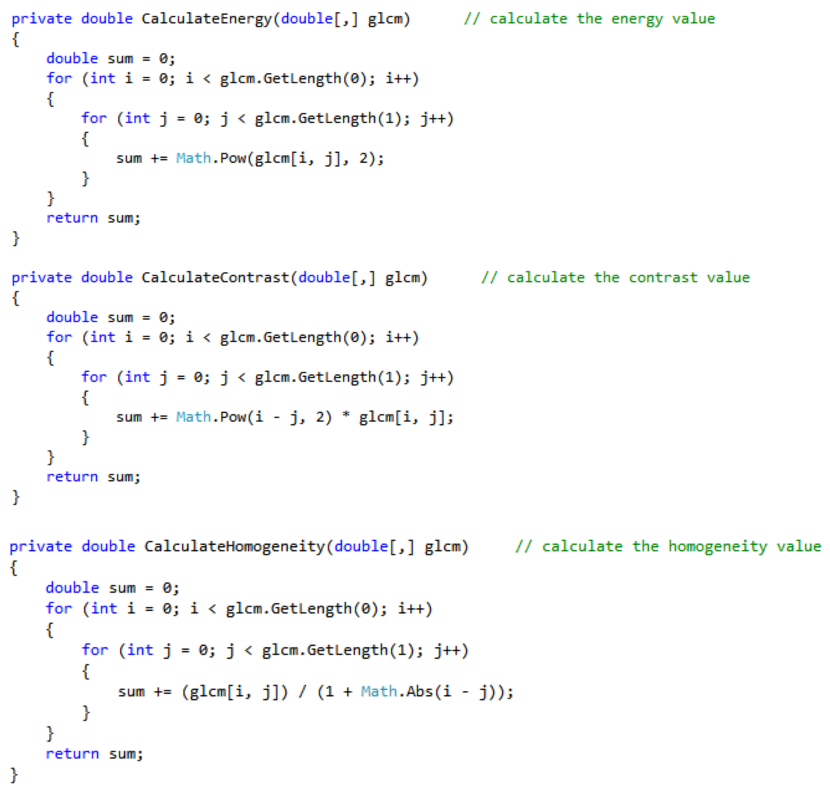
$$Entropy = -\sum_i \sum_j P[i,j] \log_2 P[i,j].$$

$$Energy = \sum_i \sum_j P^2[i,j].$$

$$Contrast = \sum_i \sum_j (i-j)^2 P[i,j].$$

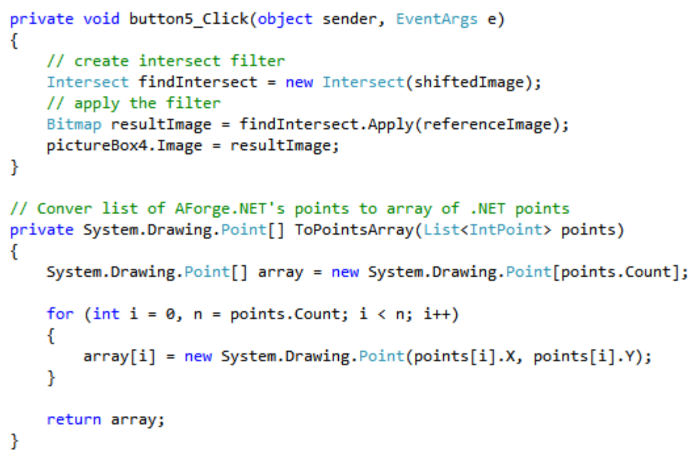
$$Homogeneity = \sum_i \sum_j \frac{P[i,j]}{1+|i-j|}.$$

```
private void AnalyseBitmapTexture(Bitmap bitmap, out double entropy, out double energy, out double contrast, out double homogeneity)
{
    GrayLevelCooccurrenceMatrix matrix = new GrayLevelCooccurrenceMatrix();
    double[,] glcm = matrix.Compute(UnmanagedImage.FromManagedImage(bitmap));
    entropy = CalculateEntropy(glcm);
    energy = CalculateEnergy(glcm);
    contrast = CalculateContrast(glcm);
    homogeneity = CalculateHomogeneity(glcm);
}

private double CalculateEntropy(double[,] glcm)     // calculate the entrophy value
{
    double sum = 0;
    for (int i = 0; i < glcm.GetLength(0); i++)
    {
        for (int j = 0; j < glcm.GetLength(1); j++)
        {
            if (Math.Abs(glcm[i, j]) > 0.0001)
                sum += glcm[i, j] * Math.Log(glcm[i, j], 2);
        }
    }
    return -sum;
}
```



```
private double CalculateEnergy(double[,] glcm)      // calculate the energy value
{
    double sum = 0;
    for (int i = 0; i < glcm.GetLength(0); i++)
    {
        for (int j = 0; j < glcm.GetLength(1); j++)
        {
            sum += Math.Pow(glcm[i, j], 2);
        }
    }
    return sum;
}

private double CalculateContrast(double[,] glcm)      // calculate the contrast value
{
    double sum = 0;
    for (int i = 0; i < glcm.GetLength(0); i++)
    {
        for (int j = 0; j < glcm.GetLength(1); j++)
        {
            sum += Math.Pow(i - j, 2) * glcm[i, j];
        }
    }
    return sum;
}

private double CalculateHomogeneity(double[,] glcm)     // calculate the homogeneity value
{
    double sum = 0;
    for (int i = 0; i < glcm.GetLength(0); i++)
    {
        for (int j = 0; j < glcm.GetLength(1); j++)
        {
            sum += (glcm[i, j]) / (1 + Math.Abs(i - j));
        }
    }
    return sum;
}
```

Figure 2.7 Calculation Formula for Texture Analysis



```
private void button5_Click(object sender, EventArgs e)
{
    // create intersect filter
    Intersect findIntersect = new Intersect(shiftedImage);
    // apply the filter
    Bitmap resultImage = findIntersect.Apply(referenceImage);
    pictureBox4.Image = resultImage;
}

// Conver list of AForge.NET's points to array of .NET points
private System.Drawing.Point[] ToPointsArray(List<IntPoint> points)
{
    System.Drawing.Point[] array = new System.Drawing.Point[points.Count];

    for (int i = 0, n = points.Count; i < n; i++)
    {
        array[i] = new System.Drawing.Point(points[i].X, points[i].Y);
    }

    return array;
}
```

Figure 2.8 Intersect the Image Button

Figure 1.8 show the Intersect filter that we used to check the image result after we are trying to intersect them. If the image turns out with very smooth rectangle and circle, we can consider the image to be accepted and vice versa.

3.0 RESULTS AND DISCUSSION

When starting the program, a graphical user interface (GUI) will start (Fig.4.1).

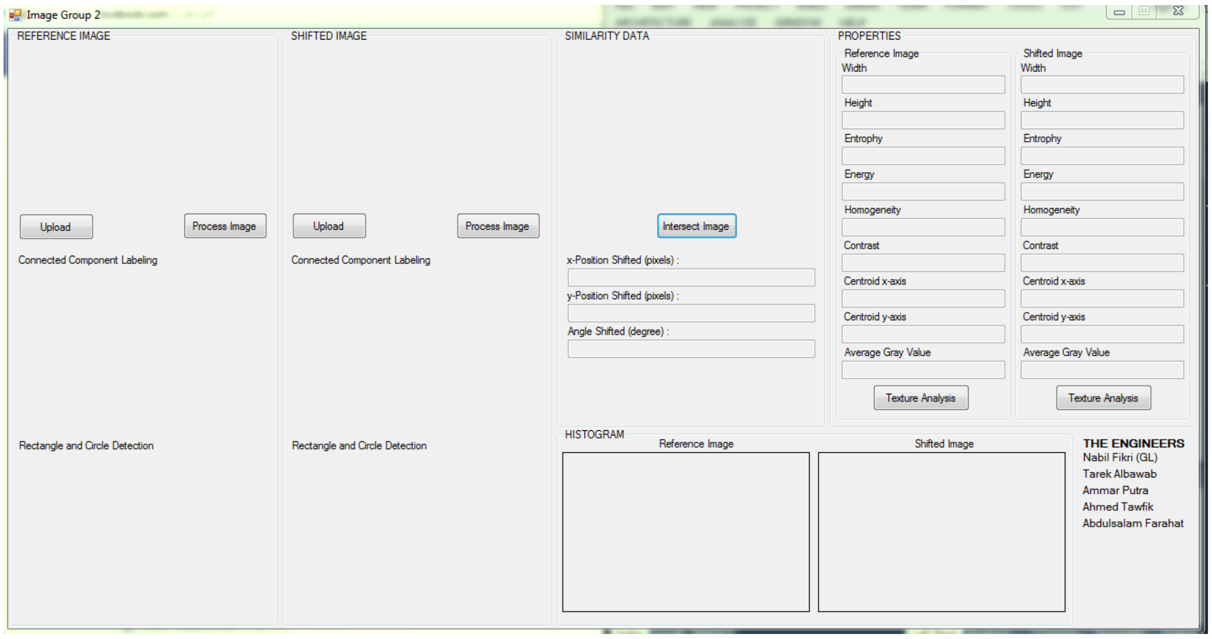


Fig.3.1

Press Upload button to upload the images that you want to start the processing (Fig.4.2).

Two kinds of images, Reference Image and Shifted Image. The image that you want to set it as the main image for your processing, is called the Reference image.

The image that you want to be processed and compared with the Reference image is called Shifted Image.

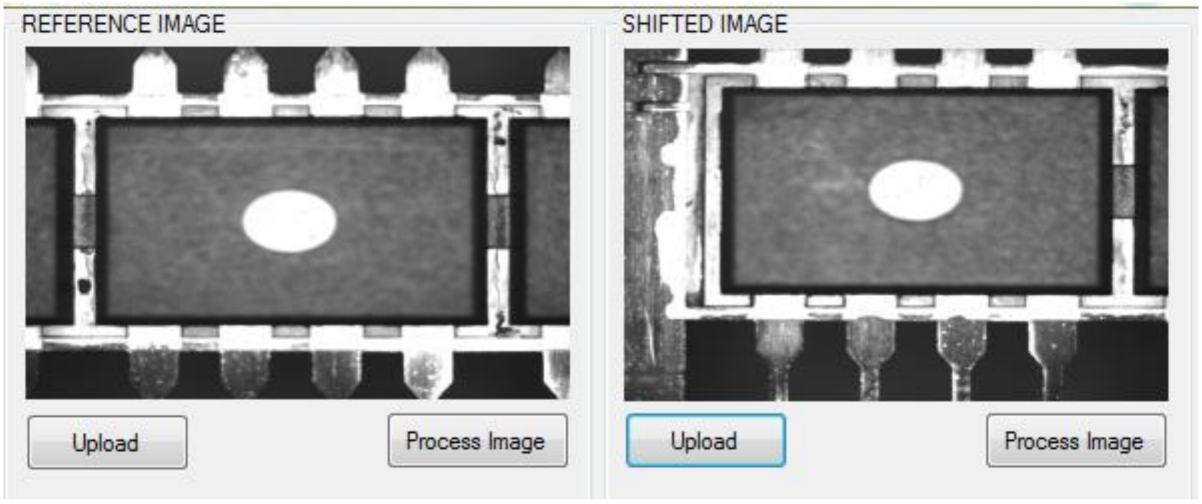


Fig.3.2

If Process Image button is pressed, the image will be processed, the processing will detect the Rectangle and Circle in the image (Fig.4.3).

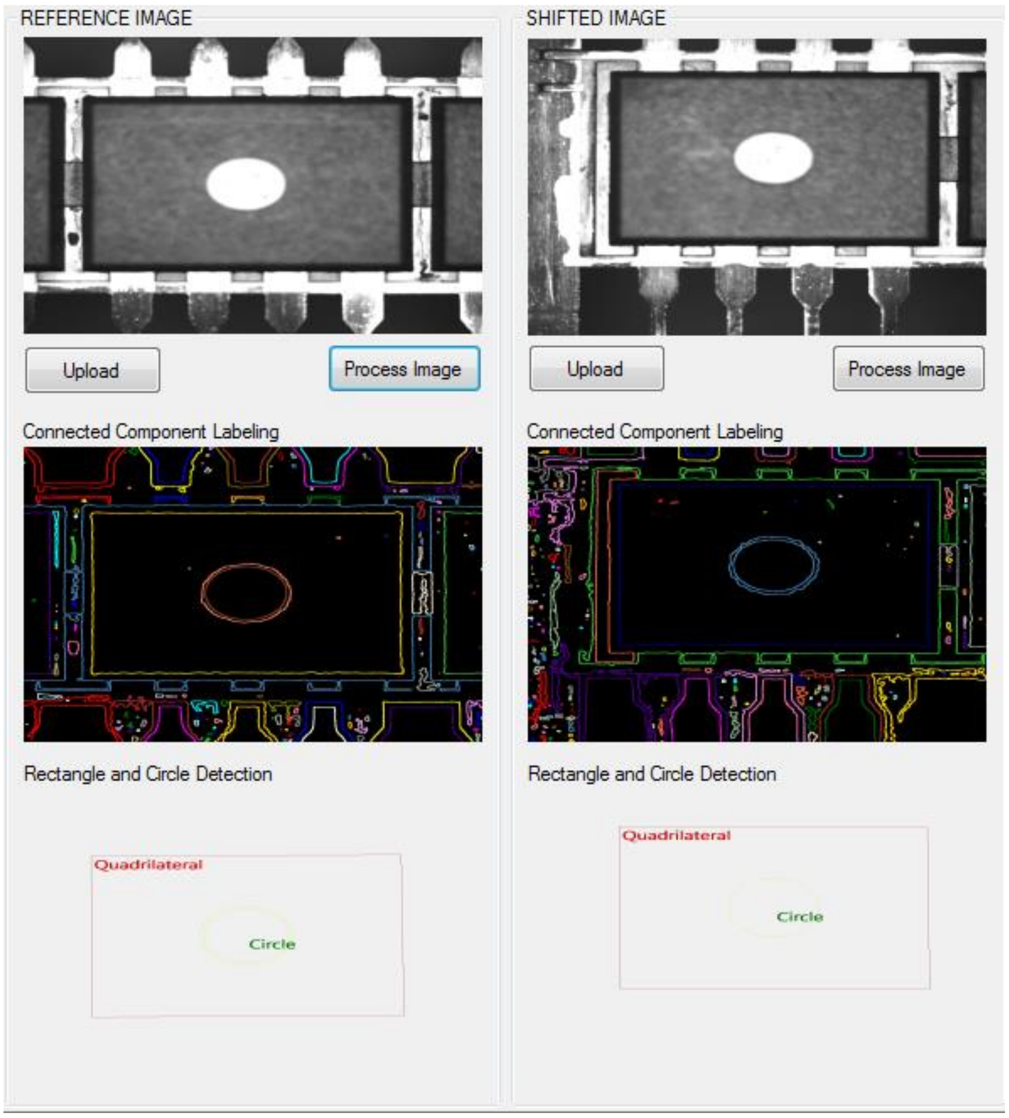
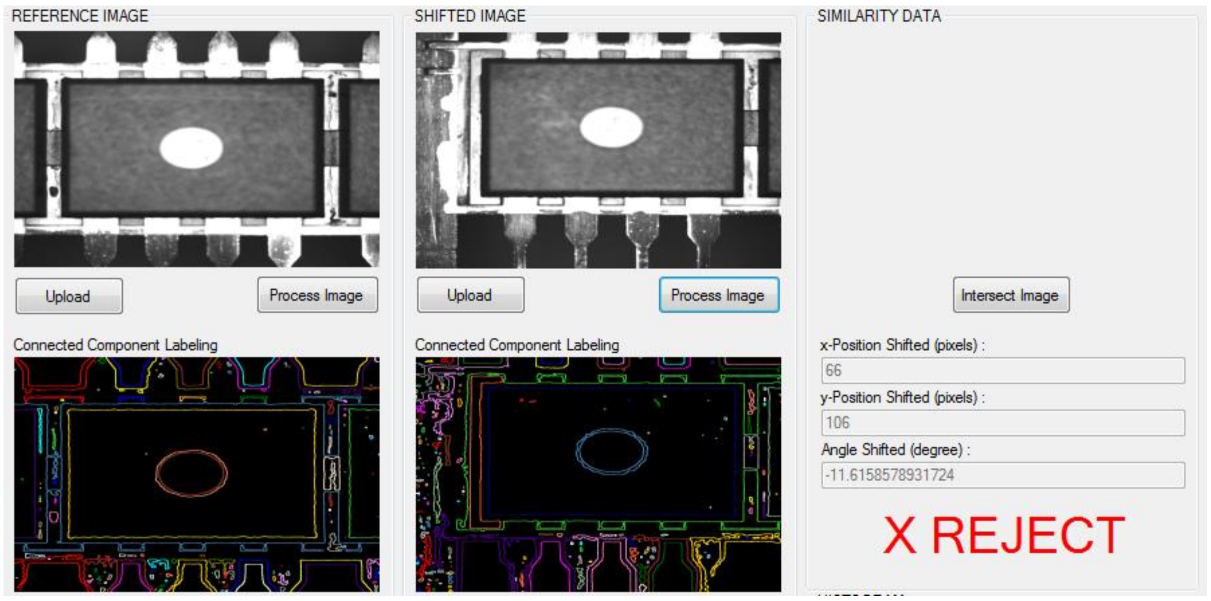

REFERENCE IMAGE
SHIFTED IMAGE
Upload
Process Image
Upload
Process Image
Connected Component Labeling
Connected Component Labeling
Rectangle and Circle Detection
Rectangle and Circle Detection
Quadrilateral
Circle
Quadrilateral
Circle

Fig.3.3

Furthermore, the process image will show if the Shifted Image is accepted or rejected with respect to the Reference Image (Fig.4.4) & (Fig.4.5).



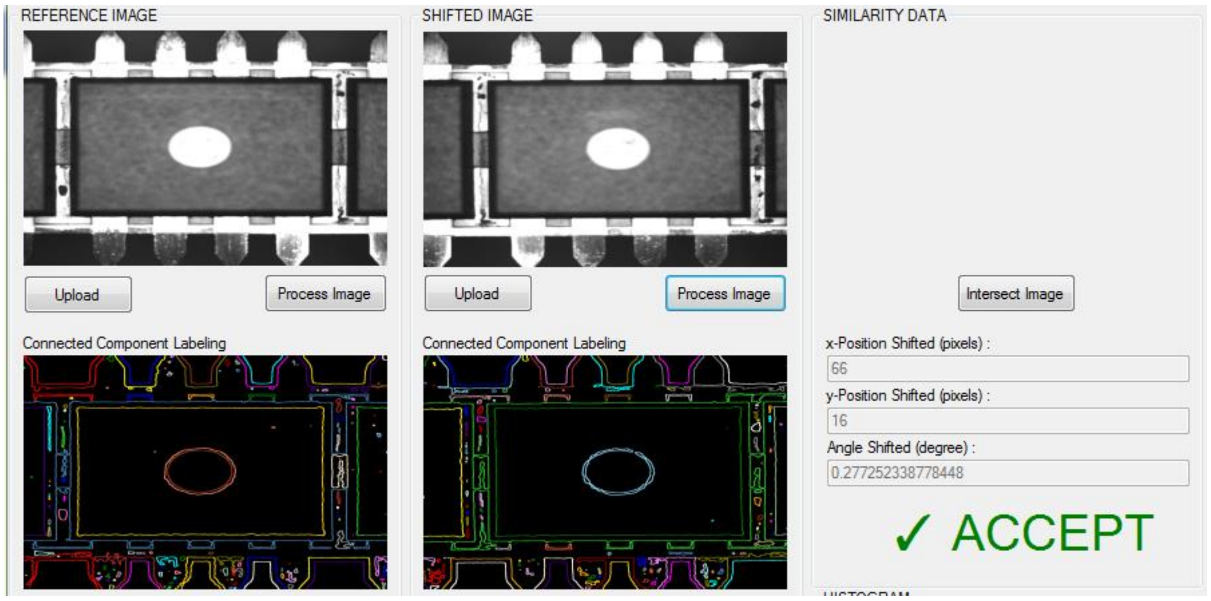
x-Position Shifted (pixels) : 66

y-Position Shifted (pixels) : 106

Angle Shifted (degree) : -11.6158578931724

X REJECT

Fig.3.4



x-Position Shifted (pixels) : 66

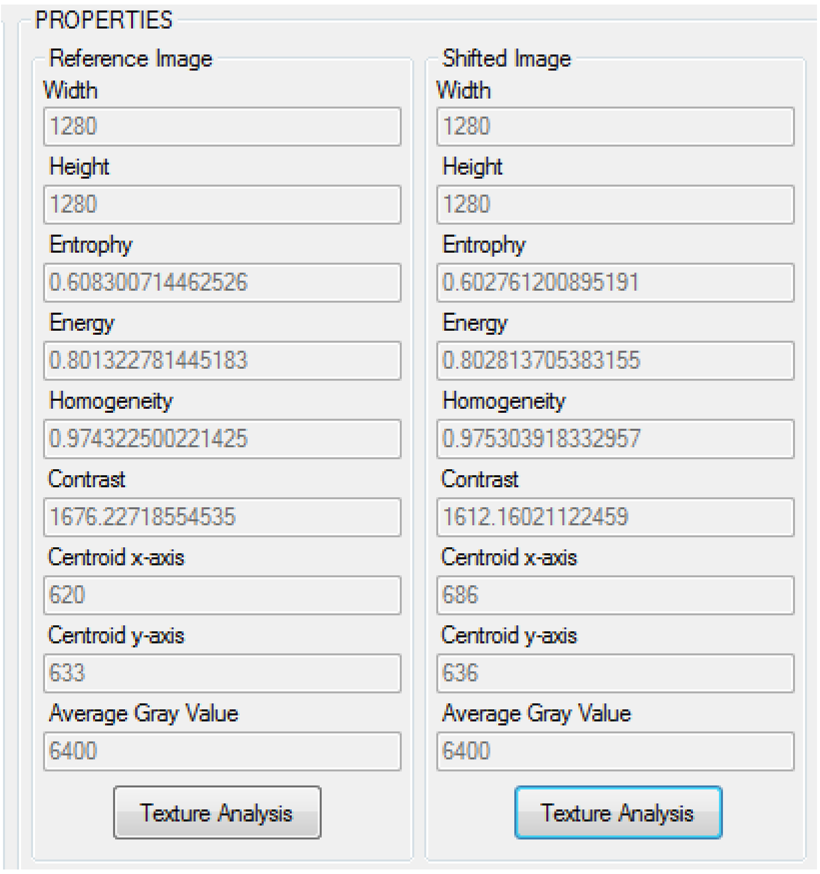
y-Position Shifted (pixels) : 16

Angle Shifted (degree) : 0.277252338778448

✓ ACCEPT

Fig.3.5

When the Texture Analysis button is pressed, properties of the photos will be shown. Properties such as: Entropy, Energy, Homogeneity, Contrast, Centroid x-axis, Centroid y-axis, and Average Grey Value (Fig.4.6).



PROPERTIES

	Reference Image	Shifted Image
Width	1280	1280
Height	1280	1280
Entrophy	0.608300714462526	0.602761200895191
Energy	0.801322781445183	0.802813705383155
Homogeneity	0.974322500221425	0.975303918332957
Contrast	1676.22718554535	1612.16021122459
Centroid x-axis	620	686
Centroid y-axis	633	636
Average Gray Value	6400	6400

Fig.3.6

Program can also show the Homogeneity data (Fig.4.7).

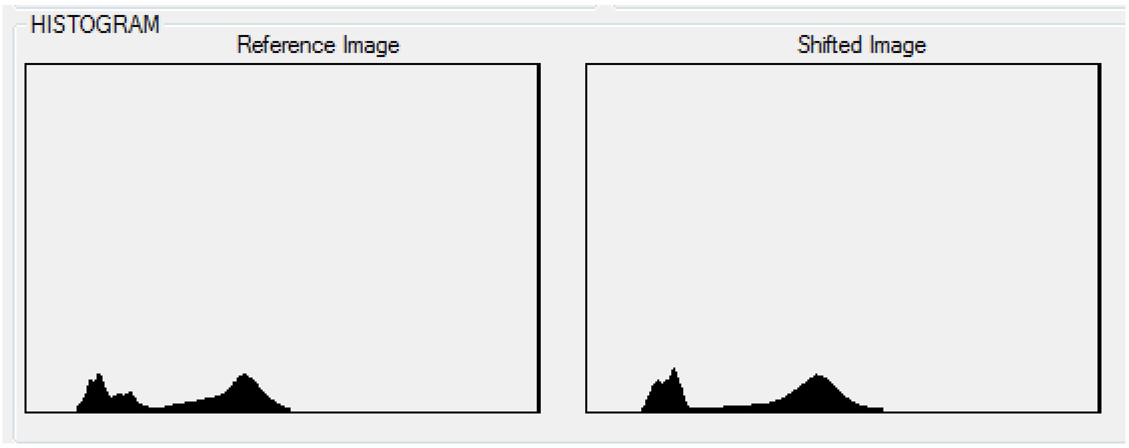


Fig.3.7

In addition to all the features that our program has, an extra feature is added to our program, which is Intersect image.

Intersect image basically shows how similar one image to another by intersect them in one image (Fig.4.8) & (Fig.4.9).

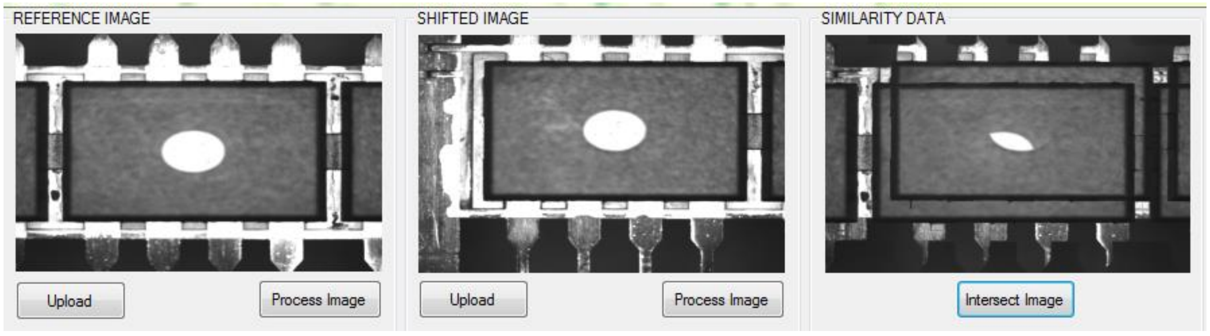


Fig.3.8

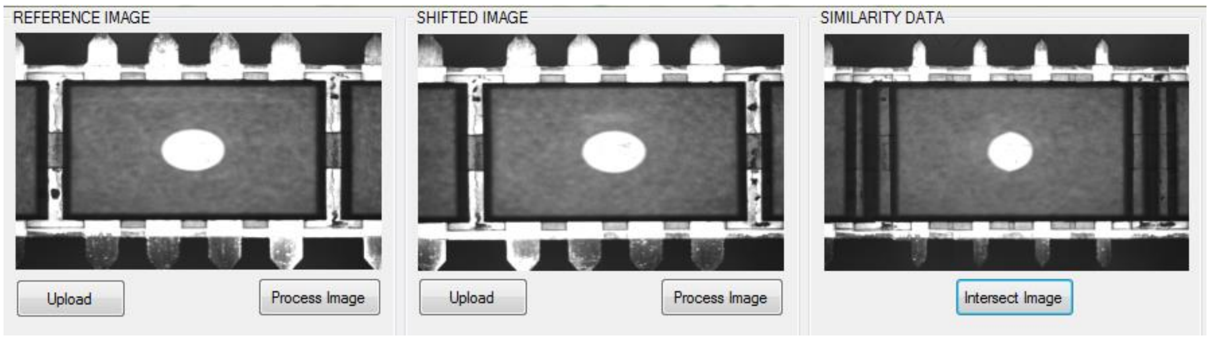


Fig.3.9

4.0 CONCLUSION

Our program gives the ability to a normal camera, to be smart. Writing the codes to enable the processing, which will therefore give the brain to the processor to process real world conditions and generate data and makes actions out of it. In other words, we have achieved the Machine Vision, by giving the Vision to the machine, to compare and analyse between different given data.

We have successfully achieve the objective of this assignment, by detecting the four sides edges with forming a rectangle, calculating the different in position and angle shift respect to reference image, performing contrast analysis and shape recognition on bright round area, further more adding another feature as our enhancement which is the Similarity Data.

5.0 REFERENCES

1. AForge.NET :: Framework Features. (n.d.). Retrieved from http://www.aforgenet.com/framework/features/ (September 14, 2015)

2. Adrian. Calculating the angle between two points on a circle. (2009, March 23). Retrieved from https://beradrian.wordpress.com/2009/03/23/calculating-the-angle-between-two-points-on-a-circle/ (September 14, 2015)

3. Souza, C. (n.d.). Primary objects/Accord.NET. Retrieved from https://github.com/primaryobjects/Accord.NET/blob/master/Sources/Accord.Imaging/GrayLevelCooccurrenceMatrix.cs (September 14, 2015)