Frankfurt University of Applied Sciences

Face Detection Using Machine Learning and FEZ Spider Kit

Project Report – Automation Lab

Department of Engineering FRA-UAS

Summer Semester 2016

Supervised by:

Prof. Dr. A. Pech

Prepared by:

Muhammad Ehsan-Ul-Haq

Matriculation: 1098587

Date: 23/05/2016

Signature: _____

Abstract

Over the past several years Machine Learning has evolved into main subareas of Information Technology and has got considerable importance in solving real life issues. With the advances in Information Technology, more and more data is available at our hands, and there is a good reason to believe, that smart data analysis will become necessary ingredient of technological progress. The main idea of this project was to build a smart application that uses one of the many available machine learning strategies along with .Net gadgeteer FEZ Spider kit, which senses the environment with different sensors. In this project, I selected to work with gadgeteer camera (attached with FEZ Spider kit) which takes pictures of a scene, and on pressing a button, sends them to the web browser using an Ethernet protocol. On the other hand, computer runs a desktop application (written in C# .Net), that gets the picture from browser and runs a machine learning (Viola- Jones) algorithm on it, and decides whether a face is present in the scene or not. Moreover, hardware of the project involves FEZ Spider Kit, gadgeteer camera, TE35 touch screen display, USB client EDP, Ethernet J11D and a button module (see Appendix). Software involves .net micro framework v4.3 and .Net framework 4.0.

Table of Contents

Topic		Page
1	**Part 1: Face detection using machine learning**	2
	1.1 Introduction	
	1.2 Viola-Jones Algorithm	
	1.2.1 Feature Extraction	
	1.2.2 Integral Image Representation	
	1.2.3 Attentional Cascade	
	1.2.3.1 Structure of Attentional Cascade	
	1.2.3.2 AdaBoost Algorithm	
	1.3 Source Code	
	1.4 Building Face Detection App	
2	**Part 2: .Net Gadgeteer**	19
	2.1 Introduction	
	2.2 .NET Micro Framework	
	2.3 Setting Up Development Environment	
	2.4 Building “Ethernet Camera” App	
	2.4.1 Setting Up	
	2.4.2 Coding the Software	
	2.4.3 Running the Application	
3	**Part 3: How to run complete Project?**	26
	3.1 Steps	
	3.2 Conclusion	
4	**Part 4: Further Improvements**	30
5	**References**	31
6	**Appendix**	32

Part 1: Face detection using machine learning

1.1 Introduction

In the field of computer vision, face detection has been regarded as the most complex and challenging problem, due to large intra class variations caused by changes in facial appearances, lightning and expressions. Such variations result in the face distribution to be highly nonlinear and complex in any space which is linear to original image space. Moreover, in the applications of real life surveillance and biometric, the camera limitations and pose variations make the distribution of human faces in feature space more dispersed and complicated than of frontal faces.

For years, face detection techniques have been researched and much progress has been proposed in the literature. Most of the methods focus on detecting frontal faces with good lightning conditions. We can categories them into four types, knowledge based, feature invariant, template matching and appearance based.

Knowledge-based methods use human-coded rules to model facial features, such as two symmetric eyes, a nose in the middle and mouth underneath. Feature invariant methods try to find facial features which stay invariant to lightning condition or pose e.g. edges or skin color. Template matching methods calculate correlation between pre-selected facial template and test image. Whereas, Appearance based methods adopt machine learning techniques to extract distinctive features from a pre-labeled training set [1].

Any of these methods can involve color segmentation, pattern matching complex transforms or statistical analysis, but the common goal is classification with least amount of error. Bounds on classification accuracy changes from method to method yet the best techniques are found in areas where rules for classification are dynamic and produced from machine learning process.

1.2 Viola-Jones algorithm

In 2001, Paul Viola and Michael Jones presented a paper on fast and robust method for face detection which is 15 times quicker than any technique at the time of release with 95% accuracy at around 17fps.

A face detector has to tell whether an image of arbitrary size contains a human face and if so, where it is. On natural framework for considering this problem is that of binary classification, in which a classifier is constructed to minimize the misclassification risk. Since no objective distribution can describe the actual prior probability for a given image to have a face, the algorithm must minimize both the false negative and false positive rate in order to achieve an acceptable performance.

This task requires an accurate numerical description of what sets human faces apart from other objects. It turns out that these characteristics can be extracted with a remarkable committee

learning algorithm called Adaboost, which relies on a committee of weak classifiers to form a strong one through a voting mechanism. A classifier is weak if, in general, it cannot meet a predefined classification target in error terms [2].

An operational algorithm must also work with a reasonable computational budget. Techniques such as integral image and attentional cascade make the Viola-Jones algorithm [3] highly efficient: fed with a real time image sequence generated from a standard webcam, it performs well on a standard PC.

1.2.1 Feature Extraction

Viola-Jones use rectangular Haar-like features which are reminiscent to Haar basis functions. These features are based on intensity difference readings which are quite easy to compute. There are actually three feature types used with varying number of sub rectangles, two based on two rectangles, one on three and one on four rectangles. The main advantage of using rectangular features over pixel is that speed is increased and ad-hoc domain knowledge is implied as well. Also, the calculation of feature is facilitated with "integral image". With the use of integral image Viola-Jones are able to calculate in just one go and speed is drastically improved.

In more simplest form, features are just pixel intensity set evaluations. This is where the sum of pixels of white region are subtracted from sum of pixel of black region. The difference value is used as the feature value and which can be later combined to form a weak hypothesis on the regions of a face. Within the implementation of four Haar features, first with horizontal division, second vertical, third containing two vertical divisions, fourth two horizontal and the last containing both horizontal and vertical divisions as shown in following fig.1.

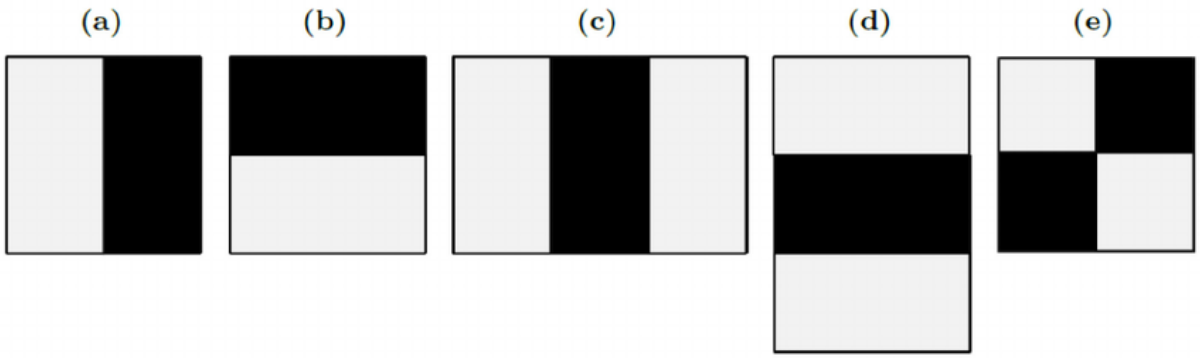


Figure 1: Haar-Like features

Consider one of these features and placed over an image. The value of the feature would be the result of summing the pixels in the white side of the rectangle and summing the pixels in the black side of the rectangle, and the calculating their difference. Hopefully, it's very clear by seeing the image that these features represent the different in the intensity of the pixels of different facial features e.g. eyes are darker than cheeks so feature b and d would be effective

to detect that difference and nose bridge is lighter than the eyes and feature c would be effective to calculate that.

Figure 2:Features applied on facial image (Lena Söderberg's)

The image above also gives an idea how search algorithm works. It starts with either small or large window and scans the image exhaustively by dislocating the search window to right and below till the end of the image. When first scan finishes, it grows window size and repeats all process again.

1.2.2 Integral Image Representation

For a successful face detection, algorithm must have two key feature, first accuracy and second speed, but generally there is a trade-off between these two. Viola-Jones in 2001 described that through the representation of an image in integral form, we can speed up the feature evaluation.

Integral images are normally easy to understand; they are constructed by simply taking the sum of pixel values above and to the left of a pixel in an image. Viola-Jones make note of the fact that integral image is actually the double integral of sample image, first along the rows and then along the columns. Integral images are equivalent to summed-area table as shown in following figure.

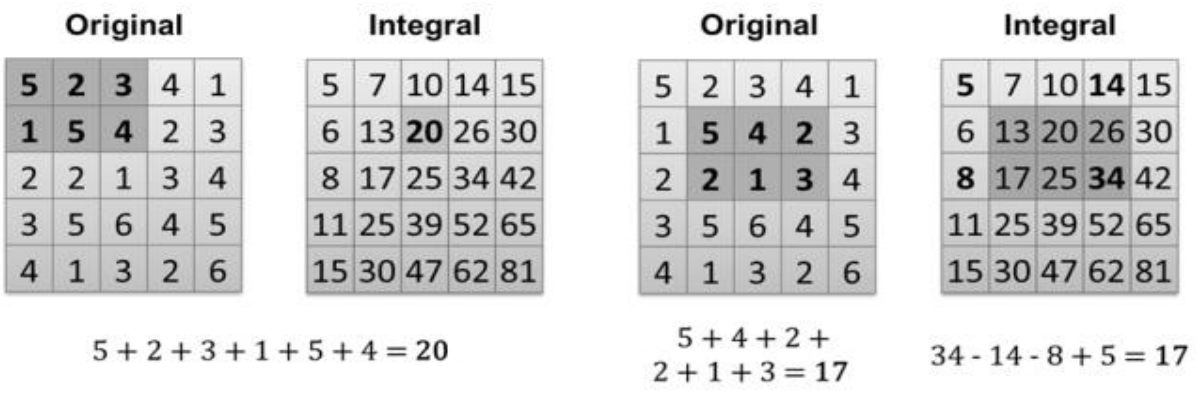


Figure 3: Integral image representation as summed-area table

The main advantage of using the integral image is that it speeds up the feature extraction, because any rectangle in an image can be calculated from that integral image, in only four memory calls and three mathematical operations. Given a rectangle specified as four coordinate points *A(x1, y1), B(x2, y2)* as upper left and right and *C(x3, y3)* and *D(x4, y4)* as lower left and right, evaluating the area of the rectangle is done in four image references to integral image as following:

Sum of the grey rectangle = D - (B + C) + A

This represents a huge performance boost in term of feature extraction. So we can summarize above result in following equation. [3]

$$F(x,y) = \sum_{a=0}^{x}\sum_{b=0}^{y} I(a,b)$$

$$\sum_{a=x0}^{x1}\sum_{b=y0}^{x2} I(a,b) = F(x1,y1) - F(x0,y1) - F(x1,y0) + F(x0,y0)$$

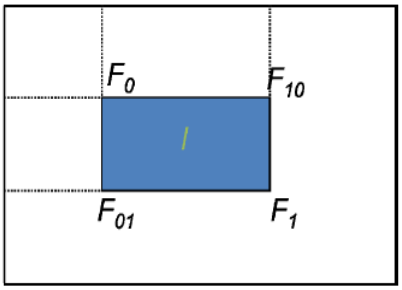


Figure 4: Fast calculation in integral image

1.2.3 Attentional Cascade

This scheme wouldn't have worked in real time if the detector wasn't extremely fast. The catch here is that the detector is extremely fast and quickly discards unpromising windows if the region doesn't contain a face.

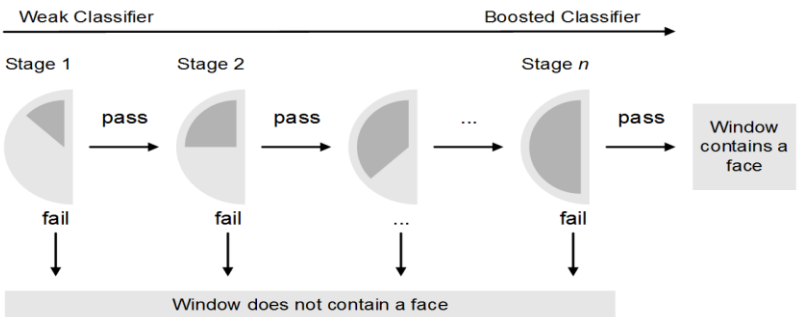


Figure 5: Attentional Cascade Viola-Jones

When it's not sure about a given region, it spends more time trying to decide whether it's a face or not. When it gives up trying to reject, it can only conclude it is face.

A cascade is a way of combining classifiers in such a way that given classifier is only processed after all the classifier coming before have already been processed. The classification scheme used in Viola-Jones is a cascade of boosted classifiers. Each stage in a cascade is strong classifier, so that it can obtain really high rejection rate by combining a series of weak classifiers. [5]

1.2.3.1 Structure of attentional cascade

The final detector is a 30-layer cascade of classifiers which included a total of 4297 features. The first classifier in the cascade is constructed using two features and rejects about 60% of non-faces while correctly detecting close to 100% of faces. The next classifier has five features and rejects 80% of non-faces while detecting almost 100% of faces. The next three layers are 20-feature classifiers followed by two 50-feature classifiers followed by five 100-feature classifiers and then twenty 200-feature classifiers.

The two, five and first twenty-feature classifiers were trained with the 4916 faces and 10,000 non-face sub-windows (also of size 24 by 24 pixels) using the Adaboost training procedure. The non-face sub-windows were collected by selecting random sub-windows from a set of 9500 images which did not contain faces. Different sets of non-face sub-windows were used for training the different classifiers to ensure that they were somewhat independent and didn't use the same features.

1.2.3.1 AdaBoost algorithm

Many general feature selection procedures have been proposed. The final application demanded a very aggressive approach which would discard the vast majority of features. For a similar recognition problem Papageorgiou et al. proposed a scheme for feature selection based on feature variance [4]. They demonstrated good results selecting 37 features out of a total 1734 features.

In practice no single feature can perform the classification task with low error. Features which are selected early in the process yield error rates between 0.1 and 0.3. Features selected in later rounds, as the task becomes more difficult, yield error rates between 0.4 and 0.5. Following table shows number of steps performed for learning process. [5]

1. Given example images (x_1, y_1)..., (x_n, y_n) where $y_1 = 0,1$ for negative and positive examples respectively.
2. Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ $for\ y_i = 0$ respectively, where *m* and *l* are the number of negatives and positives respectively
3. For t = 1, . . ., T:
    1. Normalize the weights,

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

    So that w_t is probability distribution.

2. For each feature j, train classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t , $e_j = \sum_i w_i |h_j(x_i) - y_i|$
3. Choose the classifier, h_t with the lowest e_t.
4. Update the weights:

$$w_{t+1,i} = w_{t,i} ß_t^{1-e_i}$$

Where $e_i = 0$, if example e_t is classified correctly, $e_i = 1$ otherwise, and $ß_t = \frac{e_t}{1-e_t}$

4. The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \propto_t h_t \geq \frac{1}{2} \sum_{t=1}^{T} \propto_t \\ 0 & otherwise \end{cases}$$

Where $\propto_t = log \frac{1}{ß_t}$

Table 1: The AdaBoost algorithm for classifier learning. Each round of boosting selects one feature from the 180,000 potential features [5]

1.3 Source Code

As, we have learned the theoretical aspects of face detection algorithm, lets now jump to the source code and build an application that detects human frontal faces in an image. I have decided to use Accord framework libraries for this purpose. Accord.NET provides statistical analysis, machine learning, image processing and computer vision methods for .NET applications. The Accord.NET Framework extends the popular AForge.NET with new features, adding to a more complete environment for scientific computing in .NET [6].

The class structure is shown in the figure 6 on the next page. First of all, the exhaustive search happens in **HaarObjectDetector** class, which is the main class for object detection. Its constructor accepts **HaarClassifier** class as a parameter. The main role of **HaarObjectDetector** is to scan the image with a sliding window, relocating and rescaling as per need and then to call **HaarClassifier** to check whether there is face or not.

The classifier on the other hand is completely specified by a **HaarCascade** object and its current operating scale. **HaarCascade** possesses a series of stages, which are evaluated sequentially. As soon a stage rejects the window, the classifier stops and returns false. Now comes the **Classify** method of **HaarCascadeStage** object, which contains series of decision trees. All it does is, processes several decision trees and checks if the feature value is higher than certain decision threshold. Each decision node in a trees contains a single feature, and single feature may contain two, three or four rectangles.

The framework already comes with default HaarCascade definitions available as instantiable classes, which means there is no need of *.xml files. Mainly, those definitions are constructed from OpenCV's *.xml definition files using a class generator named **HaarCascadeWriter**. Therefore, to have written something, the definitions had first to be loaded into the framework. That's why, the framework can open OpenCV's definition files using standard .NET serialization.

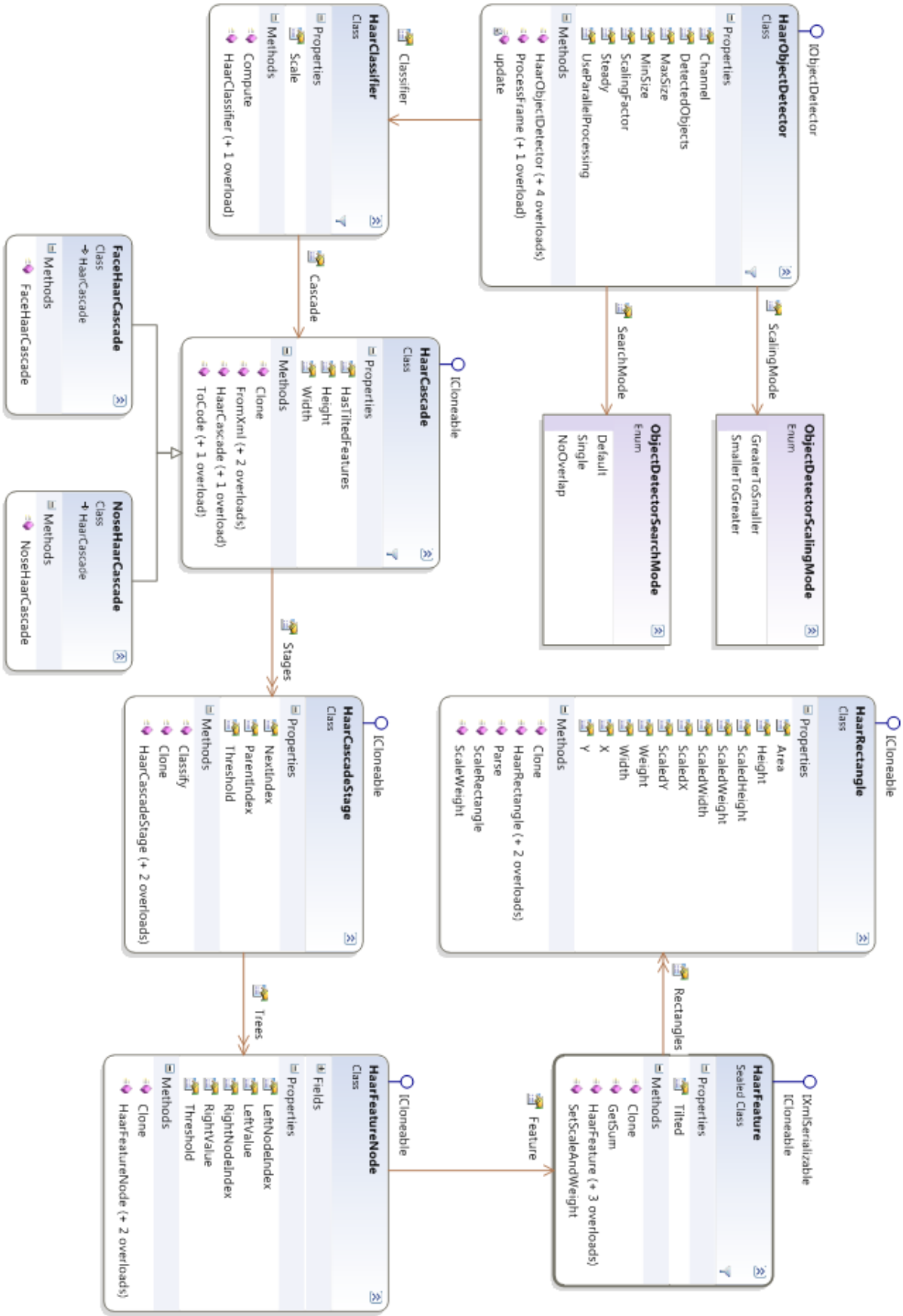


Figure 6: Class structure of Accord framework libraries v2.15.0.0 [7]

Let's start from **HaarObjectDetector** which is mainly responsible for generating and sliding window across the image. Window size keeps changing during the search and mainly depends upon the scaling factor. If it is GreaterToSamller, it shrinks and expands for SmallerToGreater selection. You can also set the search mode values for Default, Single and NoOverlap, depending upon the number of objects in image to perform detection faster. Following is the code for **HaarObjectDetector** [8]

```
#region Assembly Accord.Vision.dll, v2.15.0.0
#endregion

using AForge.Imaging;
using System;
using System.Drawing;

namespace Accord.Vision.Detection
{
    public class HaarObjectDetector : IObjectDetector
    {
        public HaarObjectDetector(HaarCascade cascade);
        public HaarObjectDetector(HaarCascade cascade , int minSize);
        public HaarObjectDetector(HaarCascade cascade , int minSize ,
ObjectDetectorSearchMode searchMode);
        public HaarObjectDetector(HaarCascade cascade , int minSize ,
ObjectDetectorSearchMode searchMode , float scaleFactor);
        public HaarObjectDetector(HaarCascade cascade , int minSize ,
ObjectDetectorSearchMode searchMode , float scaleFactor , ObjectDetectorScalingMode
scalingMode);

        public int Channel { get; set; }
        public HaarClassifier Classifier { get; }
        public Rectangle[] DetectedObjects { get; }
        public Size MaxSize { get; set; }
        public Size MinSize { get; set; }
        public float ScalingFactor { get; set; }
        public ObjectDetectorScalingMode ScalingMode { get; set; }
        public ObjectDetectorSearchMode SearchMode { get; set; }
        public int Steady { get; }
        public int Suppression { get; set; }
        public bool UseParallelProcessing { get; set; }

        public Rectangle[] ProcessFrame(Bitmap frame);
        public Rectangle[] ProcessFrame(UnmanagedImage image);
    }
}
```

As you can see in the above code, that **HaarObjectDetector** which takes **HaarClassifier** class as a parameter. Classifier contains **Compute** method and **scale** property. This method actually detects the presence of face in window at some particular stage. Following is the code for **Compute** method. [8]

```
/// <summary>
///   Detects the presence of an object in a given window.
/// </summary>

public bool Compute(IntegralImage2 image, Rectangle rectangle)
{
    int x = rectangle.X;
    int y = rectangle.Y;
    int w = rectangle.Width;
```

```
    int h = rectangle.Height;
    double mean = image.GetSum(x, y, w, h) * invArea;
    double factor = image.GetSum2(x, y, w, h) * invArea - (mean * mean);
    factor = (factor >= 0) ? Math.Sqrt(factor) : 1;

    // For each classification stage in the cascade
    foreach (HaarCascadeStage stage in cascade.Stages)
    {
        // Check if the stage has rejected the image
        if (stage.Classify(image, x, y, factor) == false)
        {
            return false; // The image has been rejected.
        }
    }

    // If the object has gone all stages and has not been rejected
    //the object has been detected.
    return true;
}
```

HaarCascade inherits from **ICloneable** interface, which provides it cloning support and enables the customize implementation that creates the copy of an existing object. It contains number of stages and each stage has a set of features. The classifier is completely specified by a **HaarCascade**. It possesses a series of stages, which are evaluated sequentially. As soon a stage rejects the window, the classifier stops and returns false. Now comes the **Classify** method of **HaarCascadeStage** object, which contains series of decision trees. All it does is, processes several decision trees and checks if the feature value is higher than certain decision threshold. Each decision node in a trees contains a single feature, and single feature may contain two, three or four rectangles. Following is the code for **HaarCascadeStage** method **Classify**.[9]

```
/// <summary>
///   Classifies an image as having the searched object or not.
/// </summary>

public bool Classify(IntegralImage2 image, int x, int y, double factor)
{
    double value = 0;

    // For each feature in the feature tree of the current stage,
    foreach (HaarFeatureNode[] tree in Trees)
    {
        int current = 0;
        do
        {
            // Get the feature node from the tree
            HaarFeatureNode node = tree[current];

            // Evaluate the node's feature
            double sum = node.Feature.GetSum(image, x, y);

            // And increase the value accumulator
            if (sum < node.Threshold * factor)
            {
                value += node.LeftValue;
                current = node.LeftNodeIndex;
            }
            else
            {
                value += node.RightValue;
```

```
                current = node.RightNodeIndex;
            }
        } while (current > 0);
    }

    // After we have evaluated the output for the
    //  current stage, we will check if the value
    //  is still lesser than the stage threshold.
    if (value < this.Threshold)
    {
        // If it is, the stage has rejected the current
        // image and it doesn't contains our object.
        return false;
    }
    else
    {
        // The stage has accepted the current image
        return true;
    }
}
```

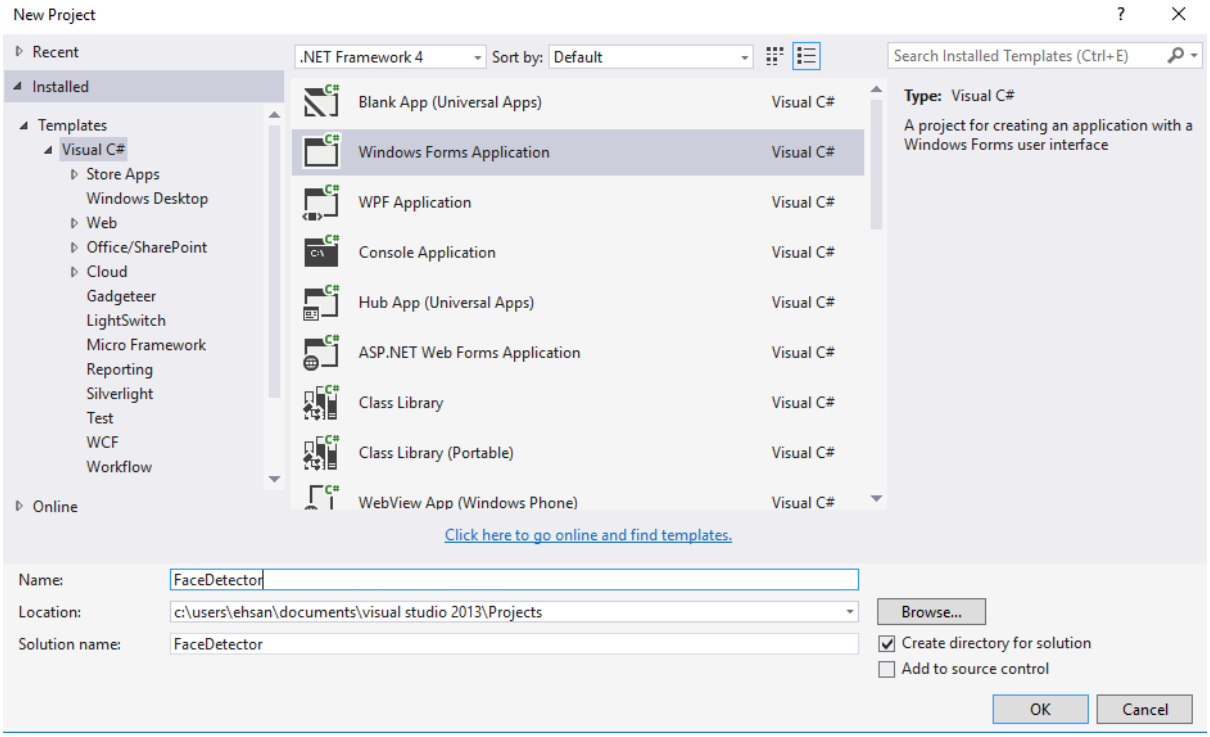
As for now, we have learned the basics of framework libraries, let's start using this code to build the app.

1.4 Building Face Detection App

Using this code is rather simple. Use following steps to as described below to build the app. [11]

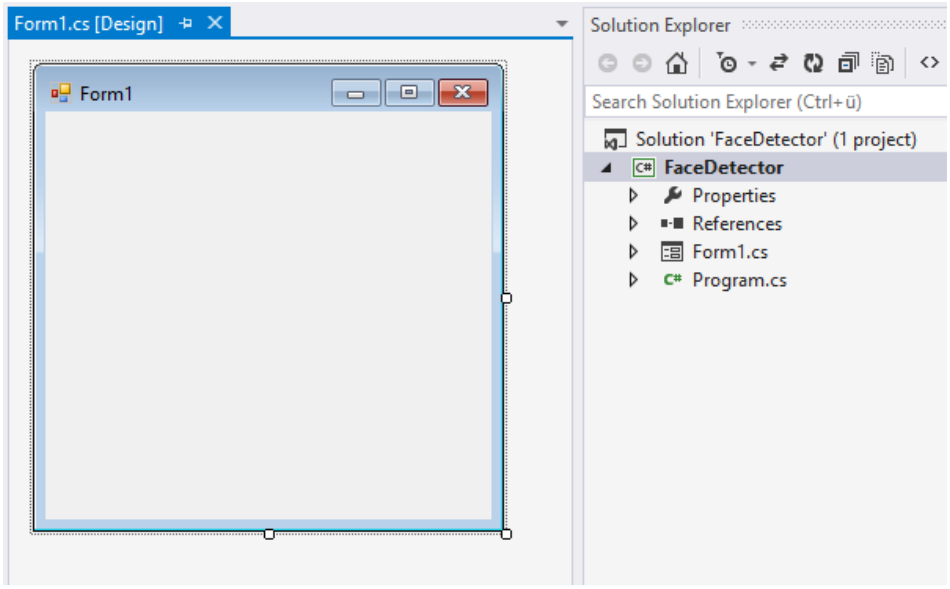
Creating Windows Form Application:

1. On the menu bar, choose **File**, **New**, **Project**. The dialog box should look like this.



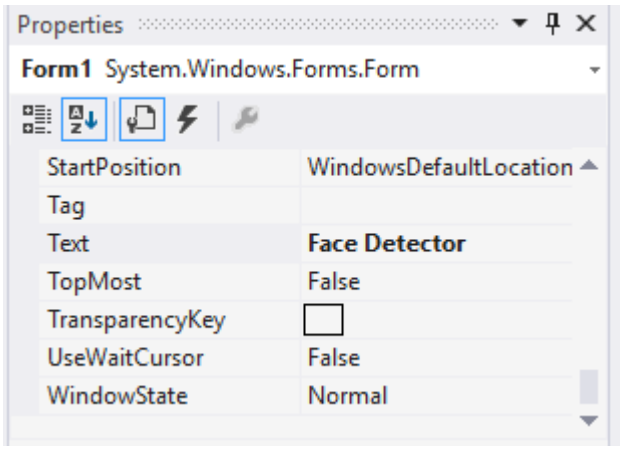
2. Choose **Visual C#** in installed templates

3. In the templates list, choose the **Windows Forms Application** icon. Name the new form **FaceDetector**, and then choose the **OK** button.
4. The following illustration shows what you should now see in the Visual Studio interface.



To set your form Properties:

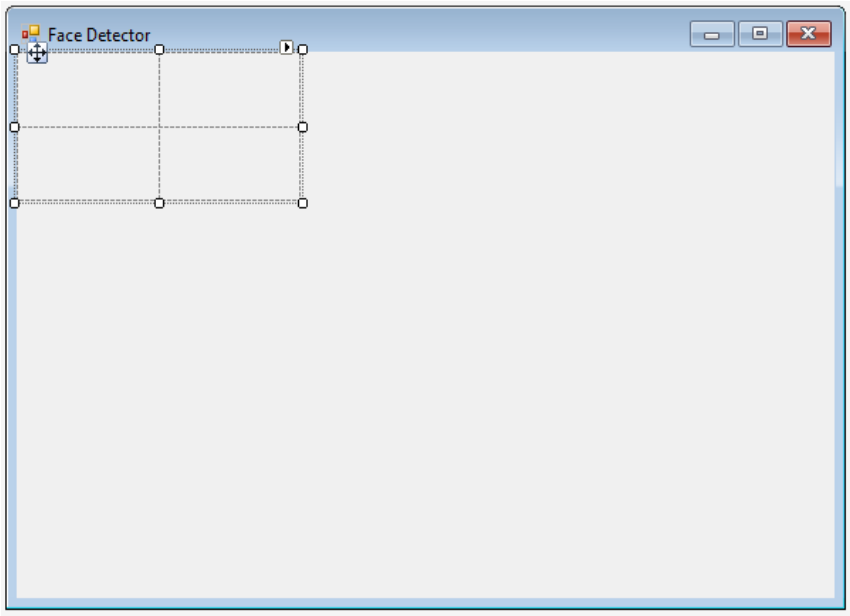
1. Choose anywhere inside the form **Form1** to select it. Look at the **Properties** window, which should now be showing the properties for the form. Forms have various properties. For example, you can set the foreground and background color, title text that appears at the top of the form, size of the form, and other properties.
2. After the form is selected, find the **Text** property in the **Properties** window. Depending on how the list is sorted, you might need to scroll down. Choose **Text**, type **Face Detector**, and then choose ENTER. Your form should now have the text **Face Detector** in its title bar, and the **Properties** window should look similar to the following picture.



3. Go back to Windows Forms Designer. Choose the form's lower-right drag handle. Drag the handle to resize the form so the form is wider and a bit taller.

To layout your form with TableLayoutPanel control:

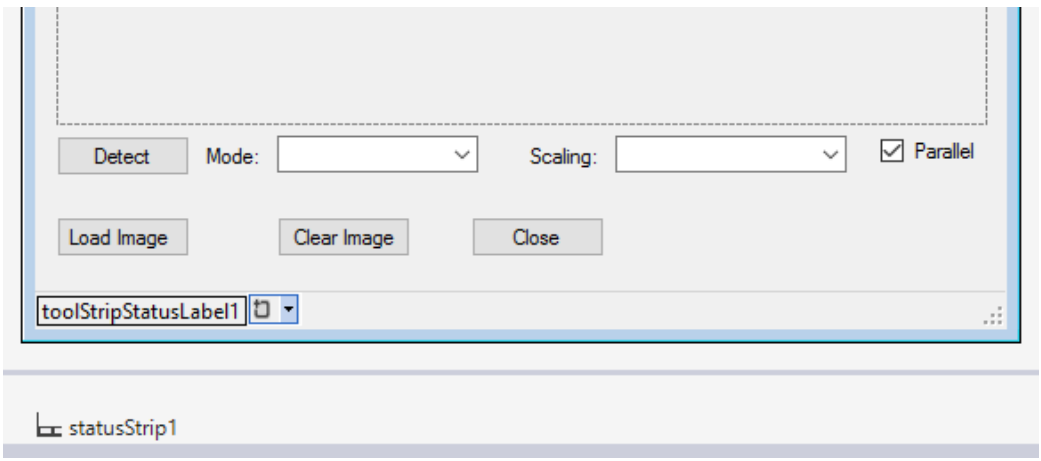
1. On the left side of the Visual Studio IDE, locate the **Toolbox** Tab. Choose the **Toolbox** tab, and the Toolbox appears. (Or, on the menu bar, choose **View**, **Toolbox**.)
2. Choose the small triangle symbol next to the **Containers** group to open it
3. You can add controls like buttons, check boxes, and labels to your form. Double-click the **TableLayoutPanel** control in the Toolbox. (Or, you can drag the control from the toolbox onto the form.) When you do this, the IDE adds a **TableLayoutPanel** control to your form, as shown in the following picture.



4. The control selector is a drop-down list at the top of the **Properties** window. In this example, it shows that a control called **tableLayoutPanel1**is selected. You can select controls either by choosing an area in Windows Forms Designer or by choosing from the control selector. Now that **TableLayoutPanel** is selected, find the **Dock** property and choose **Dock**, which should be set to **None**. Notice that a drop-down arrow appears next to the value. Choose the arrow, and then select the **Fill** button
5. Currently, the TableLayoutPanel has two equal-size rows and two equal-size columns. You need to resize them so the top row and right column are both much bigger. In Windows Forms Designer, select the TableLayoutPanel. In the upper-right corner, there is a small black triangle button.
6. Choose the **Edit Rows and Columns** task to display the **Column and Row Styles** window. And delete one column. Make it two rows and one column. Set row 1 about 80 percent and row 2 at 20 percent of total area.

To add controls to your form:

1. Go to the Toolbox tab (located on the left side of the Visual Studio IDE) and expand the **Common Controls** group. This shows the most common controls that you see on forms.
2. Choose the TableLayoutPanel control on the form. To verify that the TableLayoutPanel is selected, make sure that its name appears in the dropdown list box at the top of the **Properties** window. You can also choose form controls by using the dropdown list box at the top of the **Properties** window. Choosing a control this way can often be easier than choosing a tiny control with a mouse.
3. Double-click the **PictureBox** item to add a PictureBox control to your form. Because the TableLayoutPanel is docked to fill your form, the IDE adds the PictureBox control to the first empty cell (the upper left corner).
4. Double-click the **PictureBox** item to add a PictureBox control to your form. Because the TableLayoutPanel is docked to fill your form, the IDE adds the PictureBox control to the first empty cell (the upper left corner).
5. Choose the **Dock in parent container** link. This automatically sets the PictureBox **Dock** property to **Fill**. To see this, choose the PictureBox control to select it, go to the **Properties** window, and be sure that the **Dock** property is set to **Fill**.
6. Choose the TableLayoutPanel on the form and then add a **CheckBox** control to the form. Double-click the **CheckBox** item in the Toolbox to add a new CheckBox control to the next free cell in your table. Change Text property to **Parallel** and checked property **True**.
7. Similarly, add two **ComboBox** along with two labels and name the "**Mode**:" and "**Scaling**:" respectively
8. Go to the **Commons Control** again add four buttons and name them "**Detect**", "**Load Image**", "**Clear Image**" and "**Close**" respectively by changing their **Text** property.
9. Choose the **Close** button to select it. Hold down the CTRL key and choose the other three buttons, so that they are all selected. While all the buttons are selected, go to the **Properties** window and scroll up to the **AutoSize** property. This property tells the button to automatically resize itself to fit all of its text. Set it to **true**. Your buttons should now be sized properly and be in the right order. (As long as all four buttons are selected, you can change all four **AutoSize** properties at the same time.)
10. Go to **Tools** again and in **All Window Forms**, choose **Status Strip** and add it to the form by double clicking. On status strip add **Tool Strip Label** as following.

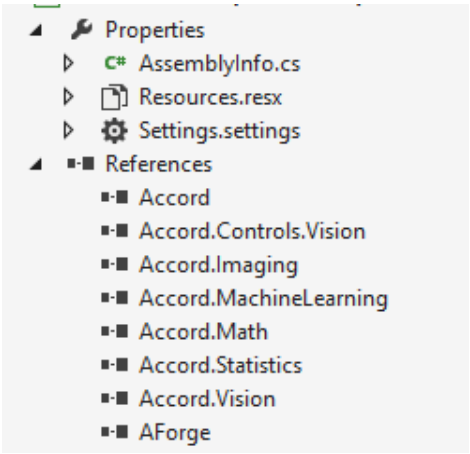


Writing code for button click Event Handlers:

1. Double click on Detect, Load Image, Clear Image and Close buttons and it will automatically generate event handlers for them in Main form as following:

```
private void detect_Click(object sender , EventArgs e)
{

}

private void loadImage_Click(object sender , EventArgs e)
{

}

private void clearImage_Click(object sender , EventArgs e)
{

}

private void close_Click(object sender , EventArgs e)
{

}
```

2. First, go to the Solution Explorer and add Accord framework libraries references as shown below.



3. Use following references in your code

```
using System;
using System.Diagnostics;
using System.Drawing;
using System.Windows.Forms;
using Accord.Imaging.Filters;
using Accord.Vision.Detection;
using Accord.Vision.Detection.Cascades;
using FaceDetection.Properties;
using System.Net;
```

4. In the **MainForm** set the properties of **ComboBoxes** and **ToolStripStatus** with the following code. And create a classifier with the name **cascade** and feed this cascade in the **HaarObjectDetector** to create a detector and set total number of stages equal to 30.

```
public MainForm()
{
    InitializeComponent();

    cbMode.DataSource = Enum.GetValues(typeof(ObjectDetectorSearchMode));
    cbScaling.DataSource = Enum.GetValues(typeof(ObjectDetectorScalingMode));

    cbMode.SelectedItem = ObjectDetectorSearchMode.NoOverlap;
    cbScaling.SelectedItem = ObjectDetectorScalingMode.SmallerToGreater;

    toolStripStatusLabel1.Text = "Please select the detector options and click Detect to begin.";

    HaarCascade cascade = new FaceHaarCascade();
    detector = new HaarObjectDetector(cascade, 30);
}
```

5. Add following code to **Detect** button event handler.

```
private void button1_Click(object sender, EventArgs e)
{
    picture = new Bitmap(pictureBox1.Image);
    detector.SearchMode = (ObjectDetectorSearchMode)cbMode.SelectedValue;
    detector.ScalingMode = (ObjectDetectorScalingMode)cbScaling.SelectedValue;
    detector.ScalingFactor = 1.5f;
    detector.UseParallelProcessing = cbParallel.Checked;
    detector.Suppression = 2;

    Stopwatch sw = Stopwatch.StartNew();

    // Process frame to detect objects
    Rectangle[] objects = detector.ProcessFrame(picture);

    sw.Stop();

    if (objects.Length > 0)
    {
        RectanglesMarker marker = new RectanglesMarker(objects, Color.Fuchsia);
        pictureBox1.Image = marker.Apply(picture);
    }

    toolStripStatusLabel1.Text = string.Format("Completed detection of {0} objects in {1}.",
        objects.Length, sw.Elapsed);
}
```

6. Add following code to Load Image button event handler:

```
private void loadImage_Click(object sender , EventArgs e)
{
    var request = WebRequest.Create("http://169.254.120.154/seepicture");
    using (var response = request.GetResponse())
    using (var stream = response.GetResponseStream())
    {
        Bitmap picture = new Bitmap(stream);
        pictureBox1.Image = picture;
    }
}
```

7. Add following code to Clear Image button event handler:

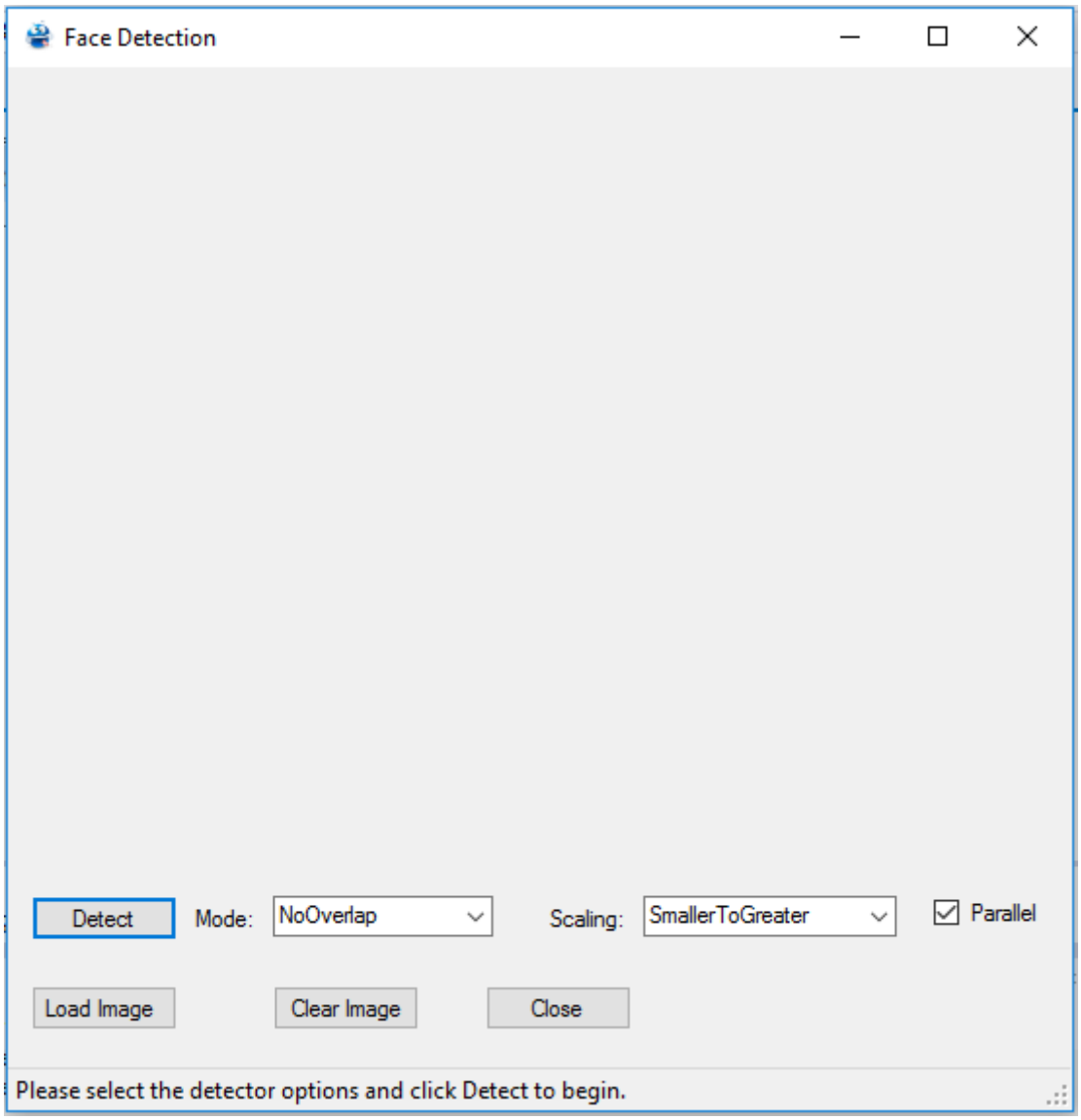
```
private void clearImage_Click(object sender , EventArgs e)
{
    pictureBox1.Image = null;
}
```

8. Add following code to Close Image button event handler:

```
private void close_Click(object sender , EventArgs e)
{
    this.Close();
}
```

Final App:

After done editing code for event handler. Go and press start button to run the application. Result would like as following.



Part 2: .NET Gadgeteer

2.1 Introduction

.NET Gadgeteer is built on top of NETMF to provide a rapid development platform that utilizes mainboards and plug-and-play modules. It is a standard maintained by Microsoft for standardizing the connections between mainboards and modules. [12]

2.2 .NET Micro Framework

.NET Gadgeteer software works on top of the .NET Micro Framework (NETMF) software. NETMF includes thousands of methods, some are borrowed from the full .NET framework. Meaning .NET Desktop developers already know how to program NETMF and .NET Gadgeteer products. NETMF also extends the standard framework with numerous hardware related methods. This includes the ability to read and write to control pins allowing you to blink some lights or to check if a button is pressed. GHI Electronics also extends the framework with some exclusive features, such as using databases and USB Host. [12]

2.3 Setting up development environment

As .NET Gadgeteer uses the latest professional tools on the market, a proper software installation is necessary. To get started, install the exact software listed on GHI Electronics' support page. [12, 13]

The installation steps will include:

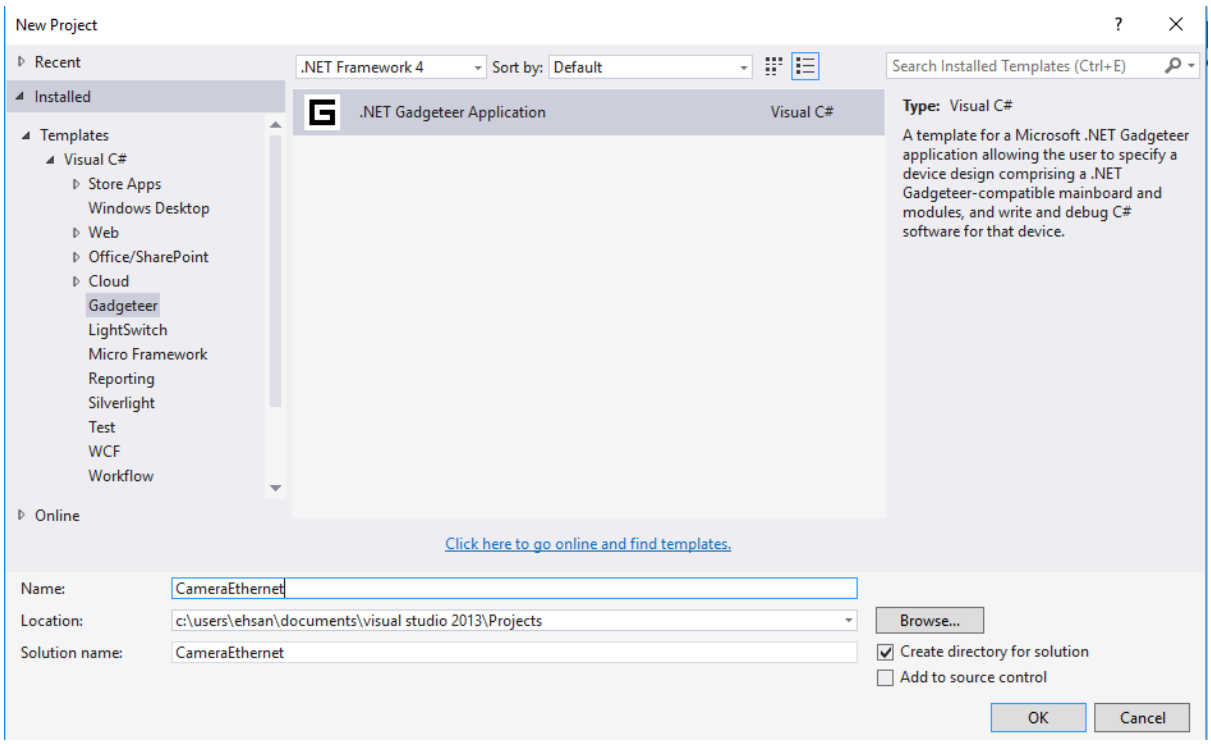
1. Microsoft Visual Studio (IDE and the compiler)
2. Microsoft NETMF Project System (plugin for Visual Studio to allow deploying/debugging)
3. Microsoft NETMF SDK (framework libraries)
4. Microsoft's .NET Gadgeteer (graphical designer and project templates)
5. GHI Electronics' SDK (drivers for mainboards and modules plus GHI Electronics' exclusive libraries)

2.4 Building “Ethernet Camera” App

After setting up the development environment, Let’s start writing application for gadgeteer. Follow the instructions as described below.

2.4.1 Setting Up

1. Run Visual Studio and start a new project. Click on “FILE > New Project...”. Select C# and then Gadgeteer as shown below.

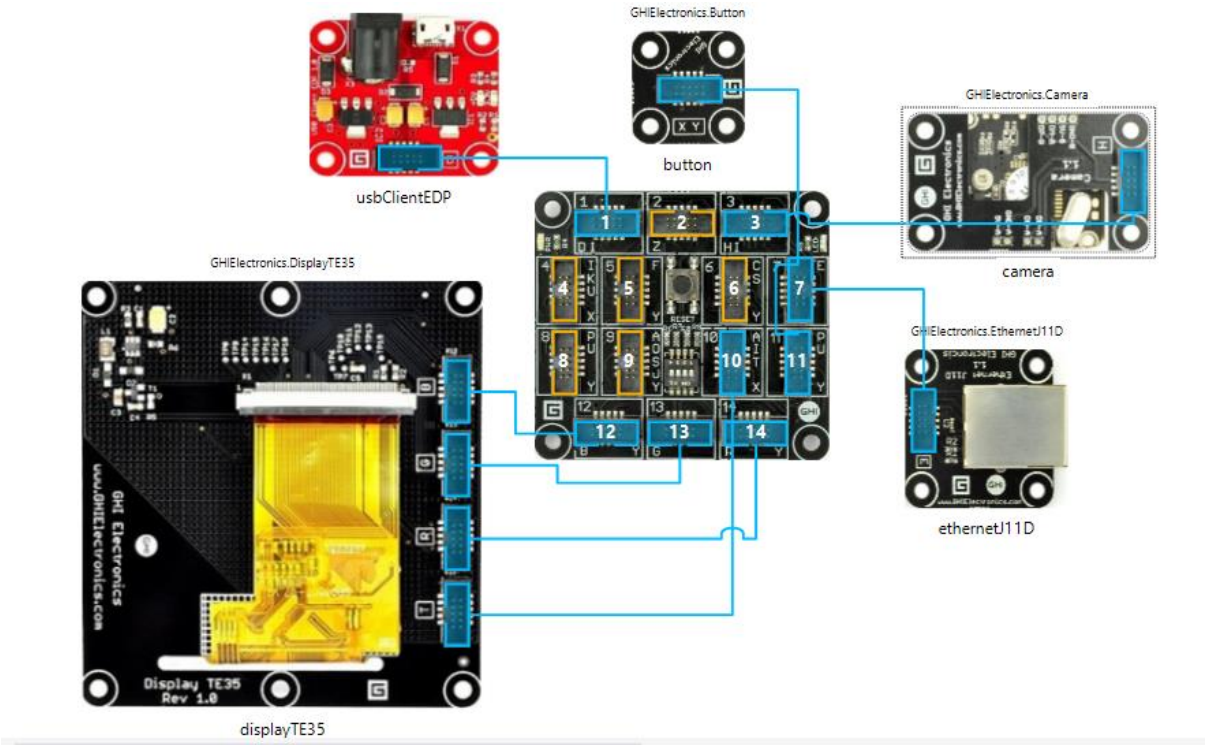


2. The next step is to select the proper mainboard. Make sure to also select the latest version of .NET Micro Framework if this option is available. Select FEZ Spider from available kits.

3. Click the Create button and the .NET Gadgeteer Visual Designer's window will now appear with the mainboard selected. Using .NET Gadgeteer's designer is simple and speeds up your design. On the designer screen, open the Toolbox, drag and drop the wanted components, right-click and select "Connect all modules" and you've finished the initial setup. This automatically includes all the necessary DLLs and generates a variable for reach component. Allowing you to focus on writing the core functionality of your design without having to write it all from the start. Saving you precious time and resources.

4. Click the Toolbox tab to show all the available modules and mainboards. From Toolbox you can drag and drop the modules to the Visual Designer's panel. Add gadgeteer camera, TE35 touch screen display, USB client EDP, Ethernet J11D and a button module.

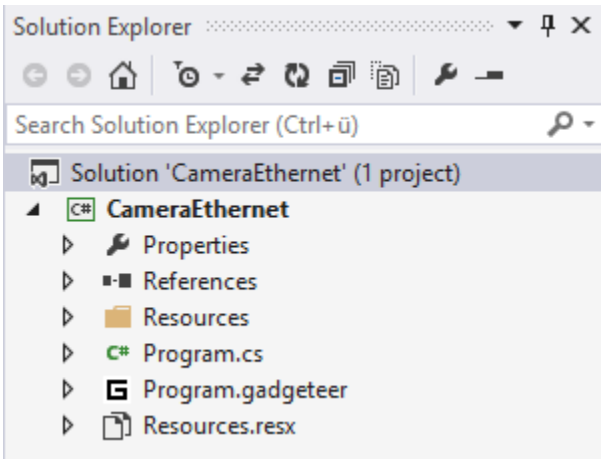
5. Right click on Visual Designer's and select “connect all modules”. It will automatically connect them with the FEZ Spider. Final design would look as following.



6. Connect all modules of the hardware accordingly with the FEZ Spider kit, and attach it with the computer using USB cable and observe a red light on power module. Attach ethernetJ11D connector to computer using an Ethernet cable.

2.4.2 Coding the software

1. Locate the Solution Explorer window. You will find two C# files in there. One is auto-generated by the designer surface, where you added the modules. Do not modify this file. Now click Program.cs to view it. In the code, you will see a “ProgramStarted” method and inside of it there is a debug statement. Everything in green is considered a comment.



2. In Program.cs use following references.

```
using System;
using System.Net;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Controls;
using Microsoft.SPOT.Presentation.Media;
using Microsoft.SPOT.Touch;

using Gadgeteer.Networking;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using System.Net.Sockets;
using Gadgeteer.Modules.GHIElectronics;
using Microsoft.SPOT.Net.NetworkInformation;
```

3. NET Micro Framework includes Socket support, allowing users to communicate using TCP/UDP. There is also HTTP support for easy web page handling for clients and servers. So, lets first set up a static IP address for the device. Add following to the ProgramStarted method.

```
ethernetJ11D.NetworkInterface.Open();
ethernetJ11D.UseThisNetworkInterface();

NetworkInterface[] networkInterfaces = NetworkInterface.GetAllNetworkInterfaces();
// loop through each interface, assign an IP accordingly
foreach (NetworkInterface ni in networkInterfaces)
{
    if (ni.NetworkInterfaceType == NetworkInterfaceType.Ethernet)
    {
        ni.EnableStaticIP("169.254.120.154" , "255.255.0.0" , "192.168.1.1");
        ni.EnableStaticDns(new string[] { "8.8.8.8" , "8.8.4.4" });
    }
}
```

4. As of now, our device has been assigned a static IP. Let's define some events in the ProgramStarted method. These events will automatically be called when network goes up/down or when button is pressed or picture is captured.

```
ethernetJ11D.NetworkUp += new GTM.Module.NetworkModule.NetworkEventHandler(ethernet_NetworkUp);
ethernetJ11D.NetworkDown += new GTM.Module.NetworkModule.NetworkEventHandler(ethernet_NetworkDown);

button.ButtonPressed += new Button.ButtonEventHandler(button_ButtonPressed);
camera.PictureCaptured += new Camera.PictureCapturedEventHandler(camera_PictureCaptured);
```

5. Also define two web events on the top of the program as following. Web event "test" will be used to test if the network is set up properly and web event "seePicture" will be used to actually see the picture captured by gadgeteer camera after pressing the button.

```
Gadgeteer.Networking.WebEvent test;
Gadgeteer.Networking.WebEvent seePicture;
```

6. Definition of NetworkUp web event is as following. It starts the local server at given IP address and well known http port 80. It also handles the incoming web requests for "test" and "seepicture" from the browser.

```
void ethernet_NetworkUp(GTM.Module.NetworkModule sender, GTM.Module.NetworkModule.NetworkState state)
{
    Debug.Print("Network is Up with following properties");
    Debug.Print("IP:" + ethernetJ11D.NetworkSettings.IPAddress.ToString());

    Gadgeteer.Networking.WebServer.StartLocalServer(ethernetJ11D.NetworkSettings.IPAddress , 80);

    foreach (string s in sender.NetworkSettings.DnsAddresses)
    Debug.Print("Dns:" + s);

    var NetworkSettings = ethernetJ11D.NetworkSettings;
    Debug.Print("IP Address: " + NetworkSettings.IPAddress);
    Debug.Print("Subnet Mask: " + NetworkSettings.SubnetMask);
    Debug.Print("Gateway: " + NetworkSettings.GatewayAddress);

    test = Gadgeteer.Networking.WebServer.SetupWebEvent("test");
    test.WebEventReceived += new WebEvent.ReceivedWebEventHandler(test_WebEventReceived);

    seePicture = Gadgeteer.Networking.WebServer.SetupWebEvent("seepicture");
    seePicture.WebEventReceived += new WebEvent.ReceivedWebEventHandler(seePicture_WebEventReceived);

    button.TurnLedOn();

}
```

7. Definition of NetworkDown web event is as following. It simply prints "Network Down" in the Debug window and turns the button led off, when network goes down.

```
void ethernet_NetworkDown(GTM.Module.NetworkModule sender , GTM.Module.NetworkModule.NetworkState state)
{
    button.TurnLedOff();
    Debug.Print("Network Down");
}
```

8. "test" and "seepicture" web requests are handled as following by the server.

```
void seePicture_WebEventReceived(string path, WebServer.HttpMethod method, Responder responder)
{
   if (pic != null)
        responder.Respond(pic);

   else
        responder.Respond("Take picture first");
}

void test_WebEventReceived(string path, WebServer.HttpMethod method, Responder responder)
 {
        Debug.Print("Hello world");
        string content = "<html><body><h1>Hosted on .NET Gadgteer!!</h1><a
        href=\"http://169.254.120.154/seepicture\">http://169.254.120.154/seepicture</a></body
        ></html>";
        byte[] bytes = new System.Text.UTF8Encoding().GetBytes(content);
        responder.Respond(bytes , "text/html");

 }
```

9. ButtonPressed and PictureCaptured events are handled as following. When button is pressed, picture is captured and when picture is captured then "PictureCaptured" event is called which displays the picture on lcd display. The code is as following.

```
void button_ButtonPressed(Button sender , Button.ButtonState state)
{
    camera.TakePicture();
}
```

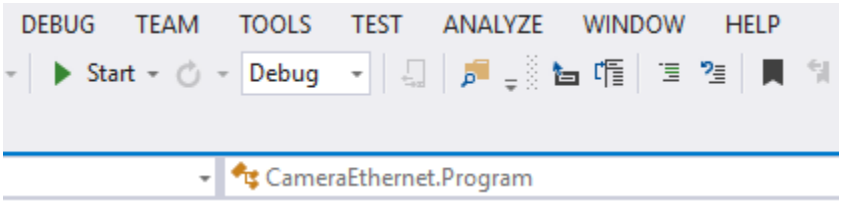
and

```
void camera_PictureCaptured(Camera sender, GT.Picture picture)
{
    pic = picture;
    displayTE35.SimpleGraphics.DisplayImage(picture, 5, 5);
}
```

2.4.3 Running the application

After done writing all the code, you are ready now to run your application. Make sure all the connections in hardware are done properly and gadgeteer is attached to the computer through USB cable and EthernetJ11D connector is attached with the computer through Ethernet cable.

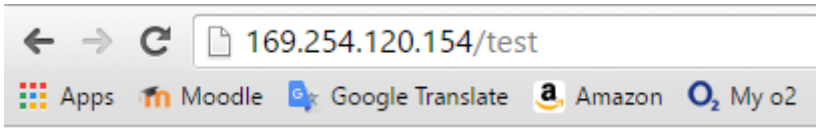
1. To make sure that there are no errors in the code, first build the solution by clicking on the menu bar item “Build” and select “Build Solution” or simply press F6 key.
2. After successful build, click deploy solution to deploy it on the gadgeteer.
3. Once it is successfully deployed, now you can run the application by clicking on the “Start” button.



4. You will see following output in the debug window. It will show what IP address, gateway and DNS is assigned to gadgeteer, and on what address local server is started.

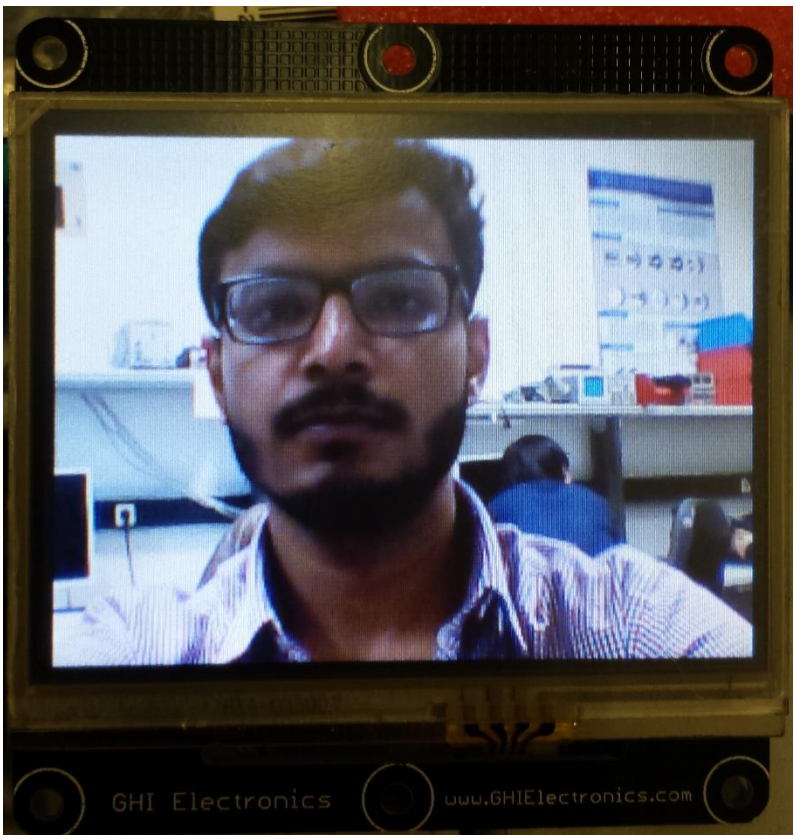
```
Using mainboard GHI Electronics FEZ Spider version 1.0
Program Started
Network Down
Network is Up with following properties
IP:169.254.120.154
Web server started at http://169.254.120.154:80/
Dns:8.8.8.8
Dns:8.8.4.4
IP Address: 169.254.120.154
Subnet Mask: 255.255.0.0
Gateway: 192.168.1.1
```

5. Let’s test if everything is working properly. Go to the web browser and type following in the browser.

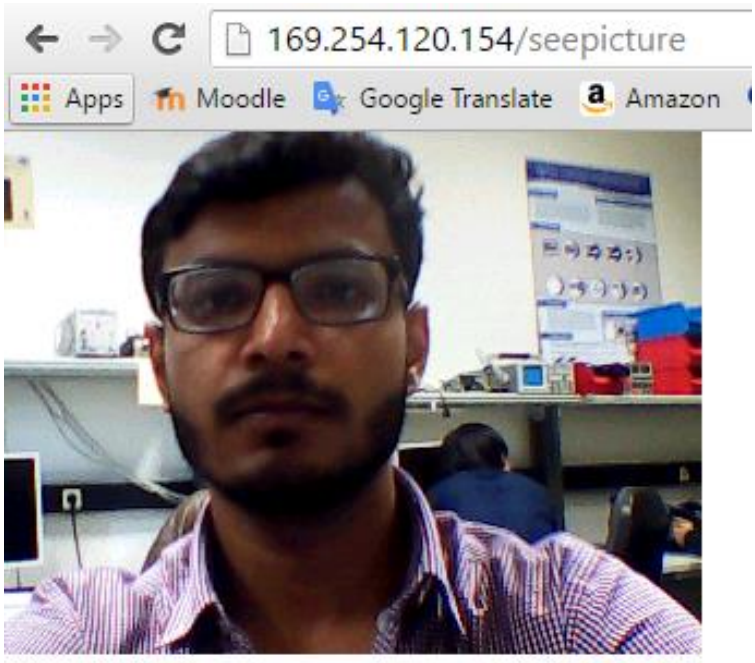


6. On successful set up you will see the following in the browser.
7. 

8. Now you are ready to take picture. Press the button module attached with the FEZ Spider kit and take a picture. As soon as the picture is captured. It will be displayed on the attached LCD display.



9. Go to web browser again and click on the link displayed in the browser to see the picture in the browser. You will see following in the browser.



Part 3: How to run the complete project?

As we are done writing code and testing our applications. Now we can start capturing and detecting human faces in the images captured by gadgeteer camera. Follow the instructions given below to run the final project.

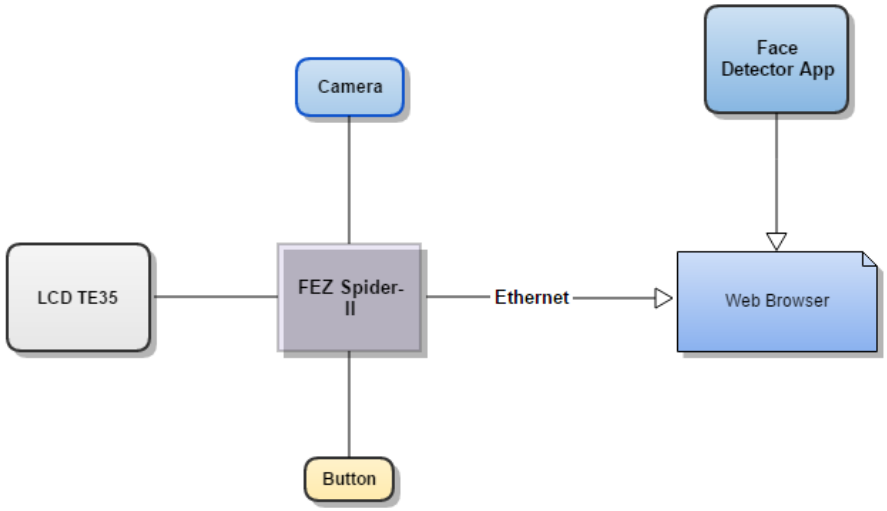


Figure 7: Project Block Diagram

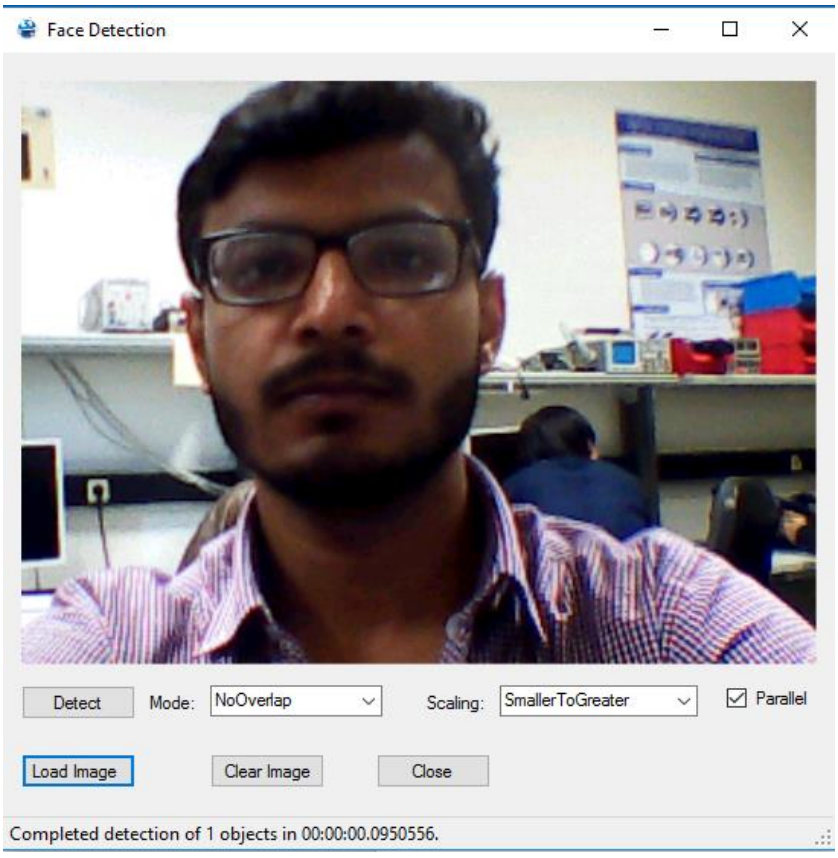
3.1 Steps

1. First check all the connections in the hardware are done properly and it is attached to the computer using USB and Ethernet cable.
2. Open "CameraEthernet" project in visual studio and click on the program.cs file.
3. Build the solution by pressing F6 key, after successful build, deploy solution to the device.
4. Press the Start button on Visual Studio to run the app. You will see following in the Debug window.

```
Using mainboard GHI Electronics FEZ Spider version 1.0
Program Started
Network Down
Network is Up with following properties
IP:169.254.120.154
Web server started at http://169.254.120.154:80/
Dns:8.8.8.8
Dns:8.8.4.4
IP Address: 169.254.120.154
Subnet Mask: 255.255.0.0
Gateway: 192.168.1.1
```

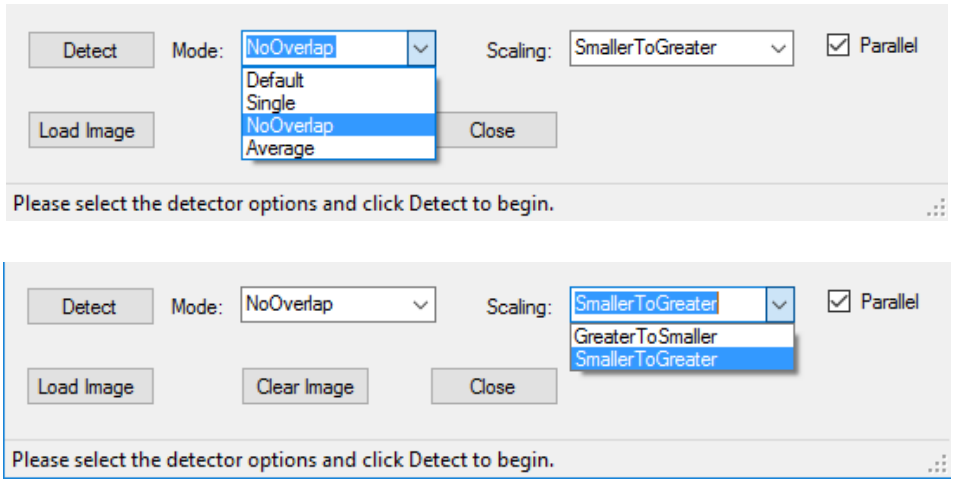
5. Go to the browser and type: `http://169.254.120.154/test` to check if server is responding properly.
6. If server is responding properly, you are ready to take picture. Press the button attached with the FEZ Spider kit and take a picture which contains a human face. Captured image will be displayed on the attached LCD.
7. Go to the browser and click on the given link `http://169.254.120.154/seepicture` to see the picture.
8. Now open "Face Detection" project in another visual studio window. And press F6 to build solution.
9. After successful build, press Start to run the application.

10. Press "Load Image" button on the application window to load picture captured by gadgeteer camera. You will see following.

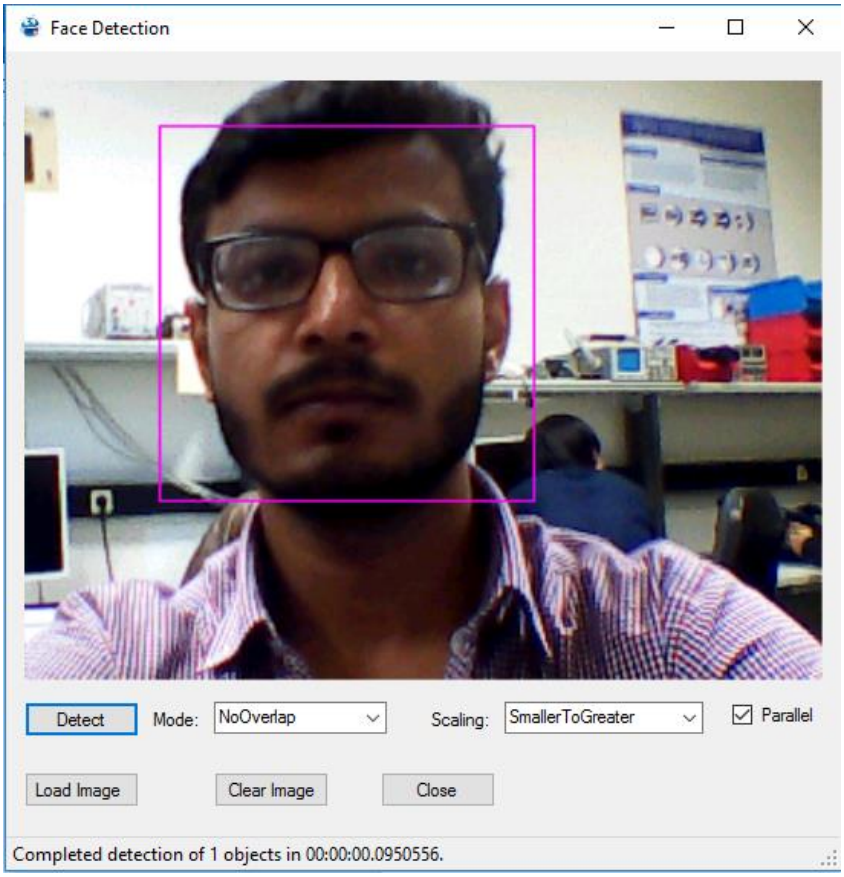


11. Once image is loaded in the app, now you can run face detection algorithm to detect faces in the image. You can set the mode of detector "Default", "Single", "Overlap" and "Average" depending on the number of faces in the image to improve the detection

and scaling “SmallerToGreater” or “GreaterToSmaller” for sliding window of detector as shown below.



12. After choosing appropriate Mode and Scaling, press Detect button. Machine learning algorithm will run on the image and if a face is detected, it will draw a square around it.



13. To detect faces on new image, click Clear Image button and to load new image, click Load Image button. Once new image is loaded, you can click Detect button to detect faces.

14. To close the app, click on Close button

3.2 Conclusion

Paul Viola and Michael Jones presented this approach which minimizes computation time, but still achieves highest accuracy. The first contribution is this technique computes rich set of image features using integral image. The integral image eliminates the need to compute multi scale image pyramid, which significantly reduces initial image processing required for object detection. The second contribution is use of AdaBoost for feature selection. Which is an aggressive technique for feature selection. So, the resulting classifier is not only computationally efficient but also simple to construct. Third contribution is use of cascade classifier which reduces computational time while maintaining accuracy.

Essentially, face detection is first step towards many advance computer vision problems, multimedia applications, biometric recognition, face recognition, face tracking and video surveillance.

Part 4: Further improvements

Extremely fast face detection makes it possible to have broad practical applications. Which include user interface, teleconferencing and image databases. The increase in speed enables real time face detection application possible when they were previously infeasible. In addition, this can be implemented on wide range of small low power devices, embedded processors and hand held devices.

No matter how better you become, there is always room for improvements. Similarly, we can extend and improve this project in many ways. Following are few ideas.

1. Face detection algorithm can be further extended to detect facial parts e.g. Eyes, Nose and lips.
2. Currently, this algorithm only works for frontal faces. We can further extend it for non-frontal face or profile detection.
3. This algorithm can also be extended for a face or object tracking problem.
4. We can also build a detector, which detects number of males and females in an image.
5. This can also be extended for facial recognition problem, in which we can set up a database of known faces and check whether under examine face matches any of them.
6. Currently, face detection algorithm only works for images. But, we can also further enhance this idea to detect faces in real time in video streaming. For that, we will have to write a TCP or UDP client server application for FEZ Spider.

Part 5: References

Following is the list of references used to create this document.

[1] Detecting faces in images: A survey in IEEE transactions on pattern analysis and machine intelligence, Vol 24, No 1, January 2002

[2] Yi-Qing Wang, An Analysis of the Viola-Jones Face Detection Algorithm, Image Processing (2014), pp. 128–148.

[3] P. Viola and M. J. Jones, Robust real-time face detection, International Journal of Computer Vision, 57 (2004), pp. 137–154. http://dx.doi.org/10.1023/B:VISI.0000013087.49260.fb.

[4] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In International Conference on Computer Vision, 1998.

[5] Paul Viola and Michael Jones, Rapid Object Detection using a Boosted Cascade of Simple Features. CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION 2001

[6] http://accord-framework.net/docs/html/R_Project_Accord_NET.htm

[7] https://raw.githubusercontent.com/accord-net/framework/development/Sources/Accord.Docs/Accord.Documentation/Diagrams/Classes/Accord.Vision.png

[8] http://accord-framework.net/docs/html/T_Accord_Vision_Detection_HaarObjectDetector.htm

[9] http://accord-framework.net/docs/html/T_Accord_Vision_Detection_HaarClassifier.htm

[10] http://accord-framework.net/docs/html/T_Accord_Vision_Detection_HaarCascadeStage.htm

[11] https://msdn.microsoft.com/en-us/library/dd492132.aspx

[12] https://www.ghielectronics.com/downloads/Gadgeteer/NET_Gadgeteer_for_beginners.pdf

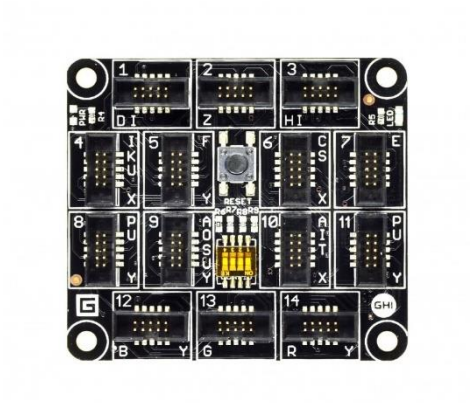
[13] https://www.ghielectronics.com/support/netmf

[14] https://www.ghielectronics.com/catalog/category/275

Appendix

FEZ Sider II

The FEZ Spider II is a .NET Gadgeteer mainboard running .NET Micro Framework, which enables users to program and debug the board from Microsoft's Visual Studio using C# or Visual Basic with a standard USB cable, allowing developers to take advantage of the extensive built-in libraries for networking, file systems, graphical interfaces and peripherals. .NET Gadgeteer allows developers to quickly add sensors and control modules to the board by utilizing a standard 10 pin socket and socket map. Click on the following links to learn more about the technology and what value is added by GHI Electronics to .NET Micro Framework and .NET Gadgeteer. [14]



Specifications:

32 - bit Processor 120 MHz, Flash- 2.87 MB, RAM- 13.67 MB, Networking- Ethernet/TCP/IP/Wi-Fi/SSL, USB Client/Host, Sockets- 14, Weight- 28g, Dimensions 57x52x11.65mm, File System- FAT16/32, Supported Image- BMP/GIF/JPG

Camera

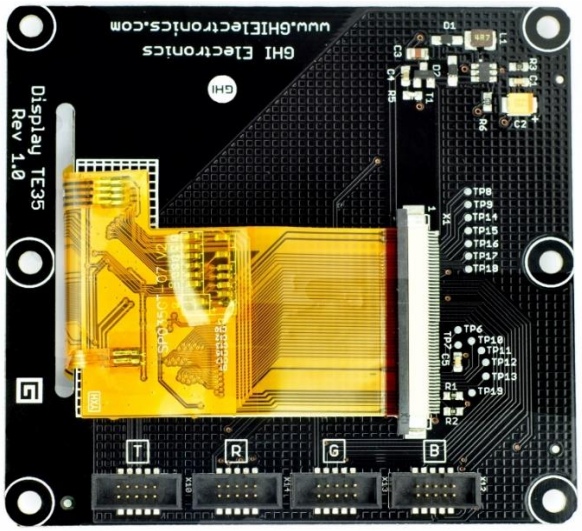
This USB Camera Module can stream images as large as 320x240 with up to 20fps on smaller images. [14]

Specifications:

Lens- 14mm, Resolution- 320x240, FPS-20, Dimension- 47x27x28.55mm

Display TE35

320x240 3.5" Color Display Module with a Touch Screen. [14]

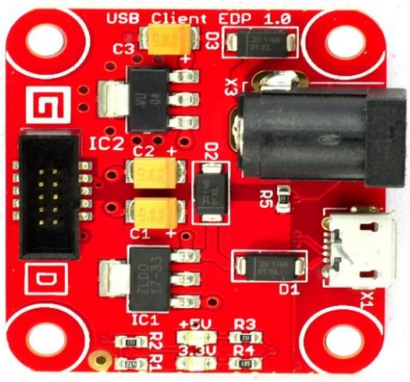


Specifications:

Resolution- 320x240, Pixel Clock- 28MHz, Power Consumption- 150 @ 3.3 V, 0 @ 5 V mA, Dimensions 82x77x11.55 mm, Weight 71 g

USB Client EDP

The USB Client EDP (Economical Dual Power), allows mainboards to be powered from USB and/or through a power pack with a 2.1mm barrel jack connector. While the module will run 6V to 9V off the power jack, we highly recommend using 6V to reduce the amount of heat generated by the module, especially when using modules that require a lot of power, like displays. Another good option would be to use a powered USB Hub. [14]



Specifications:

Required Socket Type- D, Power Consumption- 800 @3.3 V, 800 @ 5 V mA, Dimensions- 37x37x15 mm, Voltage- 6V to 9V

Ethernet J11D

This Ethernet Module gives a Gadgeteer main board instant access to networks and the internet. Not all Ethernet modules work on every board. Make sure this module is compatible with your main board. This module is compatible with GHI's FEZ Spider main board. [14]

Specifications:

Socket Type-E, Power Consumption 42 @ 3.3 V, 0 @ 5 V mA, Dimensions 32x32x16 mm, weight-8g.

Button Module

Button Module for Gadgeteer compatible devices. This module is equipped with an on-board configurable LED. [14]

Specifications:

Socket Type- X or Y, Power Consumption- 10 @ 3.3 V, 0 @ 5 V mA, Dimensions- 22x22x15 mm, weight- 1 g