UB CS550 2012

EMGU CV Tutorial

Agenda

- **Part I : Overview of Emgu CV**
- **Part II: Setup the required tools**
- **Part III: Run the Samples**
- **Part IV: Show image from a camera stream or video**
- **Part V: Optional Convert the previous Open CV tutorial to Emgu CV.**

Part I

Description

Emgu CV is a cross platform .Net wrapper to the OpenCV image processing library. Allowing OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++, IronPython etc. The wrapper can be compiled in Mono and run on Windows, Linux, Mac OS X, iPhone, iPad and Android devices.

Overview of Emgu CV Versions:

Name	Emgu CV (Open Source)	Emgu CV (Commercial Optimized)	Emgu CV for iOS (Commercial)	Emgu CV for Android (Beta)
OS	Windows, Linux, Mac OSX	Windows	iOS (iPhone, IPad, IPod Touch)	Android
Supported CPU Architecture	i386, x64	i386, x64	armeabi, armeabi-v7, i386 (Simulator)	armeabi, armeabi-v7a, x86
GPU Processing	√	√	X	X
Machine Learning	√	√	√	√
Tesseract OCR	√	√	√	√
Intel TBB (multi-thread)	X	√	X	X
Intel IPP (high performance)	X	√	X	X
Intel C++ Compiler (fast code)	X	√	X	X
Exception Handling	√	√	√	√
Debugger Visualizer	√	√	X	X
Emgu.CV.UI	√	√	X	X
License	GPL	Commercial License	Commercial License	Commercial License

Advantage of Emgu CV

Cross Platform

Emgu CV is written entirely in C#. The benefit is that it can be compiled in Mono and therefore is able to run on any platform Mono supports, including Linux, Mac OS X, iOS and Android.

Cross Language and comes with example code

Emgu CV can be used from several different languages, including C#, VB.NET, C++ and IronPython.

Other Advantages

- Image class with Generic Color and Depth
- Automatic garbage collection
- XML Serializable Image
- XML Documentation and intellisense support
- The choice to either use the Image class or direct invoke functions from OpenCV

  Although it is possible to create image by calling `CvInvoke.cvCreateImage`, it is suggested to construct a Image<TColor, TDepth> object instead. There are several advantages using the managed Image<TColor, TDepth> class

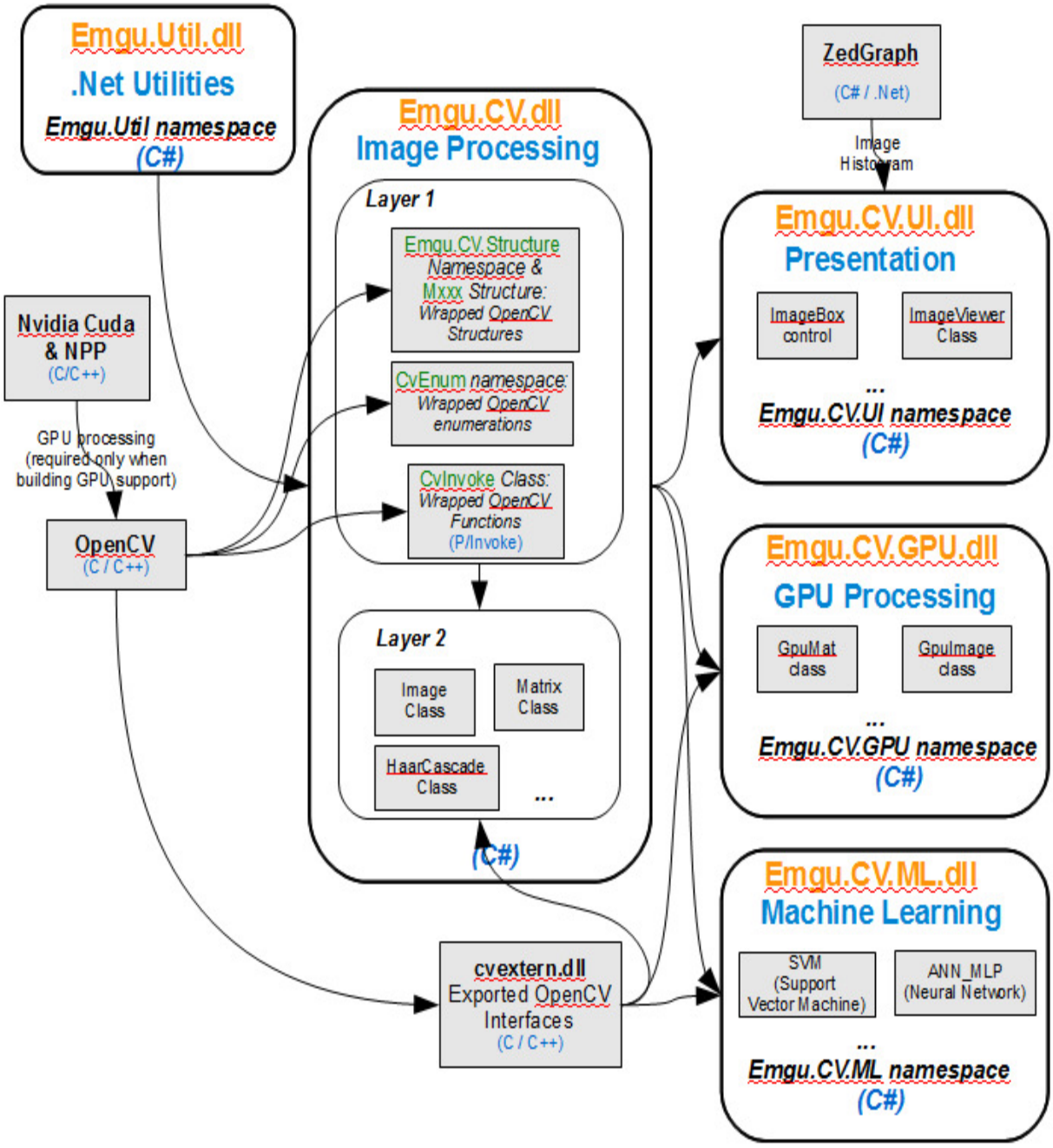
  - **Memory is automatically released by the garbage collector**
  - Image<TColor, TDepth> class can be examined by Debugger Visualizer
  - Image<TColor, TDepth> class contains advanced method that is not available on OpenCV, for example, generic operation on image pixels, conversion to Bitmap etc.

- Generic operations on image pixels

Architecture Overview

Emgu CV has two layers of wrapper as shown below

- The basic layer (***layer 1***) contains function, structure and enumeration mappings which directly reflect those in OpenCV.
- The second layer (***layer 2***) contains classes that mix in advantanges from the .NET world.



Wrapping OpenCV

Function Mapping - Emgu.CV.CvInvoke

The CvInvoke class provides a way to directly invoke OpenCV function within .NET languages. Each method in this class corresponds to a function in OpenCV of the same name. For example, a call to

```
 IntPtr image = CvInvoke.cvCreateImage(new System.Drawing.Size(400, 300),
CvEnum.IPL_DEPTH.IPL_DEPTH_8U, 1);
```

is equivalent to the following function call in C

```
 IplImage* image = cvCreateImage(cvSize(400, 300), IPL_DEPTH_8U, 1);
```

Both of which create a 400x300 of 8-bit unsigned grayscale image.

Structure Mapping - Emgu.CV.Structure.M*xxx*

This type of structure is a direct mapping to OpenCV structures.

Emgu CV Structure	OpenCV structure
Emgu.CV.Structure.MIplImage	IplImage
Emgu.CV.Structure.MCvMat	CvMat
...	...
Emgu.CV.Structure.M*xxxx*	*xxxx*

The prefix *M* here stands for Managed structure.

Emgu CV also borrows some existing structures in .Net to represent structures in OpenCV:

.Net Structure	OpenCV structure
System.Drawing.Point	CvPoint
System.Drawing.PointF	CvPoint2D32f
System.Drawing.Size	CvSize
System.Drawing.Rectangle	CvRect

Enumeration Mapping - Emgu.CV.CvEnum

The CvEnum namespace provides direct mapping to OpenCV enumerations. For example, `CvEnum.IPL_DEPTH.IPL_DEPTH_8U` has the same value as `IPL_DEPTH_8U` in OpenCV; both of which equals `8`.

Depth and Color as Generic Parameter

An Image is defined by its generic parameters: **color** and **depth**. To create a 8bit unsigned Grayscale image, in Emgu CV it is done by calling

```
Image<Gray, Byte> image = new Image<Gray, Byte>( width, height);
```

Not only this syntax make you aware the color and the depth of the image, it also restrict the way you use functions and capture errors in compile time. For example, the `SetValue(TColor color, Image<Gray, Byte> mask)` function in Image<TColor, TDepth> class (version >= 1.2.2.0) will only accept colors of the same type, and mask has to be an 8-bit unsigned grayscale image. Any attempts to use a 16-bit floating point or non-grayscale image as a mask will results a compile time error!

Creating Image

Although it is possible to create image by calling `CvInvoke.cvCreateImage`, it is suggested to construct a Image<TColor, TDepth> object instead. There are several advantages using the managed Image<TColor, TDepth> class

- Memory is automatically released by the garbage collector
- Image<TColor, TDepth> class can be examined by Debugger Visualizer
- Image<TColor, TDepth> class contains advanced method that is not available on OpenCV, for example, generic operation on image pixels, conversion to Bitmap etc.

Image Color

The first generic parameter of the Image class specific the color of the image type. For example

```
Image<Gray, ...> img1;
```

indicates that `img1` is a single channel grayscale image.

Color Types supported include:

- Gray
- Bgr (Blue Green Red)
- Bgra (Blue Green Red Alpha)
- Hsv (Hue Saturation Value)
- Hls (Hue Lightness Saturation)
- Lab (CIE L*a*b*)
- Luv (CIE L*u*v*)
- Xyz (CIE XYZ.Rec 709 with D65 white point)
- Ycc (YCrCb JPEG)

Image Depth

Image Depth is specified using the second generic parameter `Depth`. The types of depth supported in Emgu CV 1.4.0.0 include

- Byte
- SByte
- Single (float)
- Double
- UInt16
- Int16
- Int32 (int)

Creating a new image

To create an 480x320 image of Bgr color and 8-bit unsigned depth. The code in C# would be

```
Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(480, 320);
```

If you wants to specify the background value of the image, let's say in Blue. The code in C# would be

```
Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(480, 320, new Bgr(255, 0, 0));
```

Reading image from file

Creating image from file is also simple. If the image file is "MyImage.jpg", in C# it is

```
Image<Bgr, Byte> img1 = new Image<Bgr, Byte>("MyImage.jpg");
```

Creating image from Bitmap

It is also possible to create an Image<TColor, TDepth> from a .Net Bitmap object. The code in C# would be

```
Image<Bgr, Byte> img = new Image<Bgr, Byte>(bmp); //where bmp is a Bitmap
```

Automatic Garbage Collection

The Image<TColor, TDepth> class automatically take care of the memory management and garbage collection.

Once the garbage collector decided that there is no more reference to the Image<TColor, TDepth> object, it will call the `Disposed` method, which release the unmanaged IplImage structure.

The time of when garbage collector decides to dispose the image is not guaranteed. When working with large image, it is recommend to call the `Dispose()` method to explicitly release the object. Alternatively, use the ***using*** keyword in C# to limit the scope of the image

```
using (Image<Gray, Single> image = new Image<Gray, Single>(1000, 800))
{
   ... //do something here in the image
} //The image will be disposed here and memory freed
```

Getting or Setting Pixels

The safe (slow) way

- Suppose you are working on an Image<Bgr, Byte>. You can obtain the pixel on the y-th row and x-th column by calling

```
Bgr color = img[y, x];
```

- Setting the pixel on the y-th row and x-th column is also simple

```
img[y,x] = color;
```

The fast way

- The Image pixels values are stored in the Data property, a 3D array. Use this property if you need to iterate through the pixel values of the image.

Methods

Naming Convention

- Method `XYZ` in Image<TColor, TDepth> class corresponds to the OpenCV function `cvXYZ`. For example, Image<TColor, TDepth>.Not() function corresponds to `cvNot` function with the resulting image being returned.
- Method `_XYZ` is usually the same as Method `XYZ` except that the operation is performed **inplace** rather than returning a value. For example, Image<TColor, TDepth>._Not() function performs the bit-wise inversion inplace.

Operators Overload

The operators `+ - * /` has been overloaded (version > 1.2.2.0) such that it is perfectly legal to write codes like:

Image<Gray, Byte> image3 = (image1 + image2 - 2.0) * 0.5;

Generic Operation

One of the advantage of using Emgu CV is the ability to perform generic operations.

It's best if I demonstrate this with an example. Suppose we have an grayscale image of bytes

```
Image<Gray, Byte> img1 = new Image<Gray, Byte>(400, 300, new Gray(30));
```

To invert all the pixels in this image we can call the Not function

```
Image<Gray, Byte> img2 = img1.Not();
```

As an alternative, we can also use the generic method `Convert` available from the Image<TColor, TDepth> class

```
Image<Gray, Byte> img3 = img1.Convert<Byte>( delegate(Byte b) { return (Byte) (255-b); }
);
```

The resulting image `img2` and `img3` contains the same value for each pixel.

At first glance it wouldn't seems to be a big gain when using generic operations. In fact, since OpenCV already has an implementation of the `Not` function and performance-wise it is better than the generic version of the equivalent `Convert` function call. However, there comes to cases when generic functions provide the flexibility with only minor performance penalty.

Let's say you have an `Image<Gray, Byte> img1` with pixels set. You wants to create a single channel floating point image of the same size, where each pixel of the new image, correspond to the old image, described with the following delegate

```
delegate(Byte b) { return (Single) Math.cos( b * b / 255.0); }
```

This operation can be completed as follows in Emgu CV

```
Image<Gray, Single> img4 = img1.Convert<Single>( delegate(Byte b) { return (Single)
Math.cos( b * b / 255.0); }  );
```

The syntax is simple and meaningful. On the other hand, this operation in OpenCV is hard to perform since equivalent function such as `Math.cos` is not available.

Drawing Objects on Image

The `Draw( )` method in Image< Color, Depth> can be used to draw different types of objects, including fonts, lines, circles, rectangles, boxes, ellipses as well as contours. Use the documentation and intellisense as a guideline to discover the many functionality of the `Draw` function.

Color and Depth Conversion

Converting an Image<TColor, TDepth> between different colors and depths are simple. For example, if you have `Image<Bgr, Byte> img1` and you wants to convert it to a grayscale image of Single, all you need to do is

```
Image<Gray, Single> img2 = img1.Convert<Gray, Single>();
```

Displaying Image

Using ImageBox

Emgu CV recommends the use of ImageBox control for display purpose, for the following reasons

- ImageBox is a high performance control for displaying image. Whenever possible, it displays a Bitmap that shares memory with the Image object, therefore no memory copy is needed (very fast).
- The user will be able to examine the image pixel values, video frame rates, color types when the image is being displayed.
- It is convenient to perform simple image operations with just a few mouse clicks.

Converting to Bitmap

The Image class has a `ToBitmap()` function that return a Bitmap object, which can easily be displayed on a PictureBox control using Windows Form.

XML Serialization

Why do I care?

One of the future of Emgu CV is that Image<TColor, TDepth> can be XML serialized. You might ask why we need to serialization an Image. The answer is simple, we wants to use it in a web service!

Since the Image<TColor, TDepth> class implements ISerializable, when you work in WCF (Windows Communication Fundation), you are free to use Image<TColor, TDepth> type as parameters or return value of a web service.

This will be ideal, for example, if you are building a cluster of computers to recognize different groups of object and have a central computer to coordinate the tasks. I will also be useful if your wants to implement remote monitoring software that constantly query image from a remote server, which use the `Capture` class in Emgu CV to capture images from camera.

Conversion to XML

You can use the following code to convert an `Image<Bgr, Byte> image` to `XmlDocument`:

```
StringBuilder sb = new StringBuilder();
(new XmlSerializer(typeof(Image<Bgr, Byte>))).Serialize(new StringWriter(sb), o);
XmlDocument xDoc = new XmlDocument();
xDoc.LoadXml(sb.ToString());
```

Conversion from XML

You can use the following code to convert a `XmlDocument xDoc` to `Image<Bgr,Byte>`

```
Image<Bgr, Byte> image = (Image<Bgr, Byte>)
(new XmlSerializer(typeof(Image<Bgr, Byte>))).Deserialize(new XmlNodeReader(xDoc));
```

Part II

1/You need Vstudio 2008 or later, the express edition can be downloaded from here:
http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-products

2/You need the latest version of Emgu CV from this link for Win X86/32bit version

http://sourceforge.net/projects/emgucv/files/latest/download

NOTE: otherwise select the appropriate version from here:

http://sourceforge.net/projects/emgucv/files/emgucv/2.4.2/

The defaults are fine so you can use them for the complete installation.

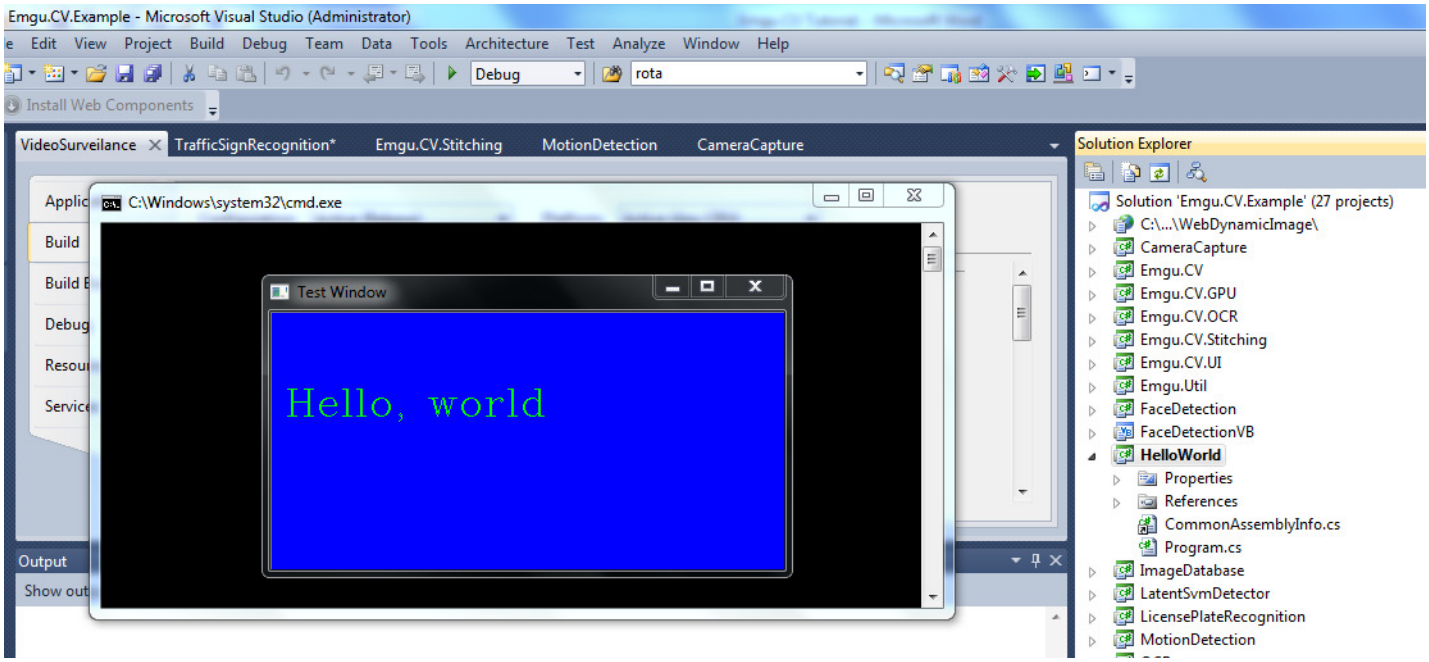
Part III

Go to the VS2010 folder in Emgu, path is as follows:

C:\Emgu\emgucv-windows-x86-gpu 2.4.2.1777\Solution\VS2010

and open **Emgu.CV.Example.sln**

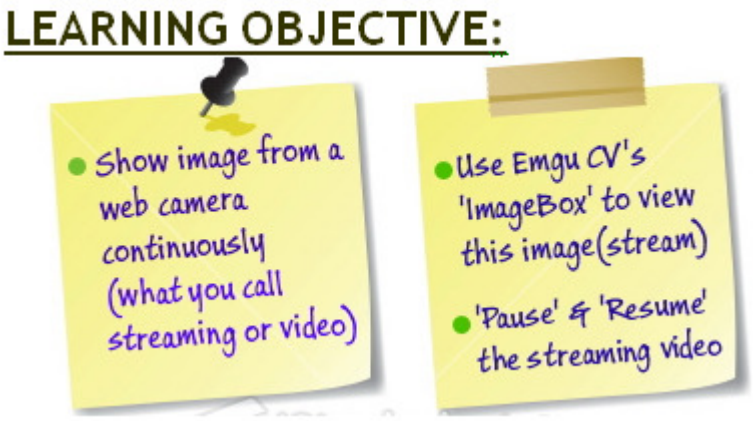
For that, Right-click the **Solution 'Emgu.CV.Example'** in the solution explorer and select **Build solution**. Then once it is built, click the debug button (green play button), now you should see a window with “hello world” pop up as shown below:



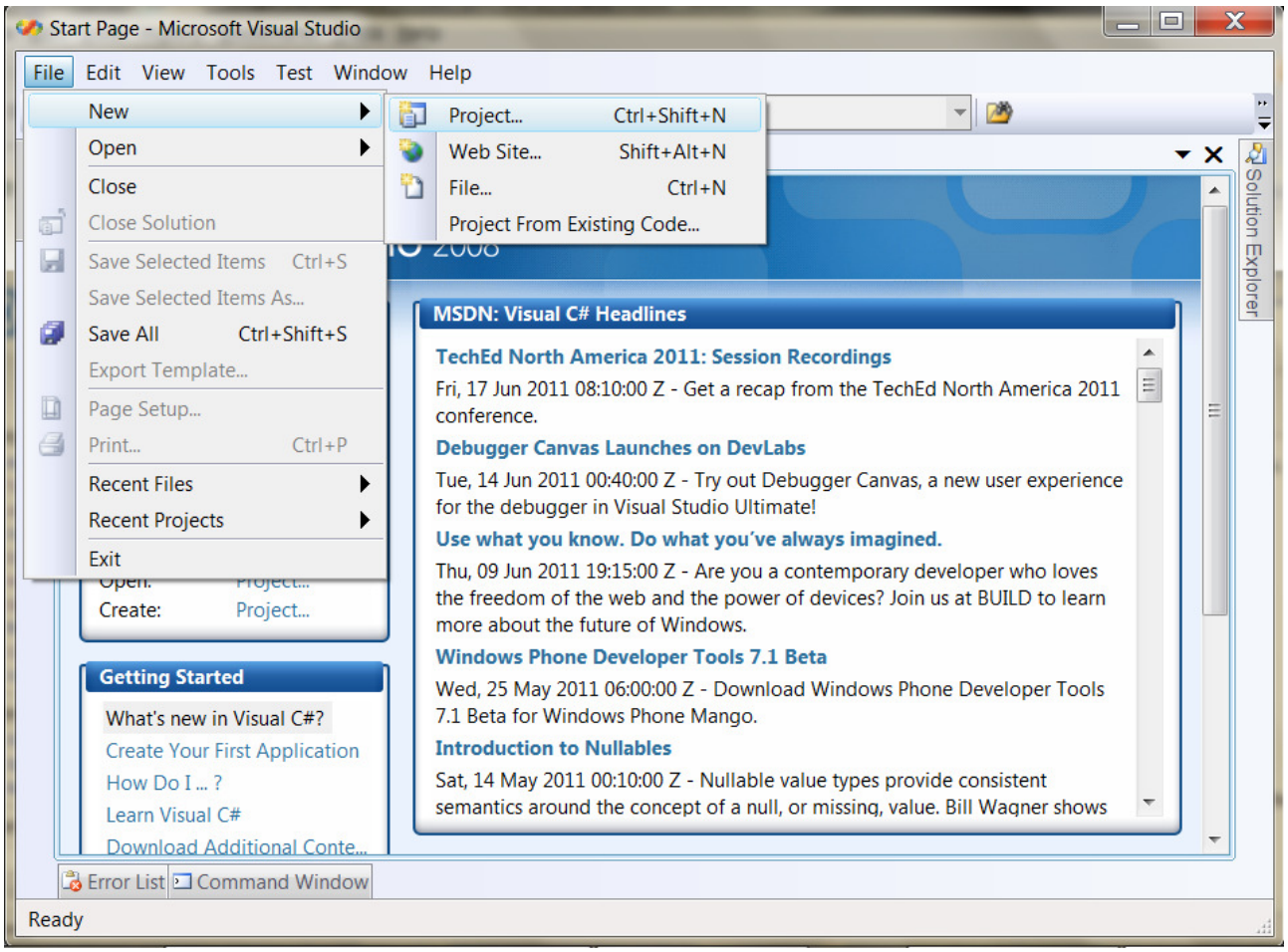
Congratulations! You just successfully configured your first EmguCV project!

After this is done, all the other EmguCV Examples are ready to be built and be debugged. You can experiment with them and use them as a model for similar tasks.

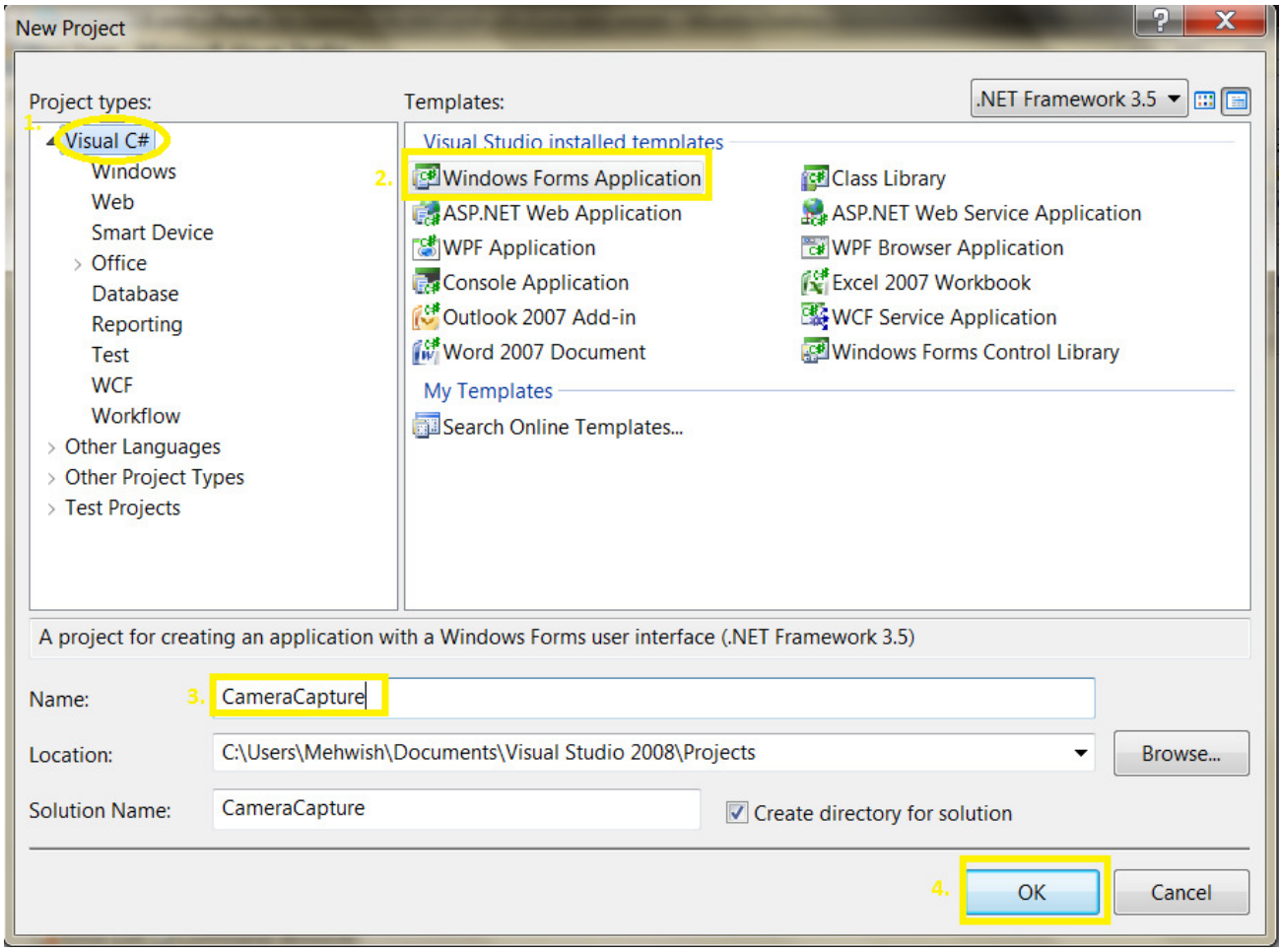
Part IV



STEP-1: open visual Studio 2010 and select **File-> New->Project** as follows:

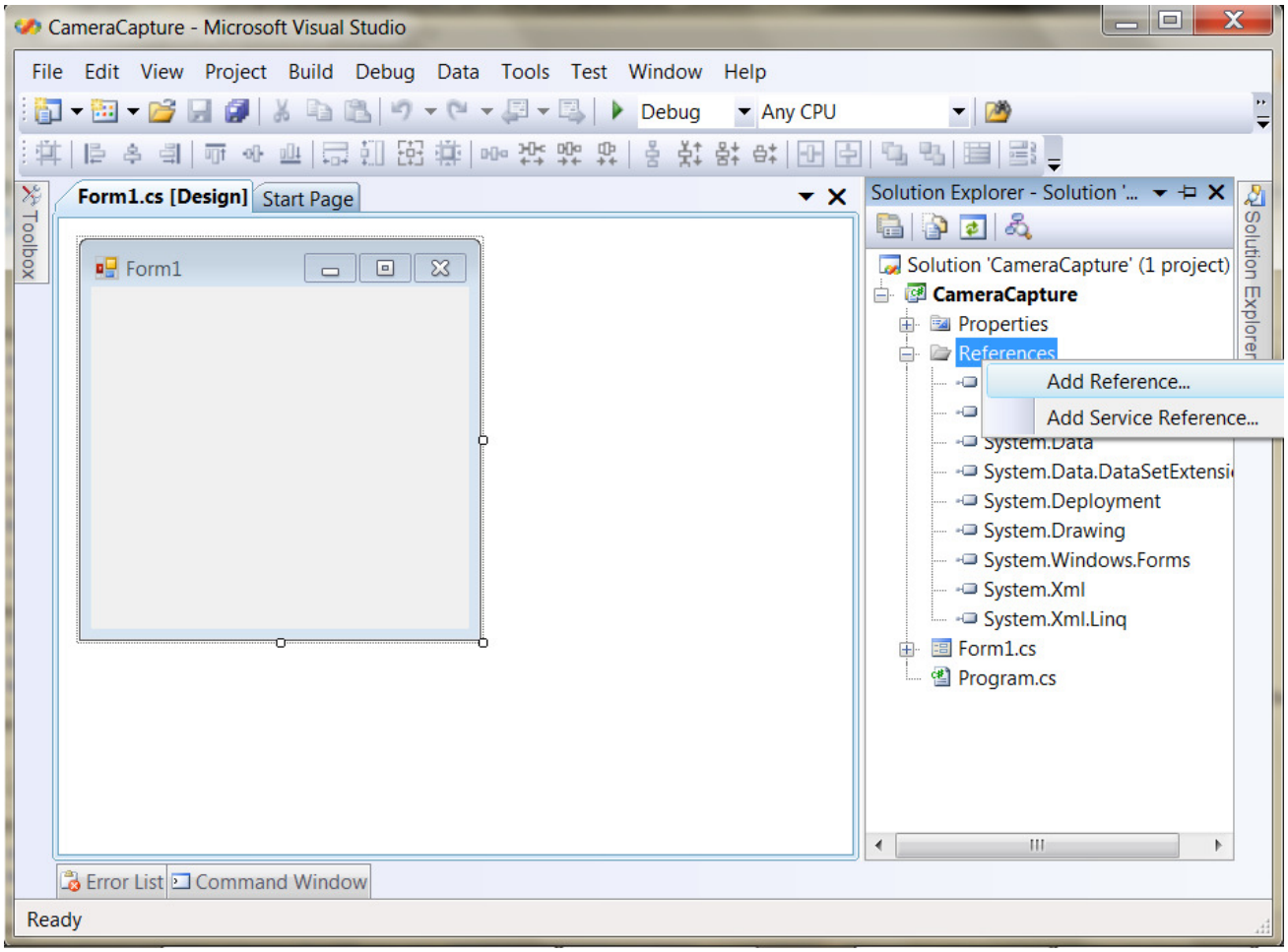


STEP-2: **in the** Visual C# **Project menu, Select** "Windows Forms Application" **and name the project** "CameraCapture" **as in figure below, and Click** "OK"

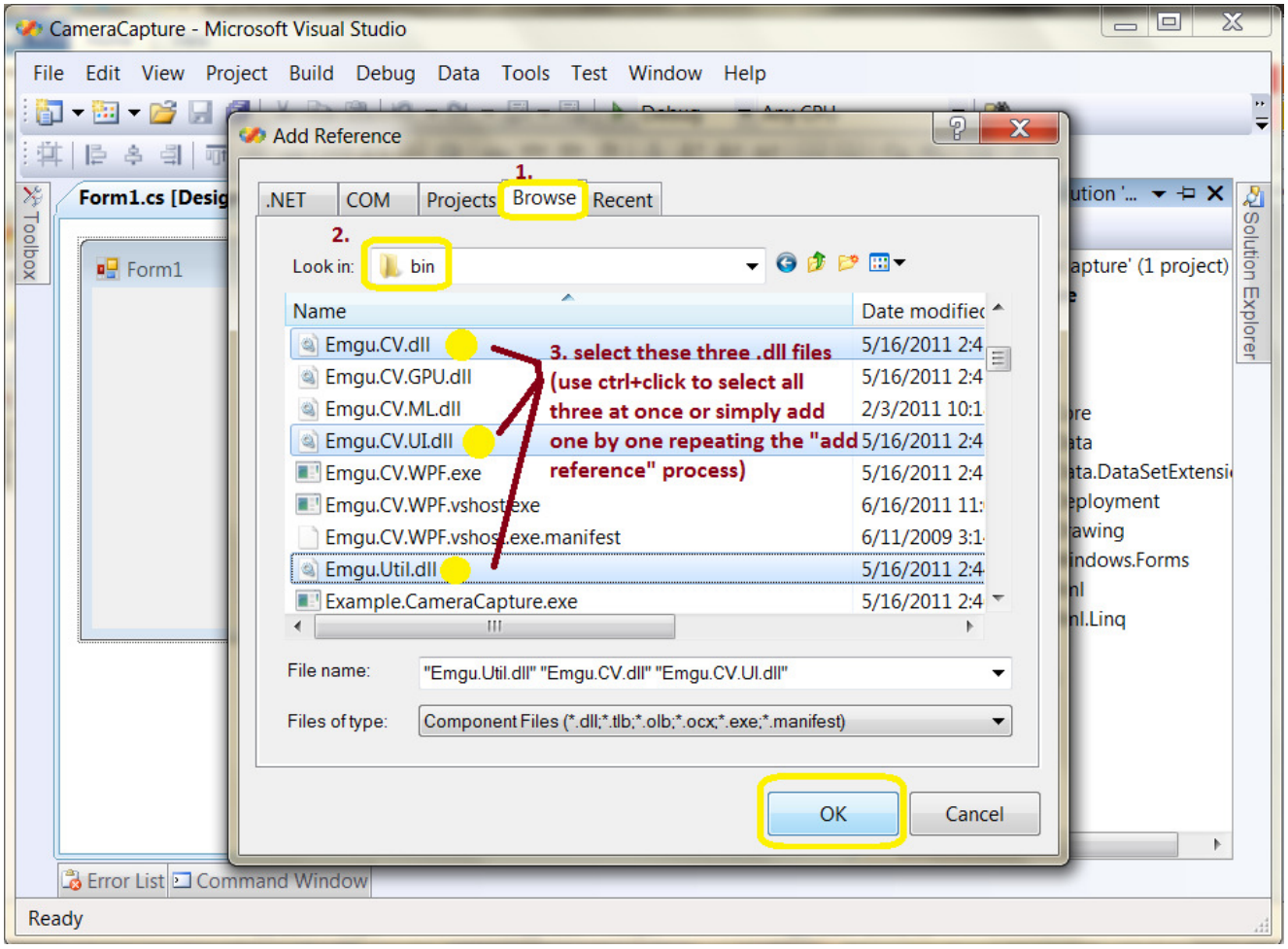


STEP-3: Lets first **add Emgu References** to our project.(though you can add them at any time later **BUT you must add references before** debugging.)

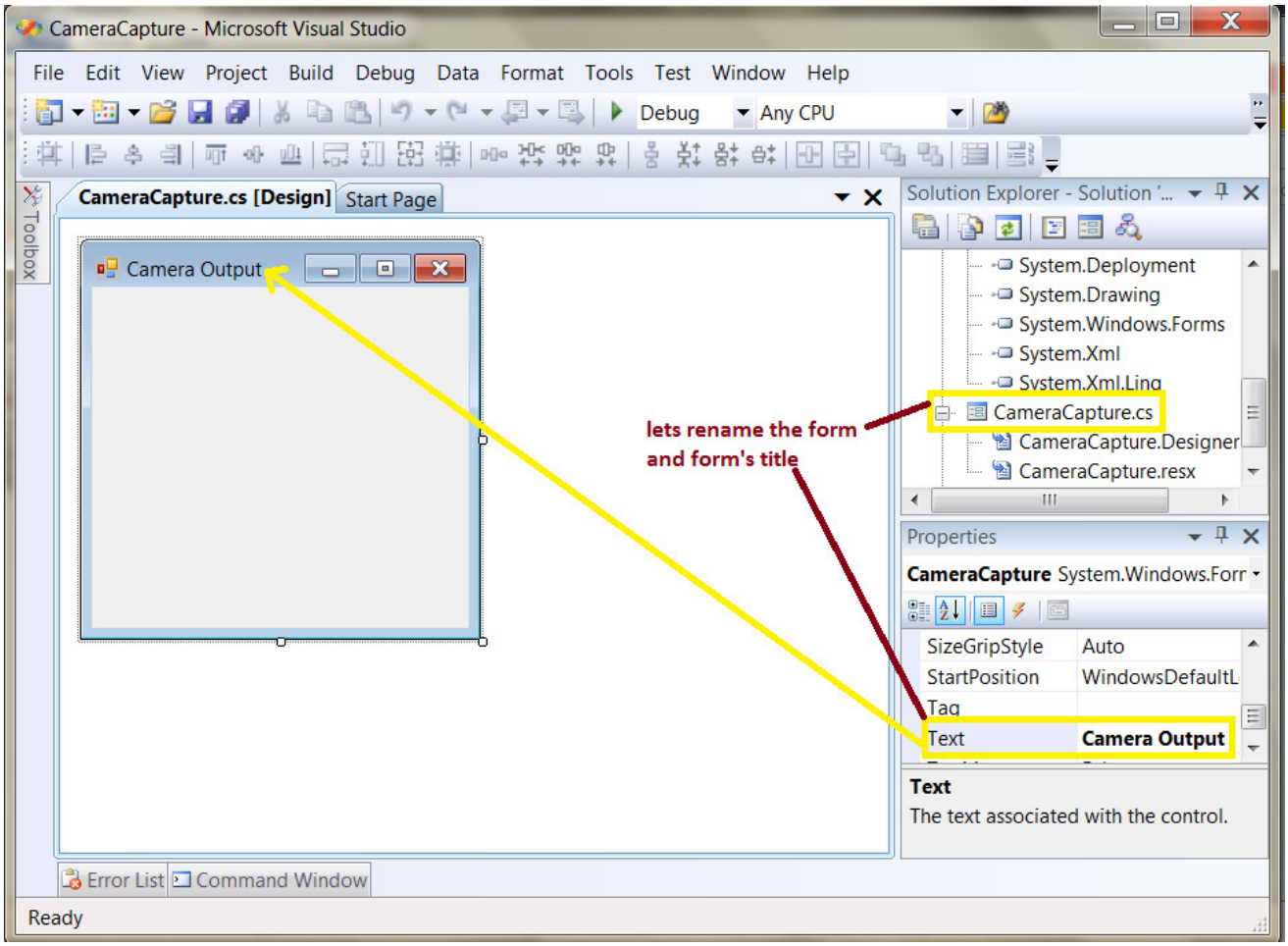
For this, just follow right-click project's "**References**" in Solution explorer, and select "**Add Reference**". the below fig shows so:



STEP-4: Select the **Browse** tab in the window that pops up, go to EmguCv's **bin**, and select the following 3 ***.dll*** files(***Emgu.CV.dll***, ***Emgu.CV.UI.dll*** and ***Emgu.Util.dll***) click **OK** to continue.



STEP-5: Rename ***Form1***.cs to ***CameraCapture***.cs and change its *Text* Field to "***Camera Output***"

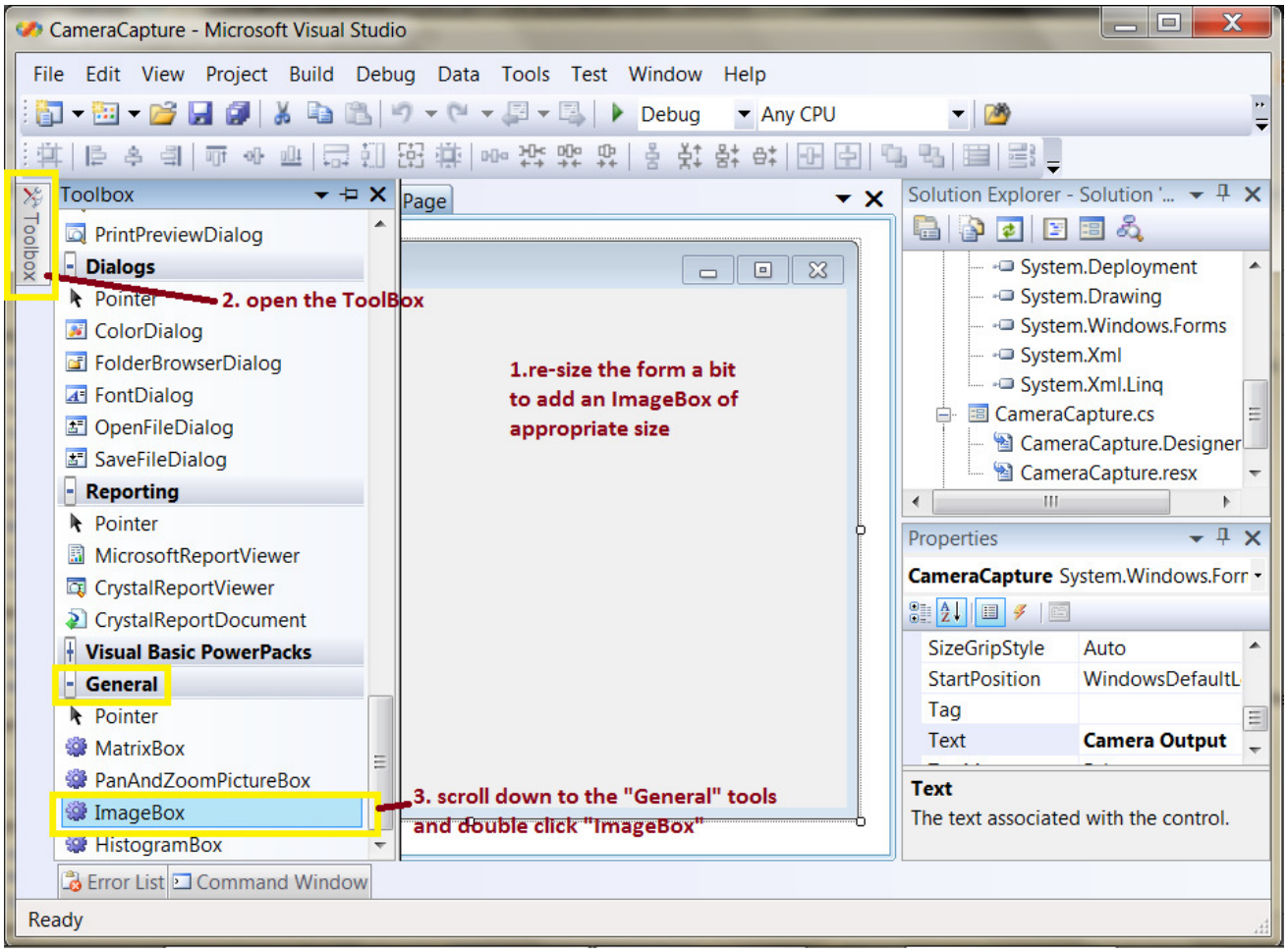


STEP-6:<u>**Add EmguCv Tools to your Visual Studio**</u>, because we will be using those tools, such as ImageBox, check Tip#4 at the end for the details or the link from the reference guide.

STEP-7: The next step is pretty simple, once you've added EmguCv tools to the Toolbox, as in fig below, just re-size the form and follow the instruction in this fig to ***<u>add an EmguCv ImageBox to the form</u>*** we created.

Note: You could use the already present PictureBox tool in Visual Studio instead of Emgu's ImageBox, but

1. it will be used a bit differently in coding than how its shown in this tutorial.
2. The purpose of using ImageBox here is to familiarize you with another aspect of EmguCv: i.e to use its own tools. They have more capabilities somewhat.



STEP-8: Add a button to the form and please do some more required "housekeeping" as below:

ImageBox Properties:

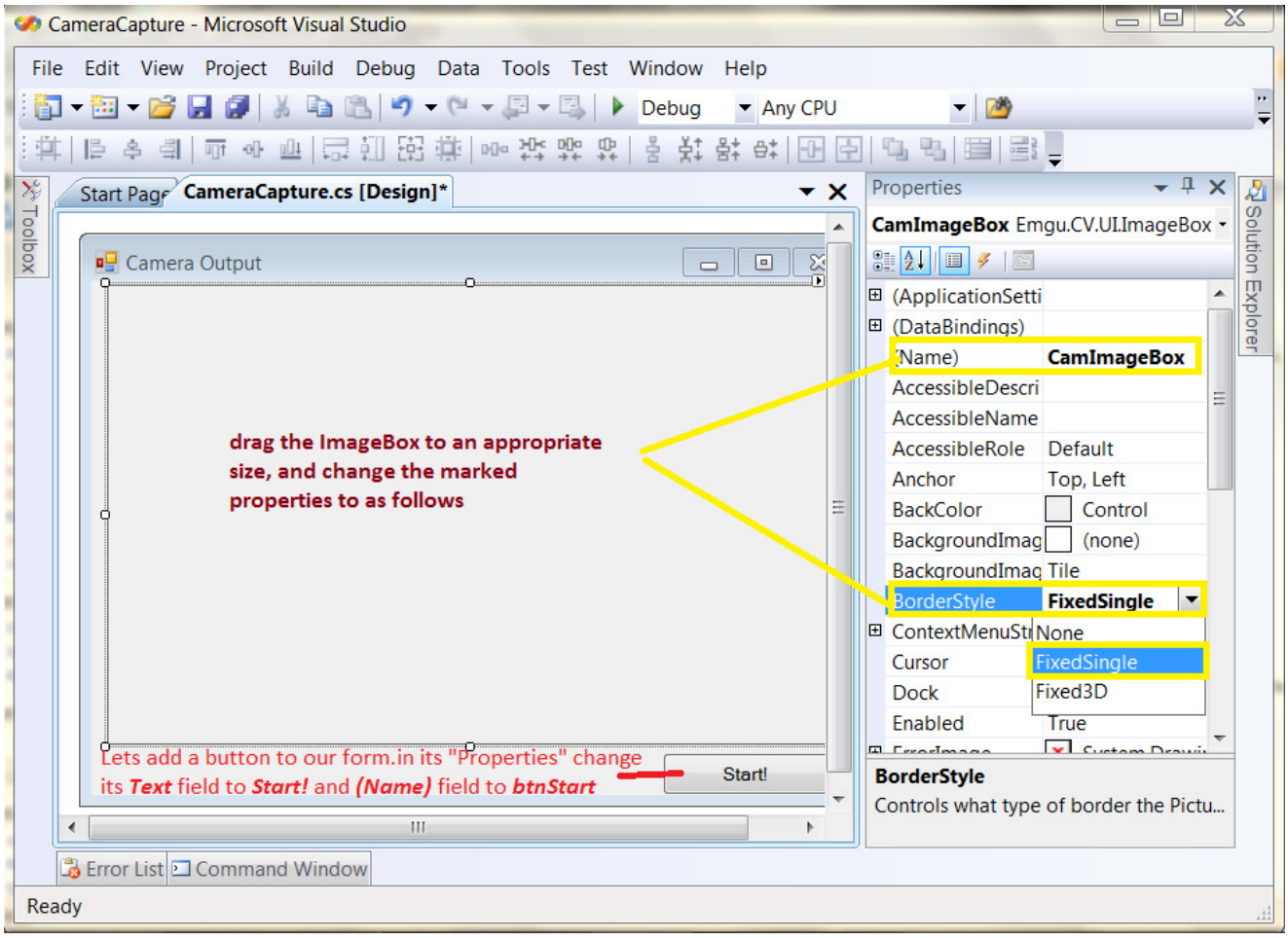
(Name) : CamImageBox

BorderStyle: Fixedsingle

Button properties:

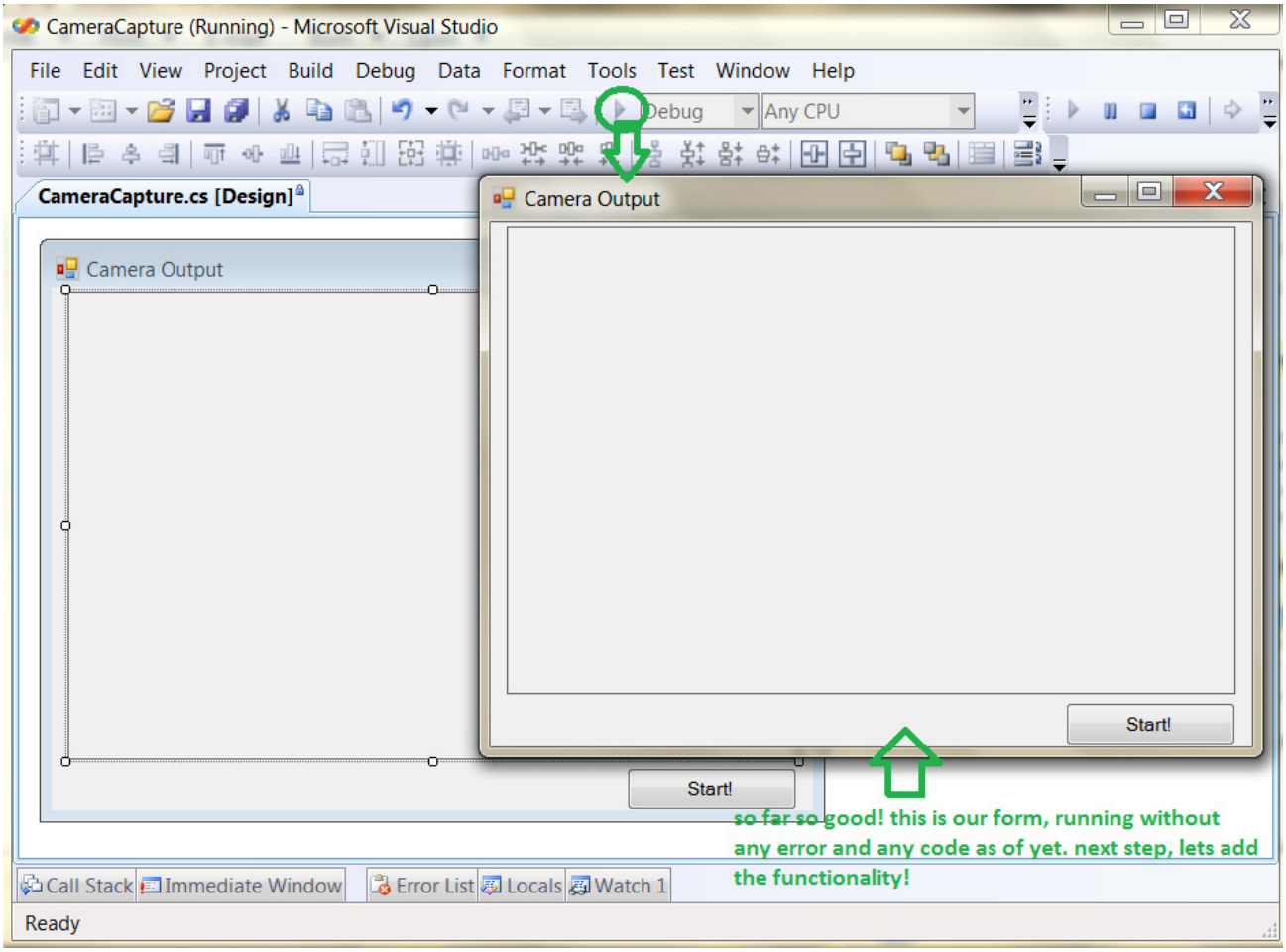
(Name) : btnStart

Text: Start!

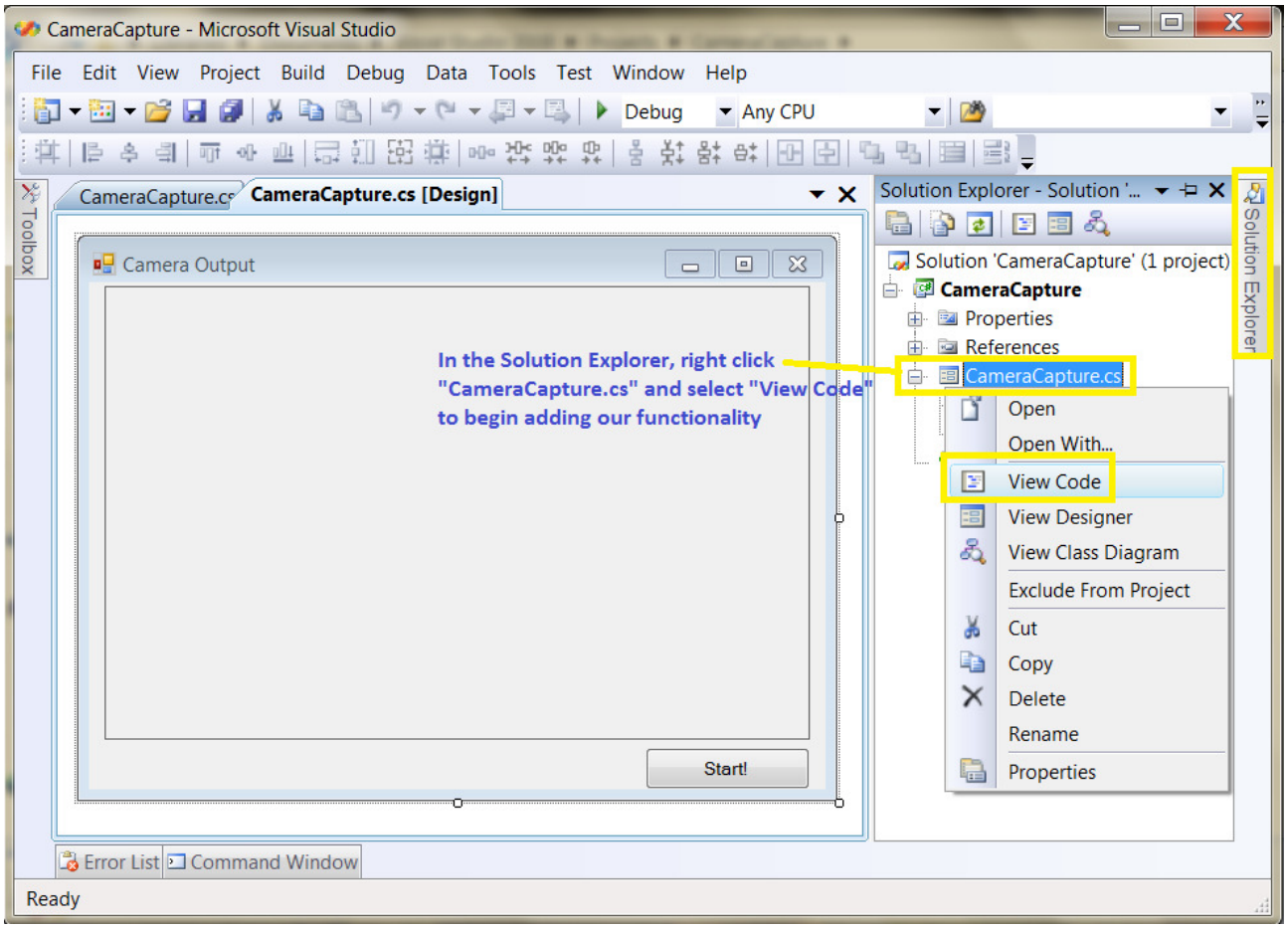


STEP-9: DEBUG & SAVE PROJECT

Okie dokie, so for now our User Interface is ready, all that's left is coding but before we code, lets **Debug** our project, it will automatically **Save** the project as well. We need to do that now. So let's see if our form is fit for run. The form should look something like below



STEP-10: to begin coding, view code behind CameraCapture.cs as in fig shown below



STEP-11: THE CODE

if unaltered and debugged, the code will only work if you've made the application like I did and more importantly if you kept the Components' Design Names EXACTLY the same as I did. Otherwise please modify it according to your component names

1. Welcome to your code view :) first things first, we had especially added the Emgu CV references, remember? Now in order to use them, add the following directives to the existing ones:

```
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.Util;
```

2. Now inside the class, declare the following global variables just above the *public CameraCapture()* initializer

```
//declaring global variables
private Capture capture;        //takes images from camera as image frames
```

```
private bool captureInProgress; // checks if capture is executing
```

3. Add a user defined function to your code and name it *Process Frame()* as shown below. In this function we'll create an EmguCv type image called ImageFrame. then capture a frame from camera and save it in "*ImageFrame*"(line 1). And then we'll load this into *CamImageBox* to show it to the user(line 2)

```
private void ProcessFrame(object sender, EventArgs arg)
{
      Image<Bgr, Byte> ImageFrame = capture.QueryFrame();
      CamImageBox.Image = ImageFrame;
}
```

To Display an EmguCV image in Windows Form Picture Box,
use this code instead of the one above:

```
//Show the image in Windows Form PictureBox called "pictureBox1"
pictureBox1.Image = ImageFrame.ToBitmap();
```

Asked by *Richard*: *"how to SAVE the captured image into specific directory or folder?"*
You can simply save the picture to anywhere like this:
Let's say, you wish to Save an EmguCV image (take the above 'ImageFrame' as an example) to your desired location, say, you want to save it at 'E:\ ' and wish to call it 'MyPic'.
Decide on the type of image you want it to be, and save it like so:

```
ImageFrame.Save(@"E:\MyPic.jpg");
```

NOTE: do NOT forget to give your image an extension! i like to save it as '.jpg' (saves space). you could try '.bmp'.
and do NOT forget the @ sign before your path string. it allows back slashes in a string, so helpful when giving a path in string.

4. Now it's time to add code behind the **Start** button. Go to **Design View** of *CameraCapture.cs* and double click the **Start** button you had added. it will take you back to the **code view** with an empty function of button click event as follows:

```
private void btnStart_Click(object sender, EventArgs e)
{
}
```

What we want is that when the ***Start***! button is pressed then camera should start working and the image stream should be visible in our ImageBox. if the stream is on, then the start button should display "Pause". and pressing it now should pause the stream and vice versa. right?

for that, we should create a new capture event if one isn't already created, i.e at startup it will create a capture event from which we will get frames from camera.
In case the capture was already created (i.e once the application had begun) then now it will do either of the following based on value of ***captureInProgress***:

when captureInProgress = true

then **Pause** the capture when btnStart is pressed. this is done by the code :

*Application.Idle -= ProcessFrame; //ProcessFrame() will be called here to **hold** its job*

when captureInProgress = false

then **Start** the capture when btnStart is pressed. this is done by the code :

Application.Idle += ProcessFrame; *//ProcessFrame() will be called here to **resume** its job*

*NOTE: this is why we had declared a bool type variable called **captureInProgress**, now do you get its use?*

Therefore,into the btnStart_Click() function, add the following code:

```
#region if capture is not created, create it now
if (capture == null)
{
    try
    {
        capture = new Capture();
    }
    catch (NullReferenceException excpt)
    {
        MessageBox.Show(excpt.Message);
    }
}
#endregion

if (capture != null)
{
    if (captureInProgress)
    {  //if camera is getting frames then stop the capture and set button Text
        // "Start" for resuming capture
        btnStart.Text = "Start!"; //
        Application.Idle -= ProcessFrame;
    }
    else
    {
        //if camera is NOT getting frames then start the capture and set button
        // Text to "Stop" for pausing capture
        btnStart.Text = "Stop";
        Application.Idle += ProcessFrame;
    }

    captureInProgress = !captureInProgress;
}
```

5. Last but not the least, add the following function to your code. which takes care of closing the application in a safe way.

```
private void ReleaseData()
        {
            if (capture != null)
                capture.Dispose();
        }
```

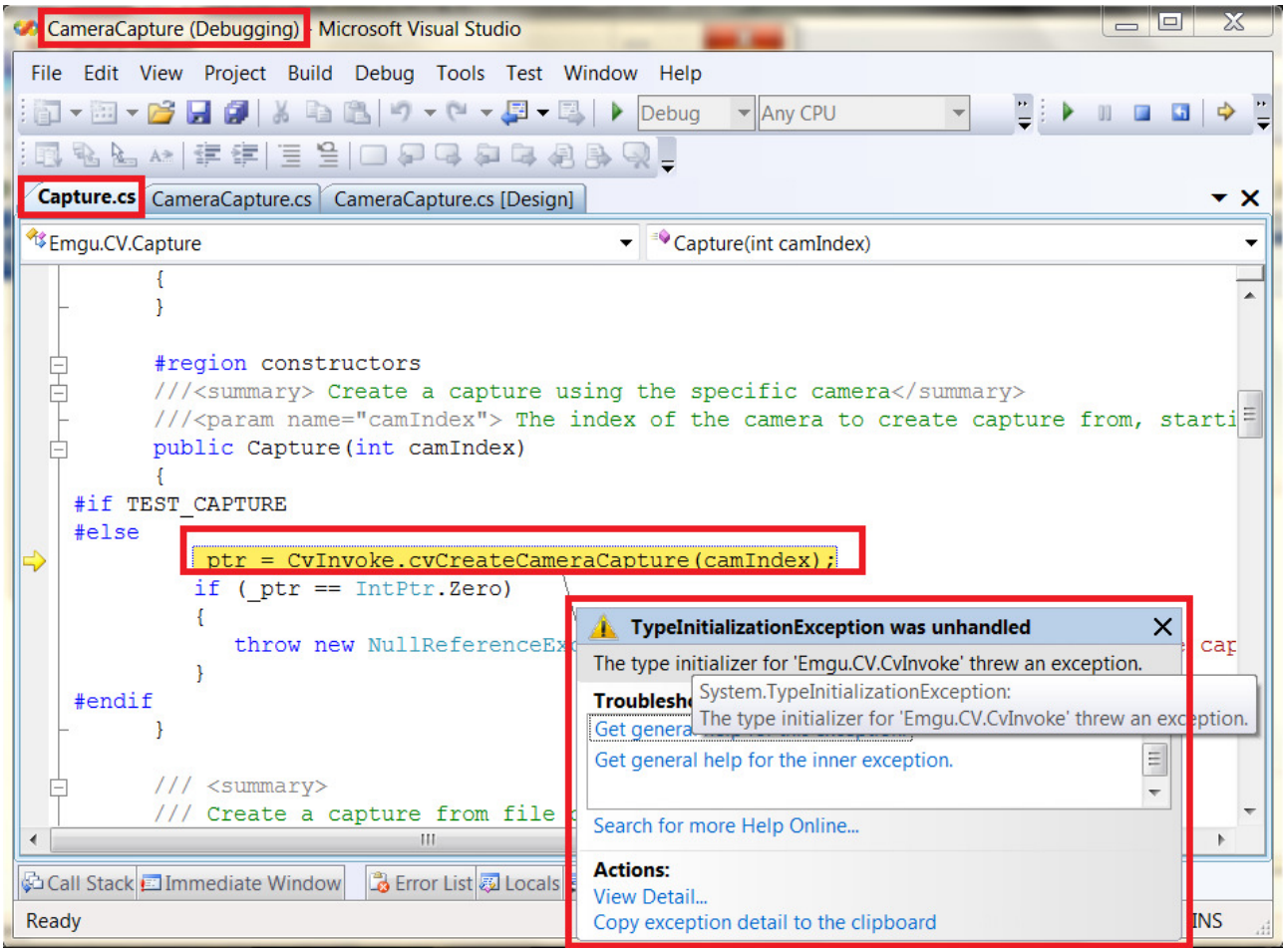
Now our Camera Capture Windows Form Application is all ready! what are waiting for? go on DEBUG it! :)

STEP 12- HANDLE ERROR

Emgu.CV.CvInvoke threw an exception!!!??

you tried to debug, but you got this BUGGING error? if you're making the application following our tutorial, please ignore this note below and continue reading this post...


IMPORTANT NOTE: FOR THOSE WHO HAVE NOT READ THE TUTORIAL FROM START OR ARE NOT MAKING THIS APPLICATION,
please read the Other post to solve your 'Emgu.CV.Invoke' error.



This error is telling you that your application just CAN'T connect to your Webcam!

DO SOME CHECKS:

Don't worry, here are some checks for you to do:

1.is the Emgu Cv's own example of Camera Capturing working perfectly?

you can find it at following path (if you installed the Emgu CV as we did)

C:\Emgu\emgucv-windows-x86 2.2.1.1150\Emgu.CV.Example\CameraCapture

go there and open CameraCapture.csproj file. it will open the solution for you. debug this solution and see if the application works or not.

if its showing you stream from your webcam then congratulations, at least your camera's working and rest assured that our application that we've just made is also perfectly as fit and fine because it is based on this EmguCV example itself.

2.did you follow every instruction carefully & correctly?

if yes, then there should be no errors in the project's **Error List** at the time you build and debug project. right?

if there are errors then you just made a mistake somewhere most probably. resolve that issue.

SOLUTION:

You might've tried various sites to find the solution to this problem. we did and trust me we couldn't find anything that worked!

they just talked about adding managed and unmanged code and dlls and what not.

well they weren't wrong! all they missed was the fact that ***WHICH*** dlls they were talking about and exactly ***where*** n ***how*** to put them!

so here's what we present:

Step 1: Add the Managed code to the folder where your .exe file is:

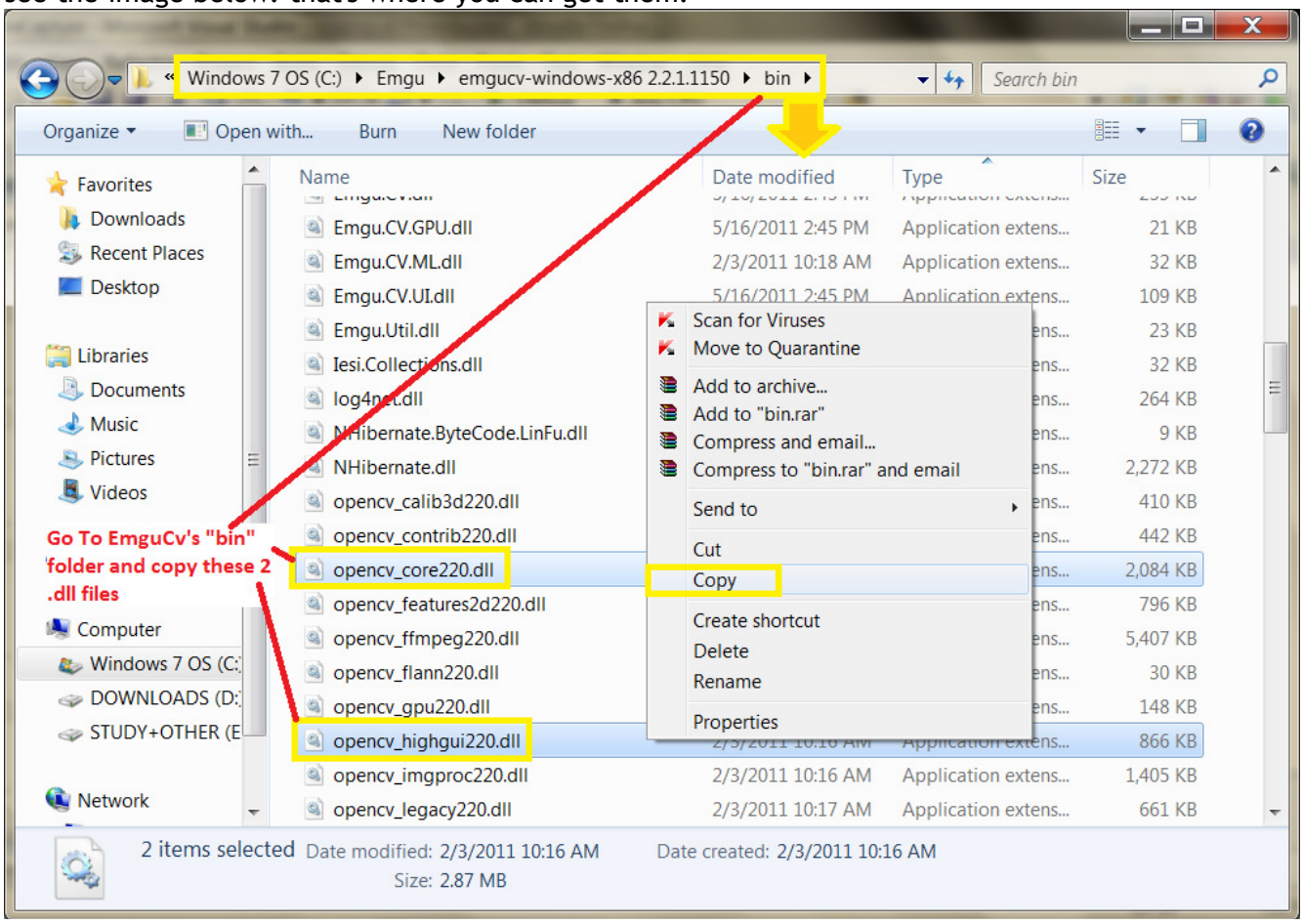
Do you remember adding the Emgu References? those 3 Emgu.CV dll files? that's it! they're the managed code you MUST add to your project. that's why i made you do that in the very first place! See your Debug folder(or Release folder if you have one) the Emgu.Cv dlls will be present like in the Camera Capture's Debug(second image below).

Step 2: Add the Unmanaged code to the folder where your .exe file is:

THIS IS THE SOLUTION TO YOUR PROBLEM DEAR FRIENDS!

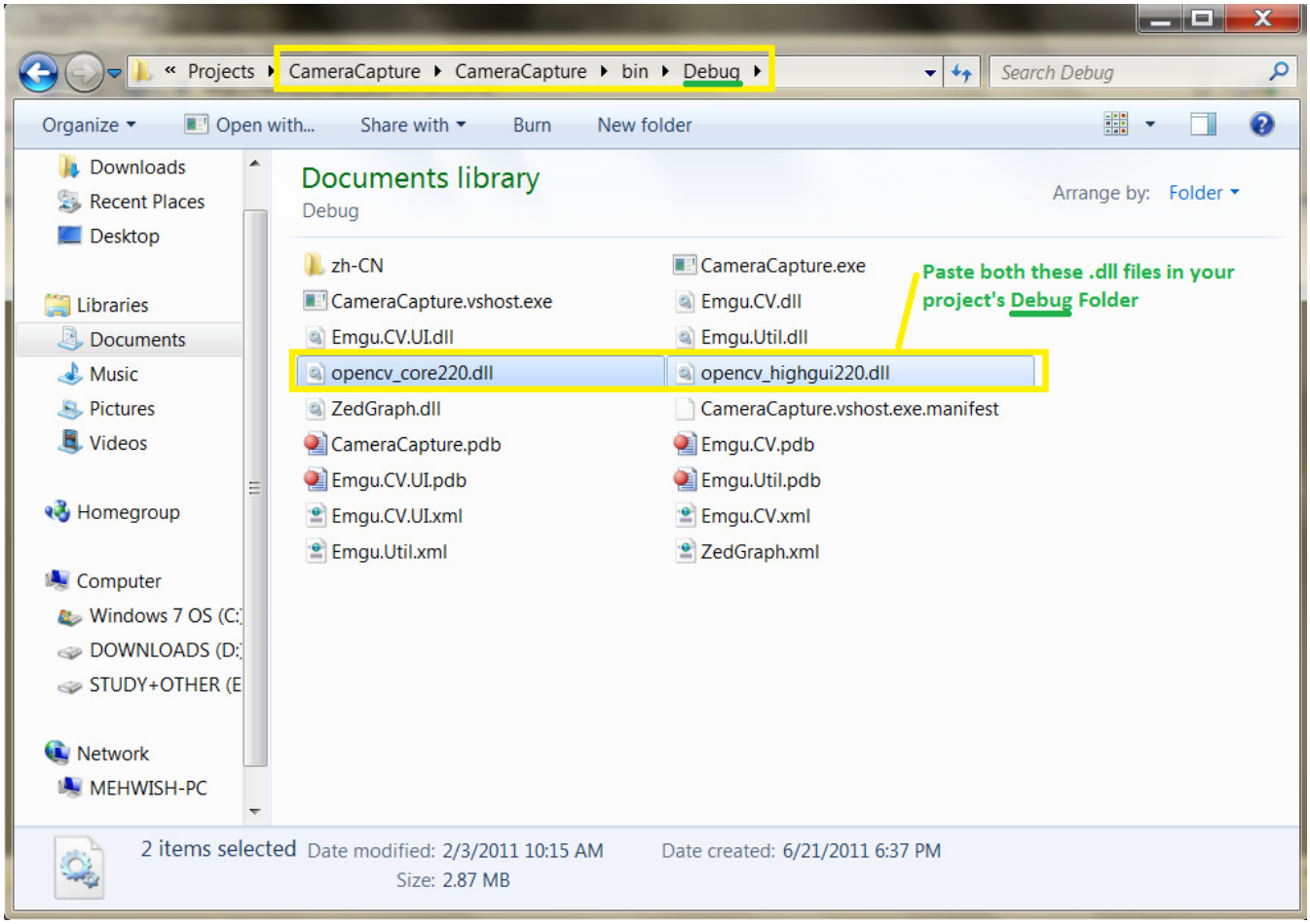
there are some opencv_xxx220.dlls in the bin folder of EmguCV directory.other than managed dlls we talked about, these are also NECESSARY to be present with your .exe file so that application works perfectly!

see the image below? that's where you can get them.



which ones to add to you project? well for Camera application these 2 as marked above are required: *opencv_core220.dll* and *opencv_highgui220.dll*.

Now paste it in your project's folder where your .exe file will execute, i.e the DEBUG folder, as shown below just paste the dlls you copied


« Projects ▸ CameraCapture ▸ CameraCapture ▸ bin ▸ Debug ▸
Search Debug
Organize
Open with...
Share with
Burn
New folder
Downloads
Recent Places
Desktop
Libraries
Documents
Music
Pictures
Videos
Homegroup
Computer
Windows 7 OS (C:)
DOWNLOADS (D:)
STUDY+OTHER (E
Network
MEHWISH-PC
Documents library
Debug
Arrange by: Folder
zh-CN
CameraCapture.vshost.exe
Emgu.CV.UI.dll
opencv_core220.dll
ZedGraph.dll
CameraCapture.pdb
Emgu.CV.UI.pdb
Emgu.CV.UI.xml
Emgu.Util.xml
CameraCapture.exe
Emgu.CV.dll
Emgu.Util.dll
opencv_highgui220.dll
CameraCapture.vshost.exe.manifest
Emgu.CV.pdb
Emgu.Util.pdb
Emgu.CV.xml
ZedGraph.xml
Paste both these .dll files in your project's Debug Folder
2 items selected
Date modified: 2/3/2011 10:15 AM
Size: 2.87 MB
Date created: 6/21/2011 6:37 PM

References

http://www.emgu.com/wiki/index.php/Main_Page

http://www.emgu.com/wiki/index.php/Download_And_Installation#Getting_the_Dependency

http://fewtutorials.bravesites.com/tutorials

http://www.emgu.com/wiki/index.php/Add_ImageBox_Control

Tips for possible issues

1/When running the examples after the install of EMGU it complains about a missing nvcuda.dll this is a workaround the issue:

Download nvcuda.zip from this link: http://www.dll-files.com/dllindex/dll-files.shtml?nvcuda

Extract **nvcuda.dll** from **nvcuda.zip**. We recommend that you extract **nvcuda.dll** to the installation directory of the program that is requesting **nvcuda.dll**.

If that doesn't work, you will have to extract **nvcuda.dll** to your system directory. By default, this is:

- `C:\Windows\System` (Windows 95/98/Me)
- `C:\WINNT\System32` (Windows NT/2000)
- `C:\Windows\System32` (Windows XP, Vista, 7)

If you use a 64-bit version of Windows, you should also place **nvcuda.dll** in `C:\Windows\SysWOW64\`

Make sure overwrite any existing files (but make a backup copy of the original file).

Reboot your computer.

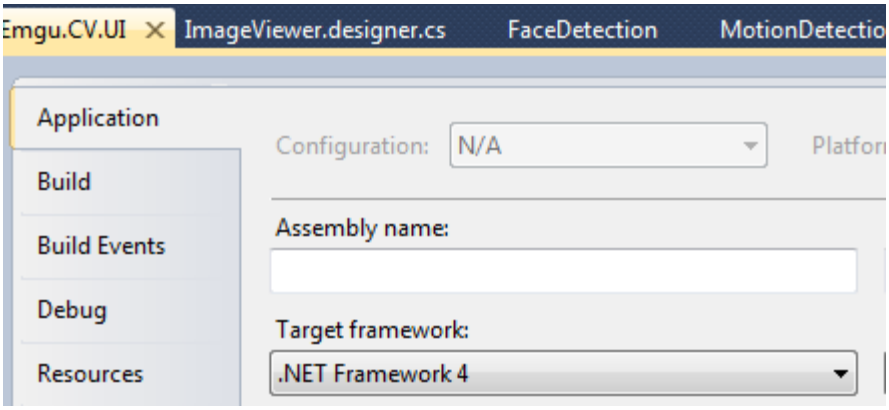
If the problem still occurs, try the following:

1. Open Windows Start menu and select "Run...".
2. Type `CMD` and press `Enter` (or if you use Windows ME, type `COMMAND`)).
3. Type `regsvr32 nvcuda.dll` and press `Enter`.

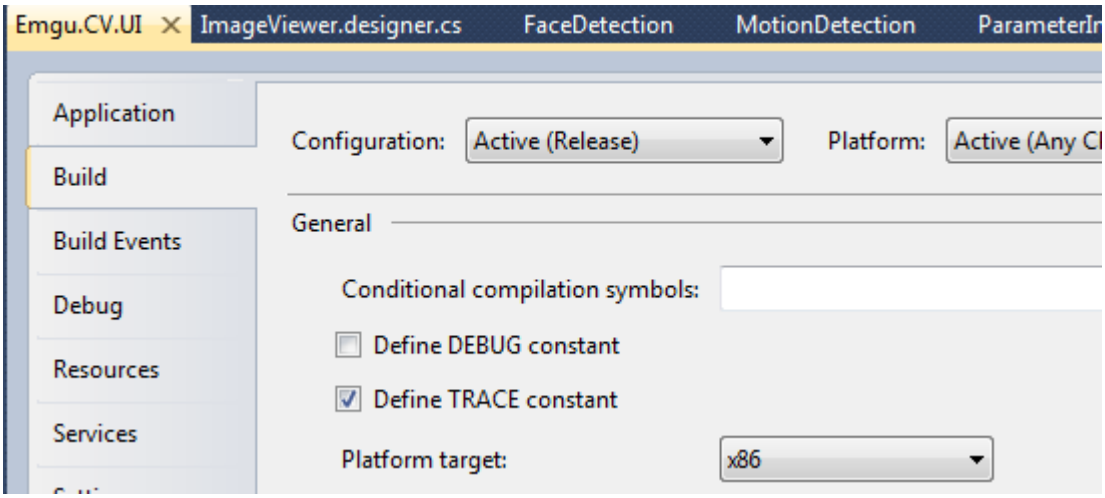
2/Emgu CV use WCF(Windows Communication Foundation) therefore requires .Net 3.0, although .net 4.0 worked better for me for some examples project, so I would recommend it, available from the Microsoft site:

http:// www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=17851

you can change the .net target platform by right clicking on the project in your solution explorer then select properties than on the Application tab select the appropriate target platform from the drop down list as shown below.



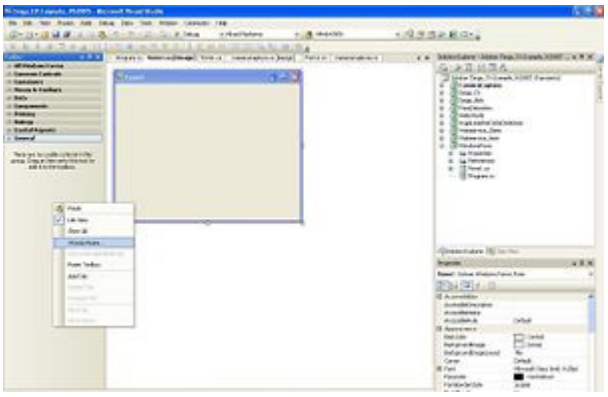
3/ If you downloaded a 32bit Emgu CV install on a 64bit OS you can still run it but make sure you change the platform target to x86 by right clicking on the project in your solution explorer then select properties than on the Build tab select x86 as shown below:



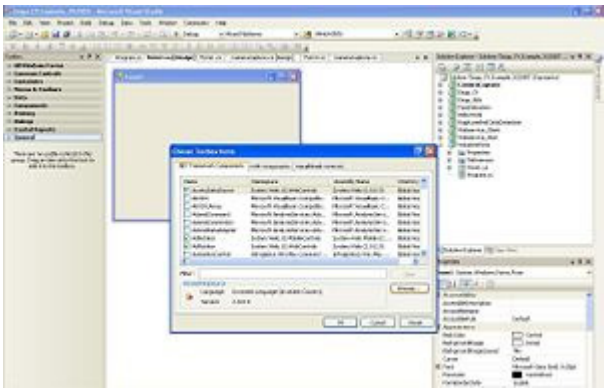
4/Add ImageBox ControlAdding ImageBox Control in Visual Studio (Windows Form)

http://www.emgu.com/wiki/index.php/Add_ImageBox_Control

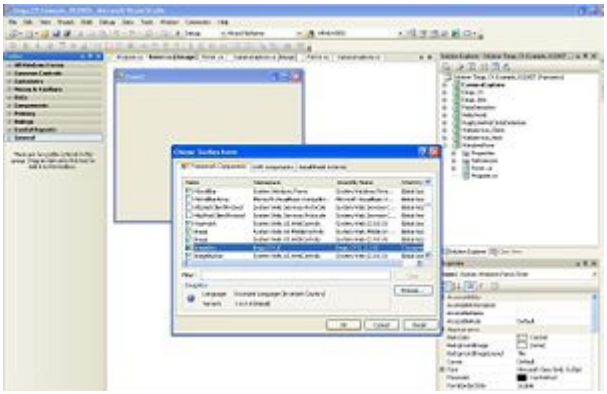
- Open your form in designer view. From Toolbox, right click in the empty space of 'General' column. This should pop up a selection menu, where 'Choose Items' selection is available, see below.



- Click on 'Choose Items', you will see a 'Choose Toolbox Item' Dialog, from where click the 'Browse..' button on the lower right corner of the dialog.



- Select 'Emgu.CV.UI.dll' file from 'Open' dialog, click the 'Open' button.
- Now you should notice the ImageBox control has been added to the 'Choose Toolbox Items' dialog. Click 'Ok'.



The ImageBox will be available from the 'General' column in the Toolbox area. Pull the control and drag it to the desired area on your form.