L2Bot using C# and Emgu

J. Ruszala

Visual Studio Express 2013 for Windows Desktop

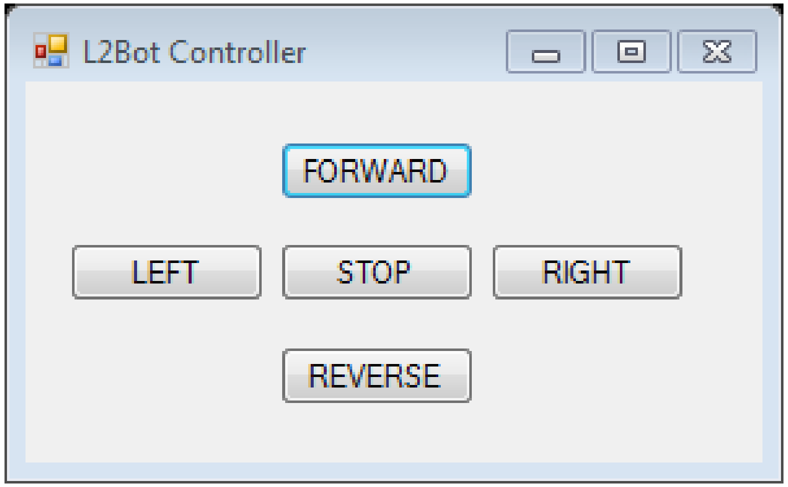
Visual Studio Express 2013 for Windows Desktop

- Visual Studio Express 2013 for Windows Desktop
  - Download from: http://www.visualstudio.com/downloads/download-visual-studio-vs#d-express-windows-desktop
  - Follow default installation options

L2Bot Button Control

L2Bot Button Control

- Objective: Create a Win Forms application to drive the L2Bot in any direction by using a graphic user interface (GUI)

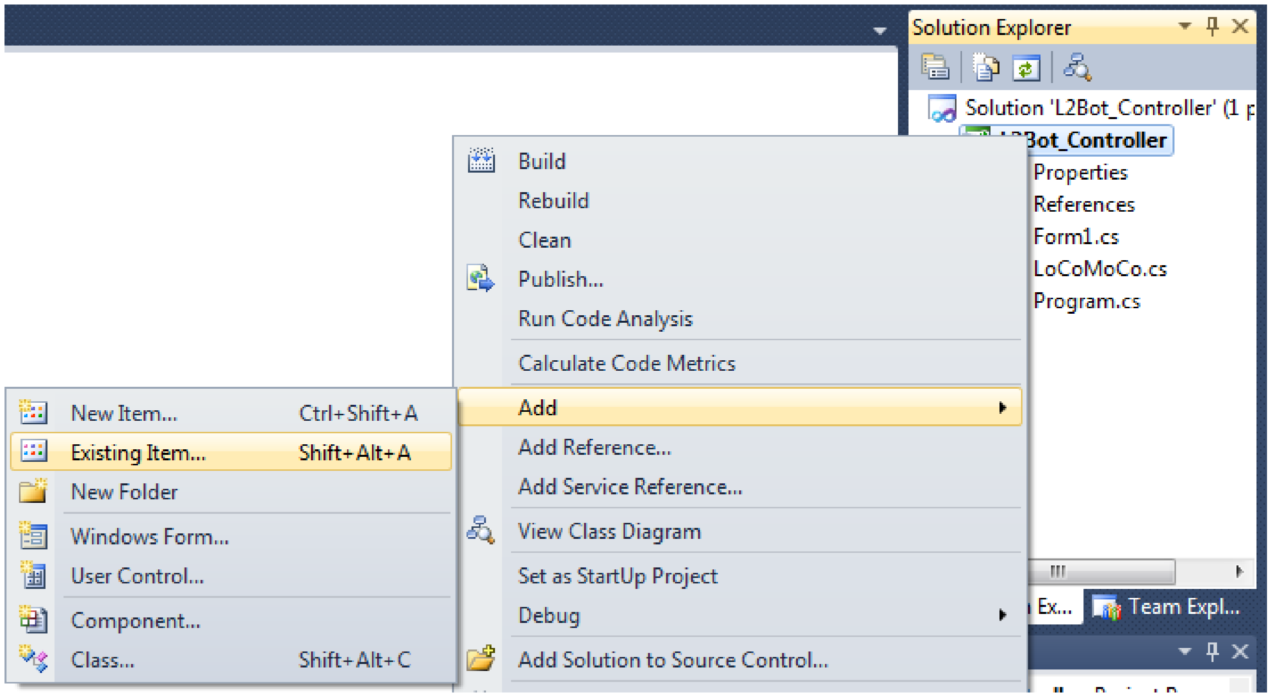


L2Bot Button Control

- Add five buttons to the windows form to control the robot:
  - Forward
  - Backward
  - Left
  - Right
  - Stop
- Giving the buttons meaningful names will help when referencing them in your C# file
- Double click each of the buttons to create an event handler

L2Bot Button Control

- The L2Bot motor controller is physically connected to the PC via a serial port. To access these serial port controls you must add the LoCoMoCo (Low Cost Motor Controller) class to your project.



L2Bot Button Control

- After adding the class to your project we are ready to start adding C# code
  - Add a class LoCoMoCo variable

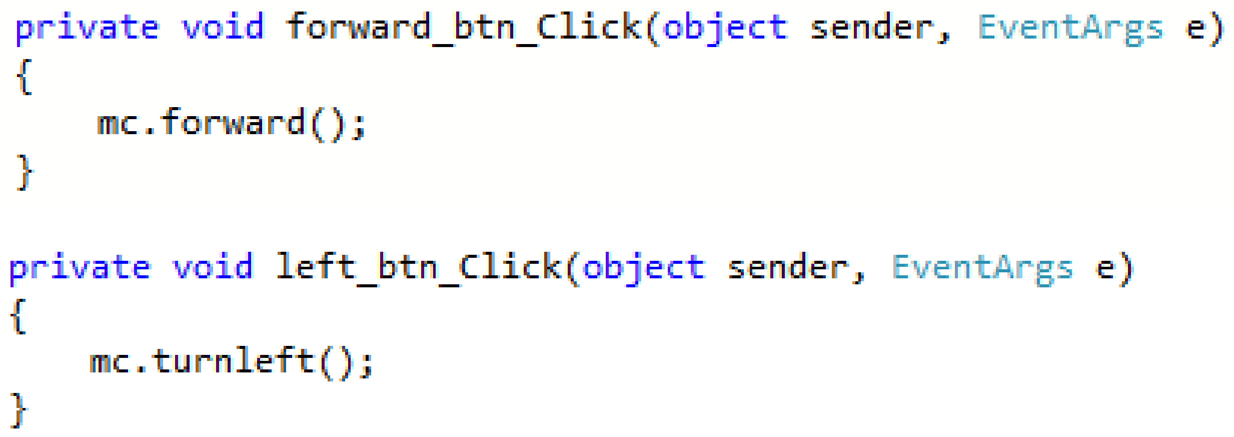
```
LoCoMoCo mc;
```

  - Set that variable to a new instance of LoCoMoCo in the Form1 method

```
public Form1()
{
    InitializeComponent();
    mc = new LoCoMoCo("COM7");
}
```

L2Bot Button Control

- For each of your button click event handlers add code to call the appropriate motor controller method

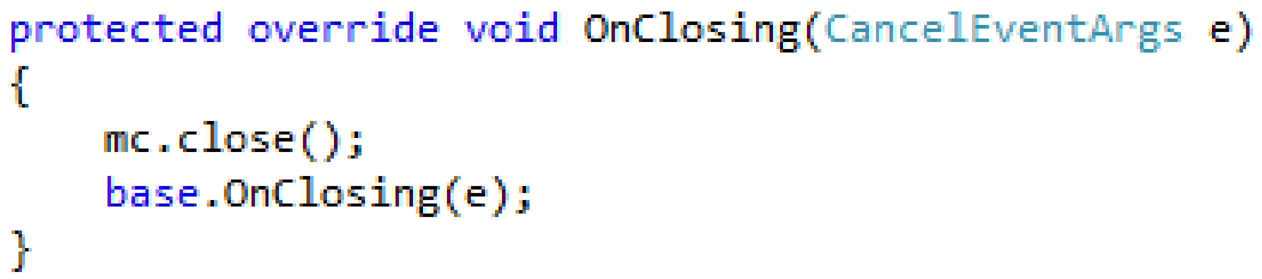


```
private void forward_btn_Click(object sender, EventArgs e)
{
    mc.forward();
}

private void left_btn_Click(object sender, EventArgs e)
{
    mc.turnleft();
}
```

L2Bot Button Control

- Add override method to close the serial port connection when the program is closed

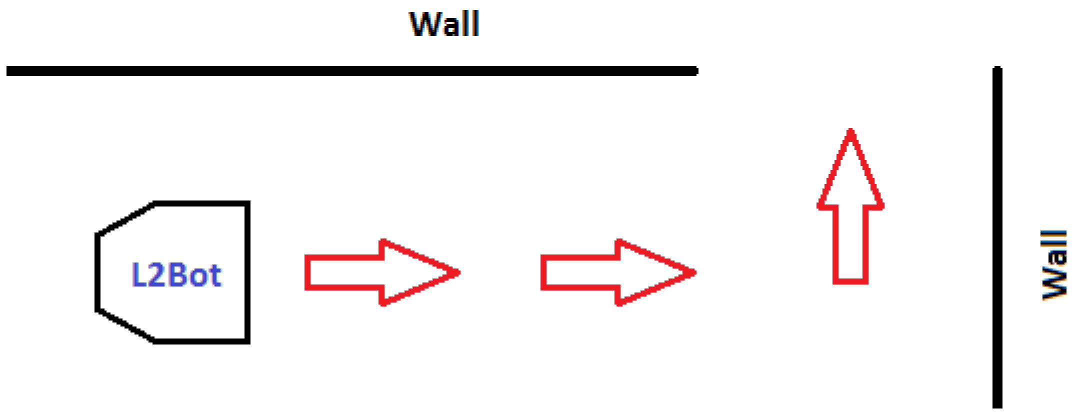


```
protected override void OnClosing(CancelEventArgs e)
{
    mc.close();
    base.OnClosing(e);
}
```

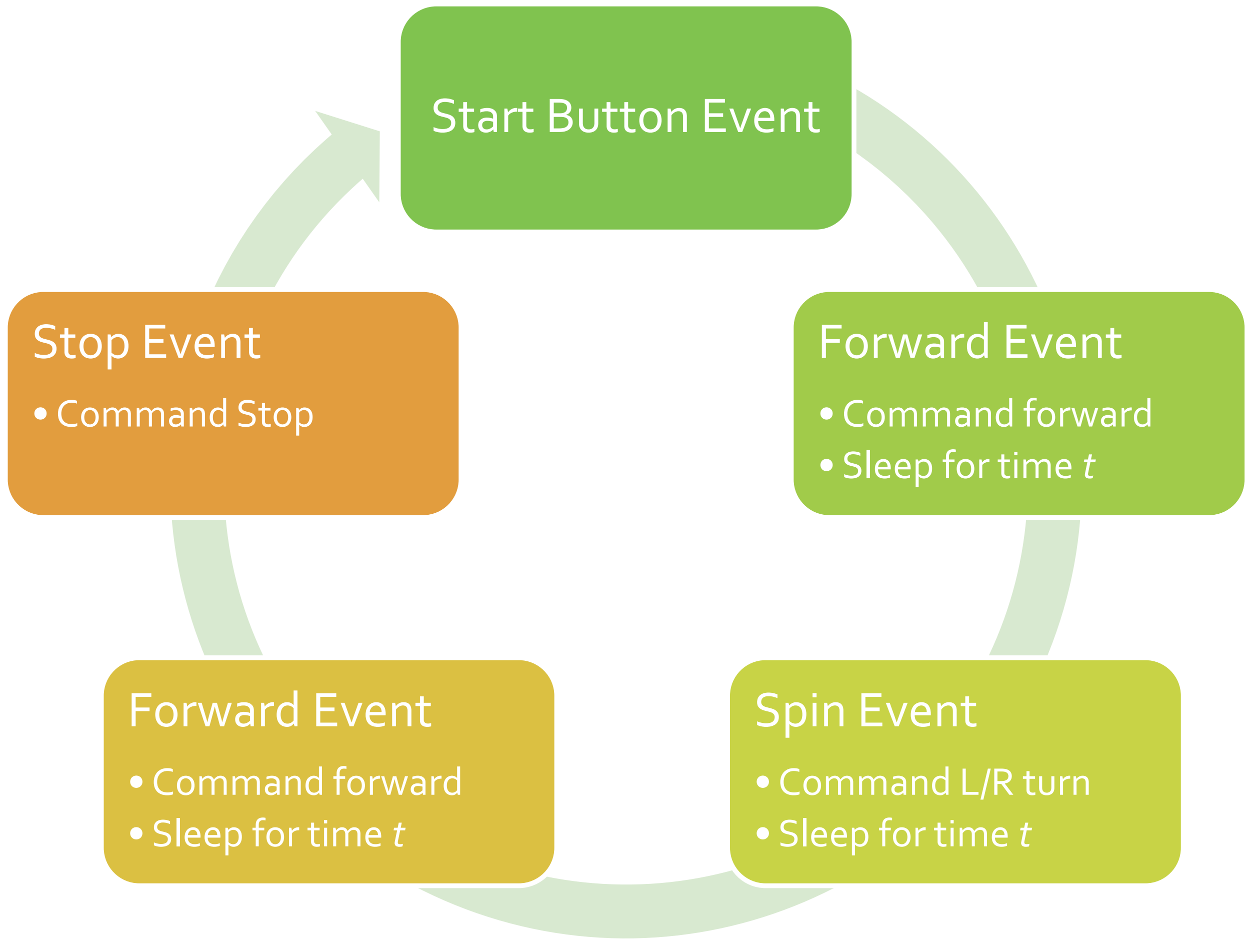
L2Bot Dead Reckoning Control

L2Bot Dead Reckoning Control

- Objective: Drive L2Bot to a new position purely based on direction and time

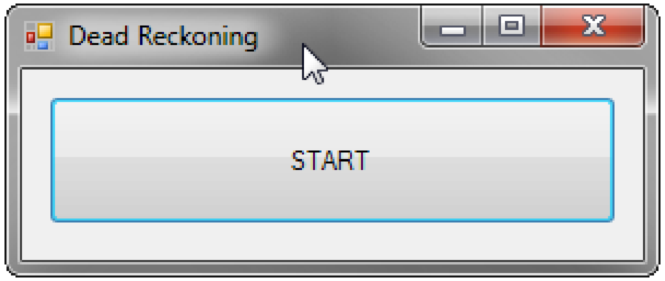


L2Bot Dead Reckoning Control



L2Bot Dead Reckoning Control

- Start by creating a new WinForms project and adding one large button to the form
  - This button will start and stop the program
  - Double click the button to create its event handler

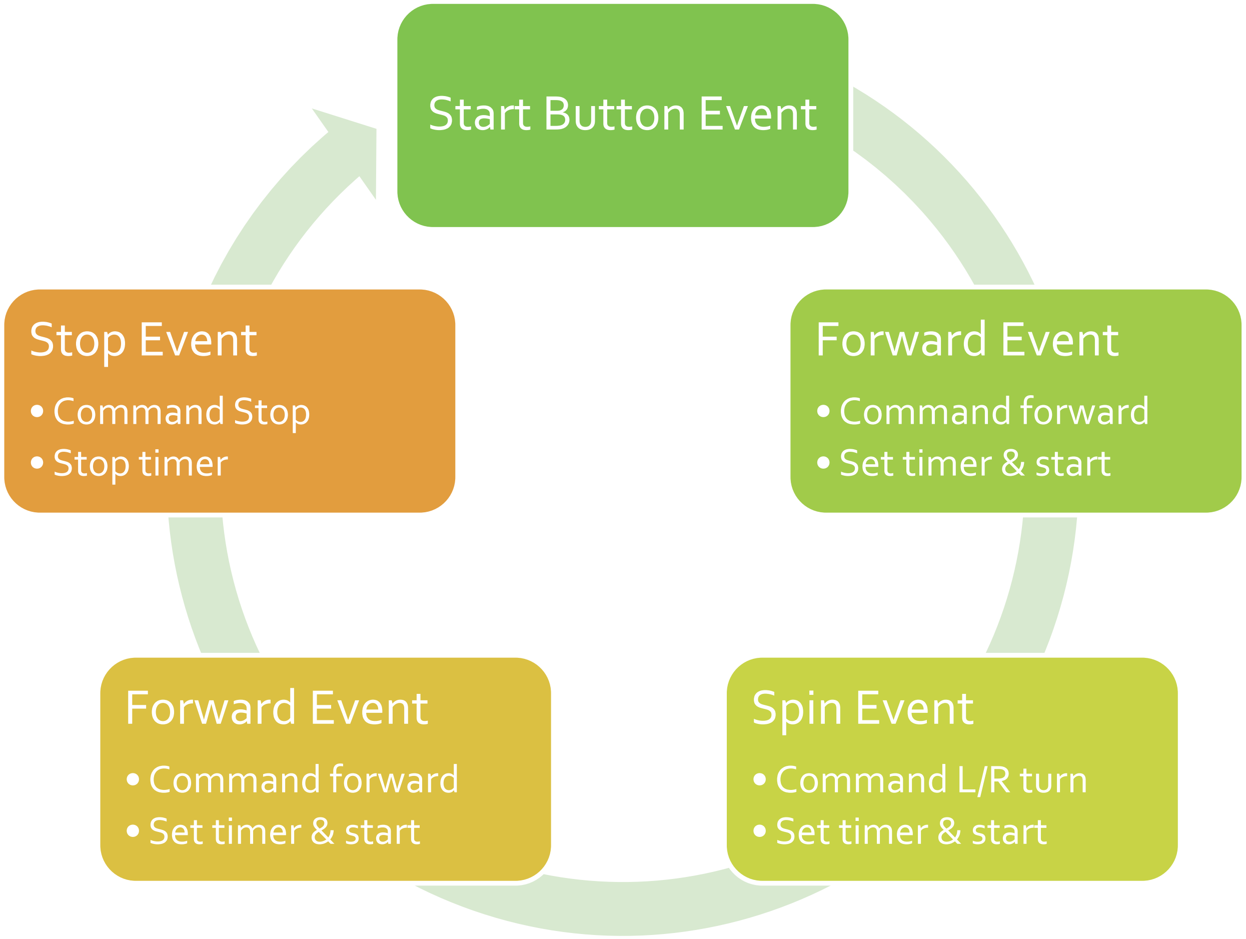


L2Bot Dead Reckoning Control

- In the button click event handler you will add the dead reckoning controls to be sent to the motor controller
- To delay the transmission of motor control commands you can add sleep method calls that will pause your program for a specified number of milliseconds

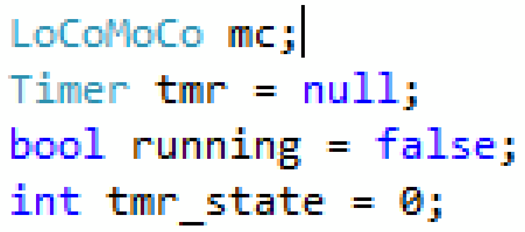
```
private void start_stop_btn_Click(object sender, EventArgs e)
{
        mc.forward();
        Thread.Sleep(5000); // Drive straight for 5 seconds
        mc.turnleft();
        Thread.Sleep(3000); // Turn left for 3 seconds
        mc.forward();
        Thread.Sleep(5000);  // Drive straight for 5 seconds
        mc.stop();
}
```

L2Bot Dead Reckoning Control Using Timers



L2Bot Dead Reckoning Control Using Timers

- As with the simple dead reckoning program, you must first add the LoCoMoCo class to your project
- Then add the following class variables

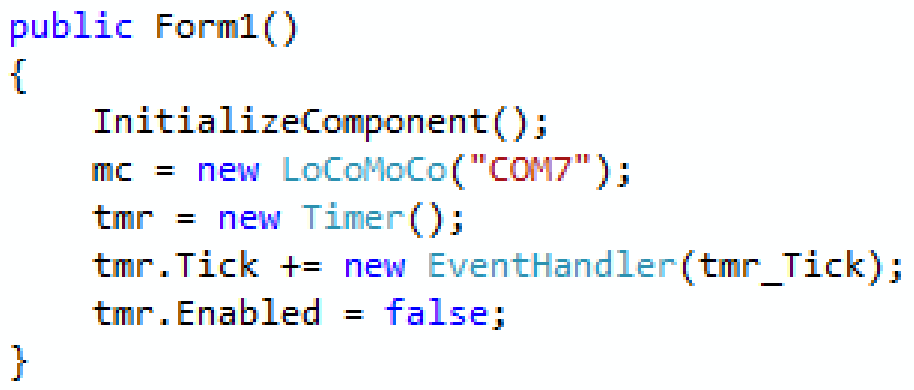


```
LoCoMoCo mc;
Timer tmr = null;
bool running = false;
int tmr_state = 0;
```

- The Timer and status variables are needed to control the time between motor controller commands

L2Bot Dead Reckoning Control Using Timers

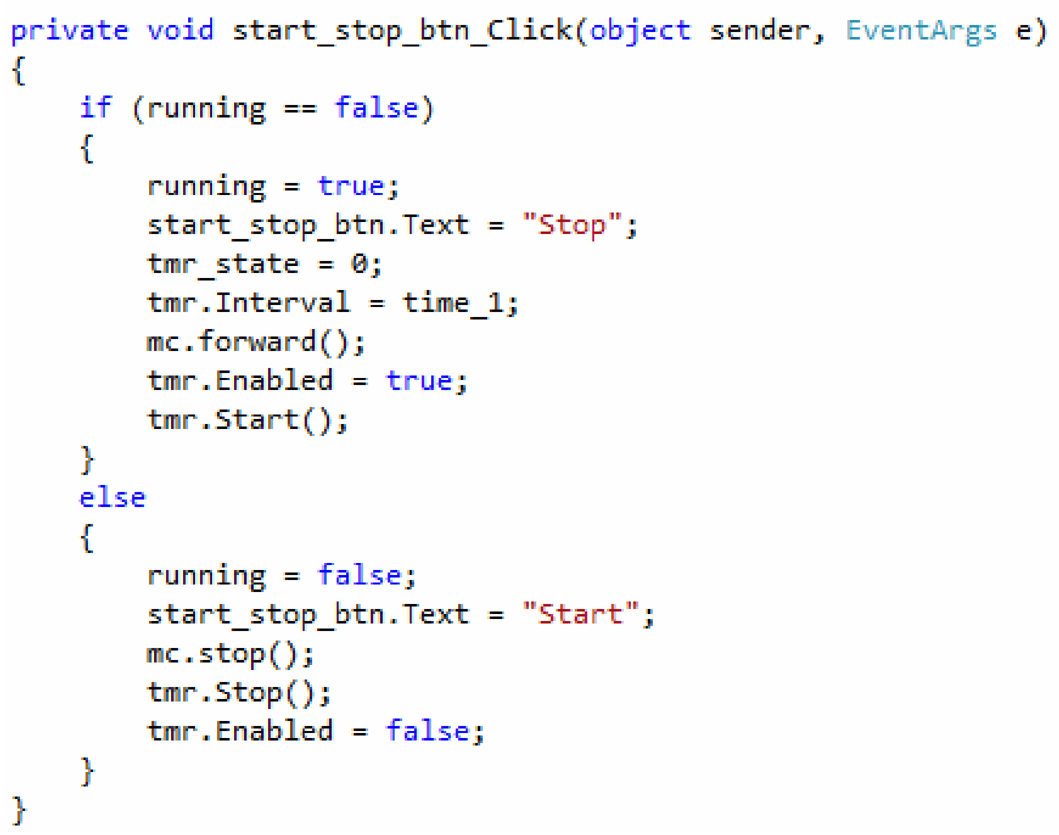
- Create a new instance of the LoCoMoCo and Timer class in the Form1 method
- Then create an event handler for the Timer instance and disable the timer until it is used



```
public Form1()
{
    InitializeComponent();
    mc = new LoCoMoCo("COM7");
    tmr = new Timer();
    tmr.Tick += new EventHandler(tmr_Tick);
    tmr.Enabled = false;
}
```

L2Bot Dead Reckoning Control Using Timers

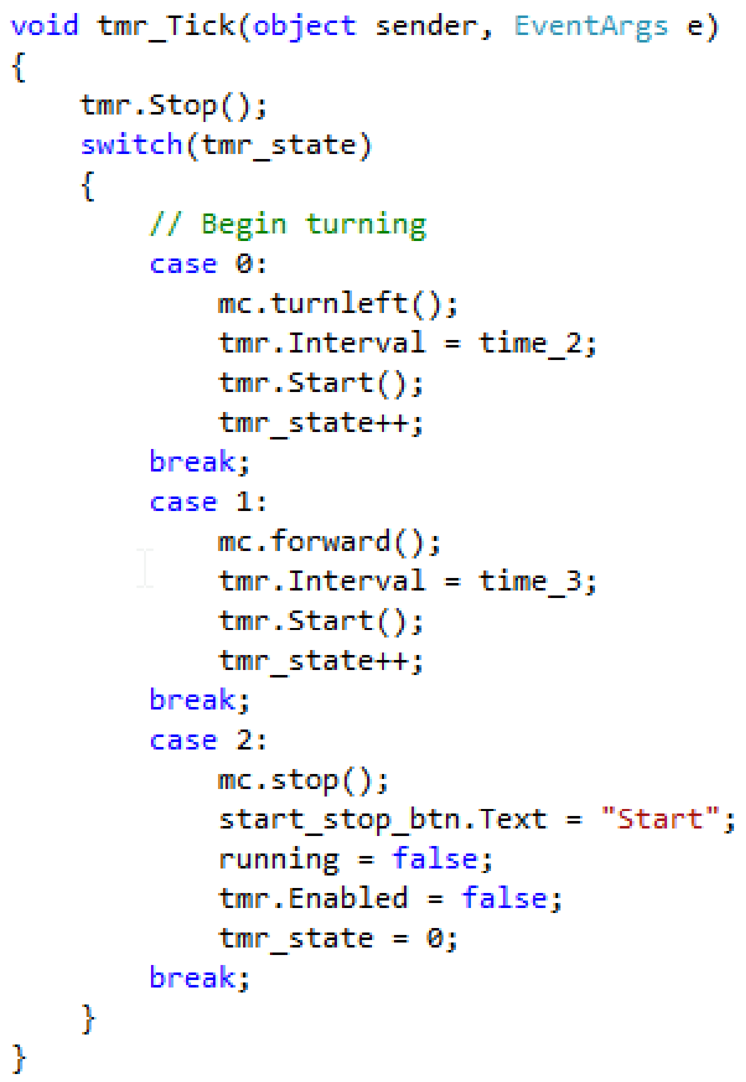
- In the button click handler, we will add the following code.
- If *running* is *false*, begin the dead reckoning control
- If *running* is *true*, abort the current dead reckoning control



```
private void start_stop_btn_Click(object sender, EventArgs e)
{
    if (running == false)
    {
        running = true;
        start_stop_btn.Text = "Stop";
        tmr_state = 0;
        tmr.Interval = time_1;
        mc.forward();
        tmr.Enabled = true;
        tmr.Start();
    }
    else
    {
        running = false;
        start_stop_btn.Text = "Start";
        mc.stop();
        tmr.Stop();
        tmr.Enabled = false;
    }
}
```

L2Bot Dead Reckoning Control Using Timers

- In the Timer tick event handler we will add the following code
- The switch statement will handle the control sequence discussed in the event flow chart



```
void tmr_Tick(object sender, EventArgs e)
{
    tmr.Stop();
    switch(tmr_state)
    {
        // Begin turning
        case 0:
            mc.turnleft();
            tmr.Interval = time_2;
            tmr.Start();
            tmr_state++;
        break;
        case 1:
            mc.forward();
            tmr.Interval = time_3;
            tmr.Start();
            tmr_state++;
        break;
        case 2:
            mc.stop();
            start_stop_btn.Text = "Start";
            running = false;
            tmr.Enabled = false;
            tmr_state = 0;
        break;
    }
}
```

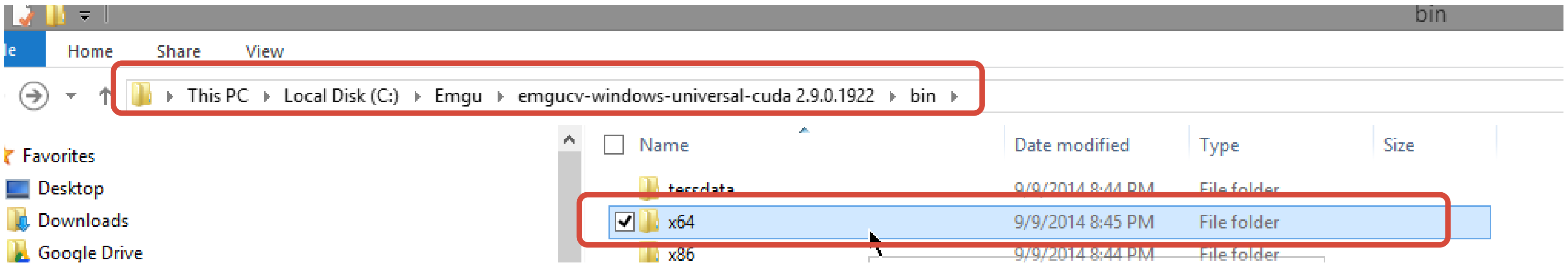
Emgu Setup

Downloading

- Download Emgu Version 2.9.0 beta from the following link
  - http://sourceforge.net/projects/emgucv/files/emgucv/2.4.9-beta/
  - Choose the correct version dependent on the operating system you will be using
  - Once downloaded, run the install .exe

Emgu CV Path Reference

- Once Emgu CV is installed find the installation path to Emgu Version/bin/x64 folder



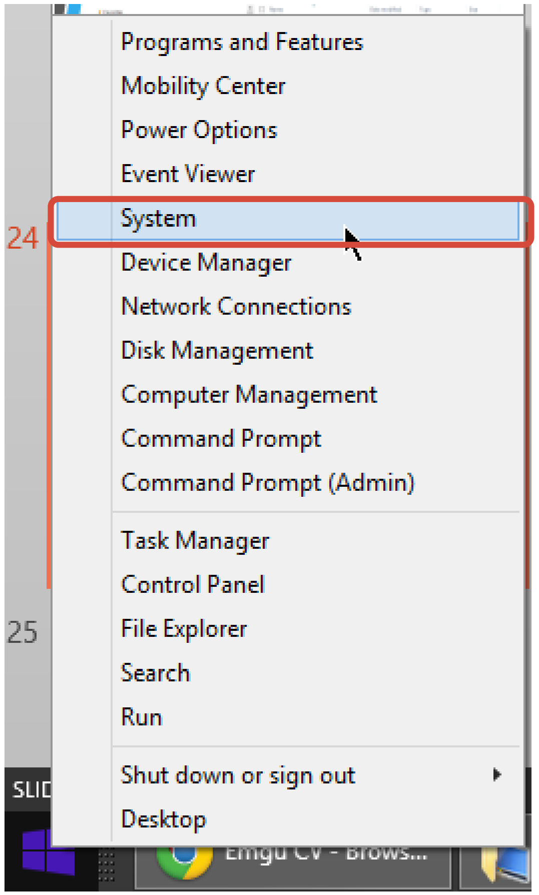
This PC ▸ Local Disk (C:) ▸ Emgu ▸ emgucv-windows-universal-cuda 2.9.0.1922 ▸ bin ▸

Name	Date modified	Type	Size
tessdata	9/9/2014 8:44 PM	File folder	
x64	9/9/2014 8:45 PM	File folder	
x86	9/9/2014 8:44 PM	File folder	

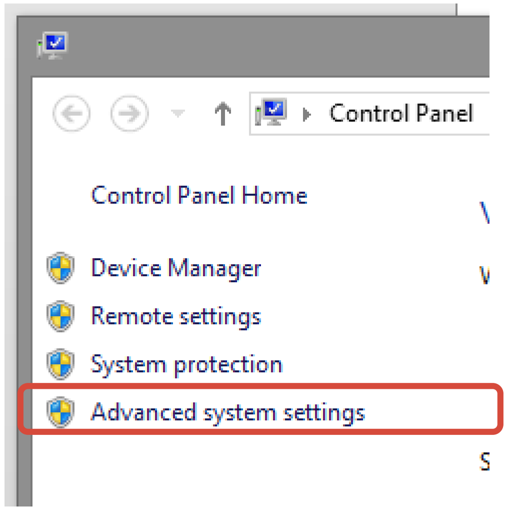
- Copy this path location to the clipboard

Emgu CV Path Reference

- From the Windows Desktop right click on the start menu and select "System"

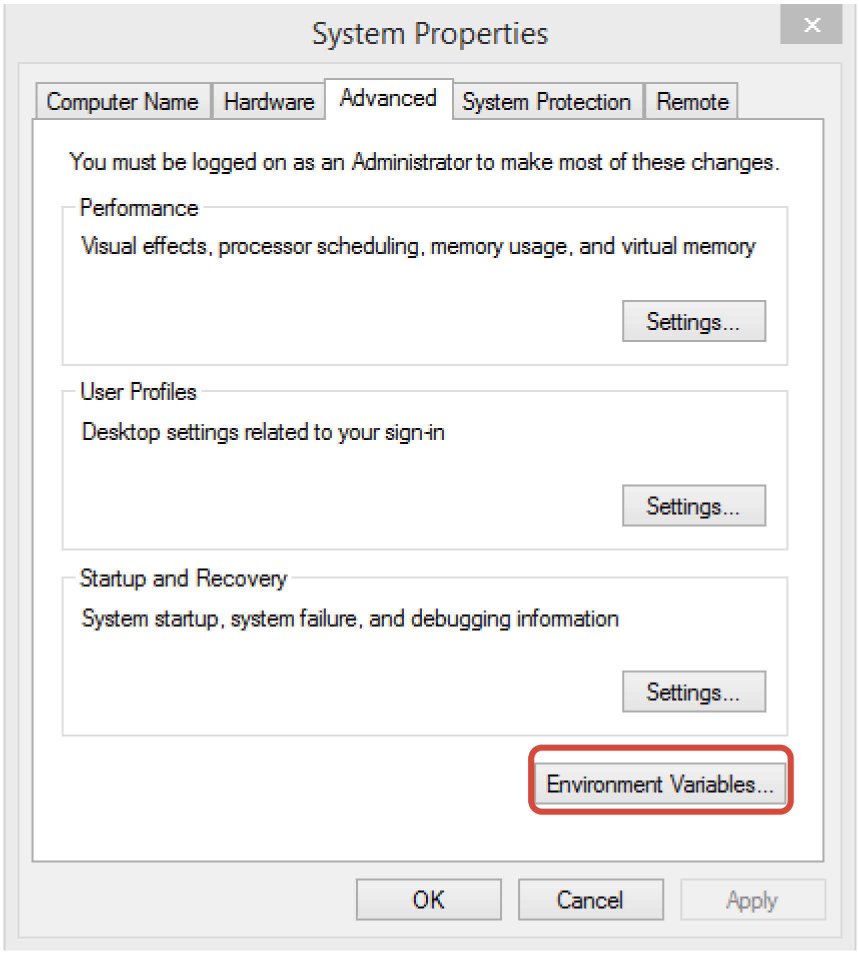


- Then select "Advanced system settings" from the left menu



Emgu CV Path Reference

- Select "Environment Variables" from the System Properties window


System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...

User Profiles
Desktop settings related to your sign-in
Settings...

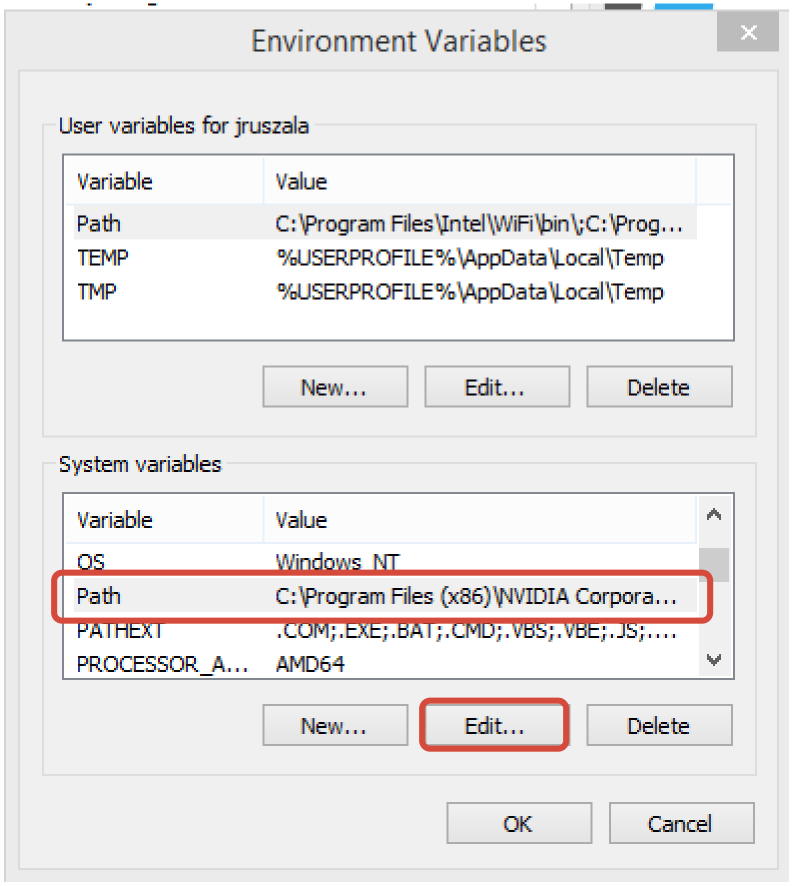
Startup and Recovery
System startup, system failure, and debugging information
Settings...

Environment Variables...

OK | Cancel | Apply

Emgu CV Path Reference

- Select the "Path" list item from the "System variables" group box and click Edit…


Environment Variables

User variables for jruszala

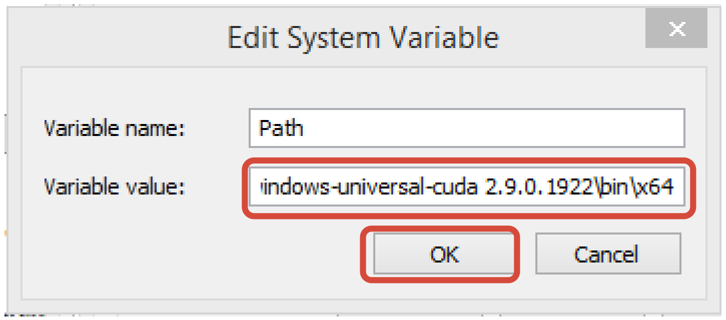
Variable	Value
Path	C:\Program Files\Intel\WiFi\bin\;C:\Prog…
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
OS	Windows_NT
Path	C:\Program Files (x86)\NVIDIA Corpora…
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;….
PROCESSOR_A…	AMD64

Emgu CV Path Reference

- Scroll to the very end of the “Variable value:” text box
- Insert a semicolon and add the emgu/bin/x64 path to the end of the path
- Click OK



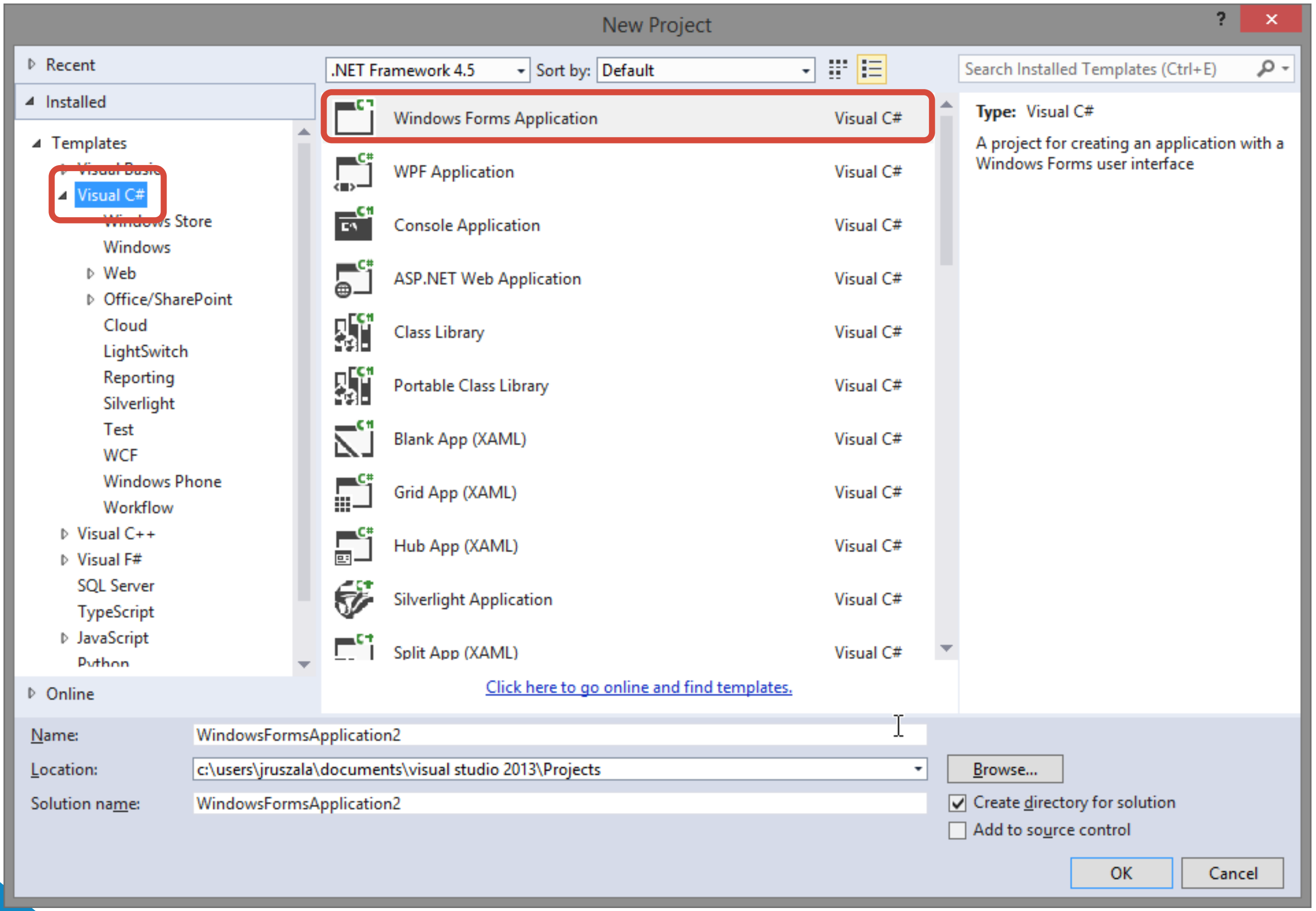
Edit System Variable

Variable name: Path

Variable value: indows-universal-cuda 2.9.0.1922\bin\x64

Emgu Setup in Visual Studio

- Create a new Visual C# Windows Forms Application



New Project

Recent
Installed
Templates
Visual Basic
Visual C#
Windows Store
Windows
Web
Office/SharePoint
Cloud
LightSwitch
Reporting
Silverlight
Test
WCF
Windows Phone
Workflow
Visual C++
Visual F#
SQL Server
TypeScript
JavaScript
Python
Online

.NET Framework 4.5 | Sort by: Default | Search Installed Templates (Ctrl+E)

Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
ASP.NET Web Application	Visual C#
Class Library	Visual C#
Portable Class Library	Visual C#
Blank App (XAML)	Visual C#
Grid App (XAML)	Visual C#
Hub App (XAML)	Visual C#
Silverlight Application	Visual C#
Split App (XAML)	Visual C#

Click here to go online and find templates.

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: WindowsFormsApplication2
Location: c:\users\jruszala\documents\visual studio 2013\Projects
Solution name: WindowsFormsApplication2

Browse...
☑ Create directory for solution
☐ Add to source control

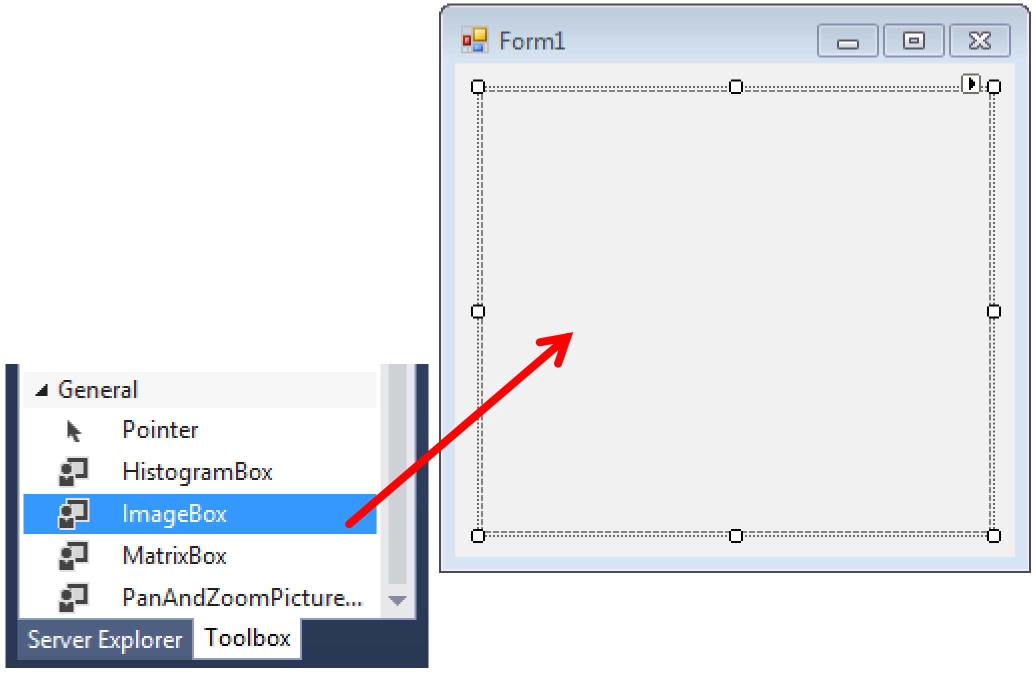
OK | Cancel

Emgu Setup in Visual Studio

- Emgu GUI components
  - Import the OpenCV GUI components by following the procedure outlined here: http://www.emgu.com/wiki/index.php/Add_ImageBox_Control
  - The 'Emgu.CV.UI.dll' file referenced can be found in the 'C:\Emgu\emgucv-windows-universal-cuda 2.9.0.1922\bin' folder

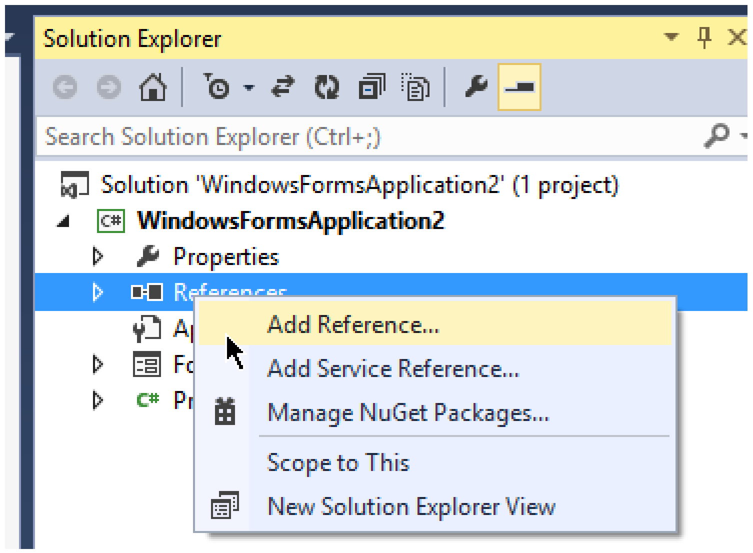
Emgu Setup in Visual Studio

- Once the GUI components have been added, drag and drop an ImageBox instance on to the current Windows Form



Emgu Setup in Visual Studio

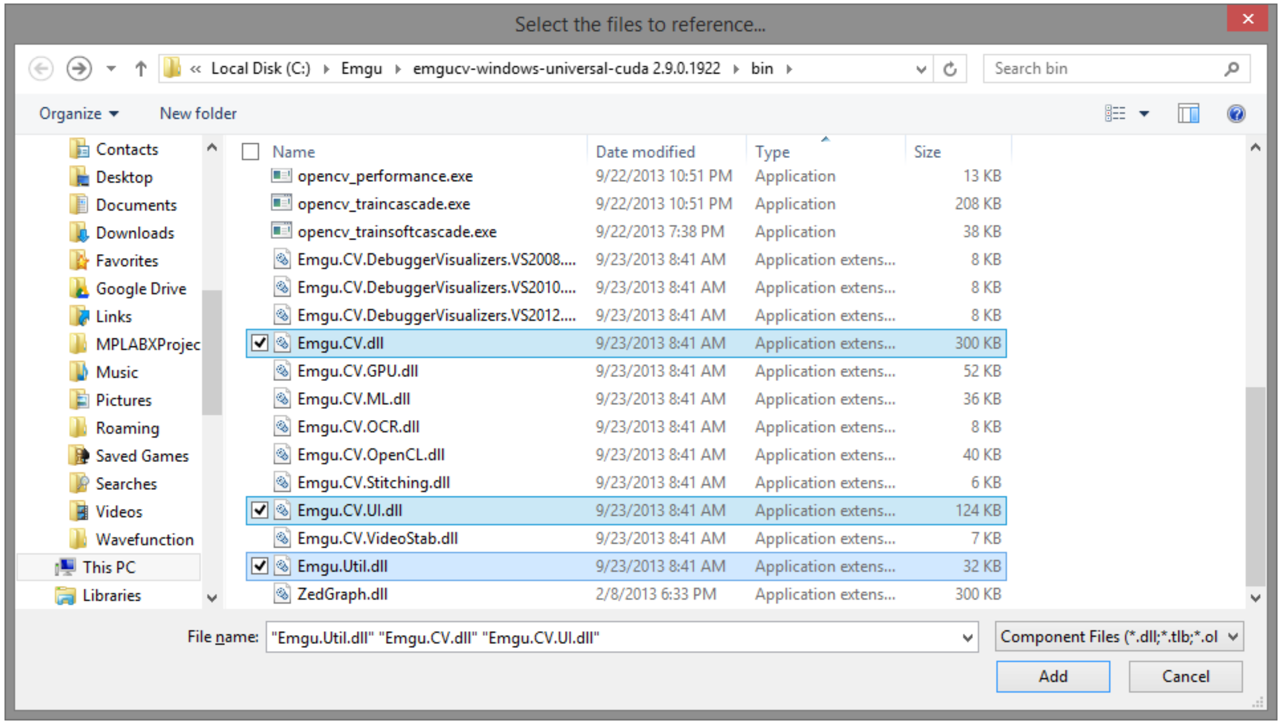
- Reference the following .dll files in the project by right clicking the 'References' folder in the Solution Explorer to the right of the screen and selecting 'Add Reference'



Emgu Setup in Visual Studio

- Using the 'Browse' tab, navigate to C:\Emgu\emgucv-windows-universal-cuda 2.9.0.1922\bin and select the following files:

- Emgu.CV
- Emgu.CV.UI
- Emgu.Util


Select the files to reference...

Local Disk (C:) › Emgu › emgucv-windows-universal-cuda 2.9.0.1922 › bin ›

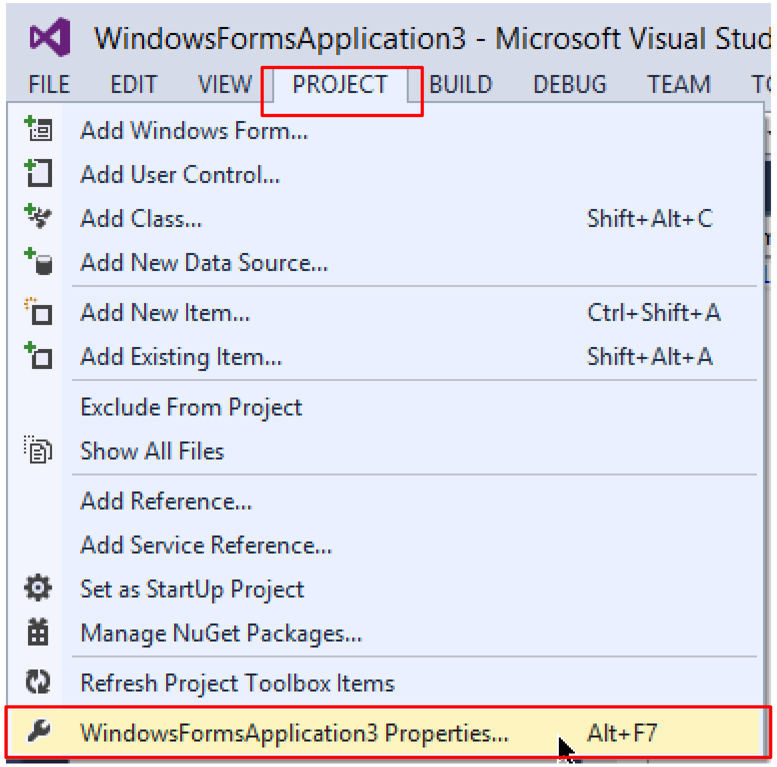
	Name	Date modified	Type	Size
	opencv_performance.exe	9/22/2013 10:51 PM	Application	13 KB
	opencv_traincascade.exe	9/22/2013 10:51 PM	Application	208 KB
	opencv_trainsoftcascade.exe	9/22/2013 7:38 PM	Application	38 KB
	Emgu.CV.DebuggerVisualizers.VS2008....	9/23/2013 8:41 AM	Application extens...	8 KB
	Emgu.CV.DebuggerVisualizers.VS2010....	9/23/2013 8:41 AM	Application extens...	8 KB
	Emgu.CV.DebuggerVisualizers.VS2012....	9/23/2013 8:41 AM	Application extens...	8 KB
✓	Emgu.CV.dll	9/23/2013 8:41 AM	Application extens...	300 KB
	Emgu.CV.GPU.dll	9/23/2013 8:41 AM	Application extens...	52 KB
	Emgu.CV.ML.dll	9/23/2013 8:41 AM	Application extens...	36 KB
	Emgu.CV.OCR.dll	9/23/2013 8:41 AM	Application extens...	8 KB
	Emgu.CV.OpenCL.dll	9/23/2013 8:41 AM	Application extens...	40 KB
	Emgu.CV.Stitching.dll	9/23/2013 8:41 AM	Application extens...	6 KB
✓	Emgu.CV.UI.dll	9/23/2013 8:41 AM	Application extens...	124 KB
	Emgu.CV.VideoStab.dll	9/23/2013 8:41 AM	Application extens...	7 KB
✓	Emgu.Util.dll	9/23/2013 8:41 AM	Application extens...	32 KB
	ZedGraph.dll	2/8/2013 6:33 PM	Application extens...	300 KB

File name: "Emgu.Util.dll" "Emgu.CV.dll" "Emgu.CV.UI.dll"

Component Files (*.dll;*.tlb;*.ol

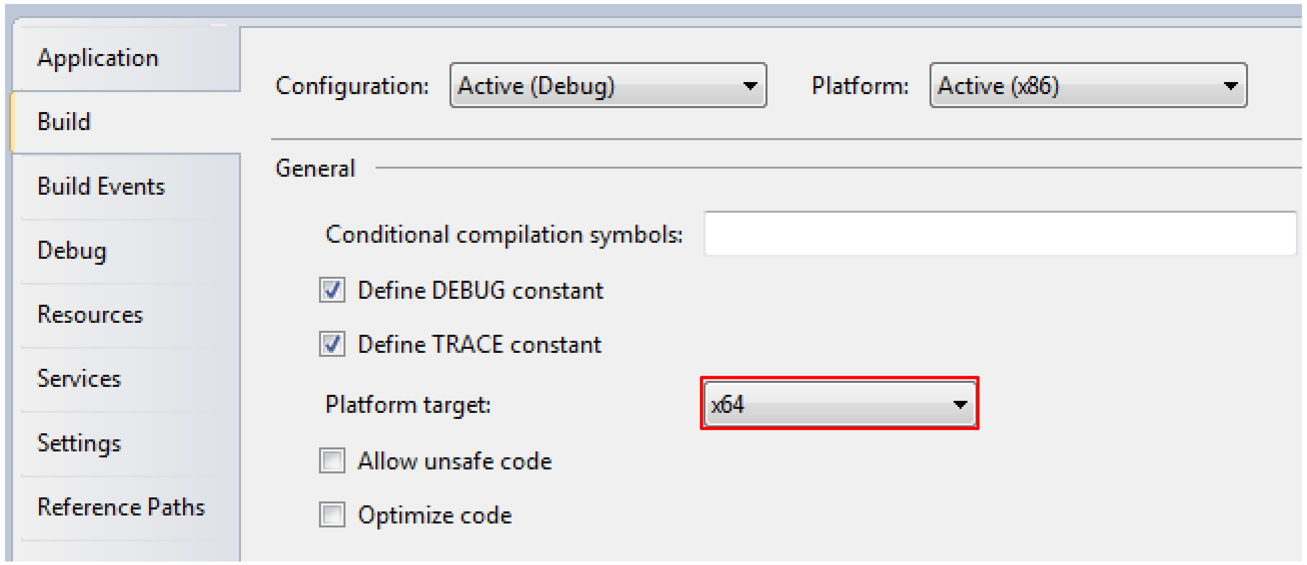
Emgu Setup in Visual Studio

- The final step in the setup is to change the 'Platform target' to x64 if you are on a 64 bit OS
  - From the 'Project' menu select *ProjectName* Properties…



Emgu Setup in Visual Studio

- Select x64 for the 'Platform target' in the Build tab



Configuration: Active (Debug)
Platform: Active (x86)

Conditional compilation symbols:
☑ Define DEBUG constant
☑ Define TRACE constant
Platform target: x64
☐ Allow unsafe code
☐ Optimize code

Emgu Basics

- Once the Emgu references have been setup in Visual Studio, you can begin writing your C# code
- First, include the referenced Emgu files in your Forms .cs file
  - using Emgu.CV;
  - using Emgu.Util;
  - using Emgu.CV.Structure;

Emgu Basics

- Next, you will need to get an instance of the Capture class that allow you to capture images from your input device.
  - Capture _capture = null;
- Add the Capture instance as a Class variable since we will use it in several of the Class methods

Emgu Basics

- In the form's Load method you will have to:
  - Create a new instance of the Capture method
  - Create an event handler to receive image frames from the input device
  - Start the Capture instance

```
private void Form1_Load(object sender, EventArgs e)
{
     _capture = new Capture();
     _capture.ImageGrabbed += capture_ImageGrabbed;
     capture.Start();

}
```

Emgu Basics

- In the Capture event handler you need to:
  - Retrieve the captured image frame
  - Resize it to fit your imageBox element
  - Display the image in the imageBox

```
void _capture_ImageGrabbed(object sender, EventArgs e)
{
    Image<Bgr, Byte> frame =
    _capture.RetrieveBgrFrame().Resize(imageBox1.Width,
    imageBox1.Height,Emgu.CV.CvEnum.INTER.CV_INTER_LINEAR);
    imageBox1.Image = frame;
}
```

Emgu Basics

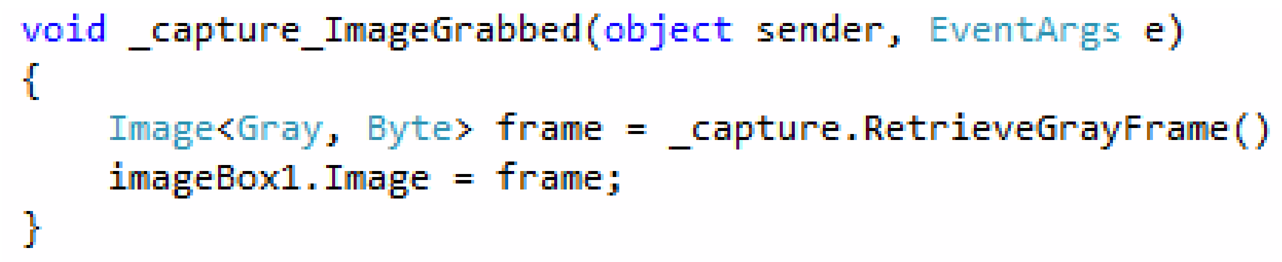
- As a last step, you need to override the onClosing method to stop the Capture instance. This will prevent an exception from being thrown when the program is closed.

```
protected override void OnClosing(CancelEventArgs e)
{
    _capture.Stop();
    base.OnClosing(e);
}
```

Color Conversion

Color Conversion

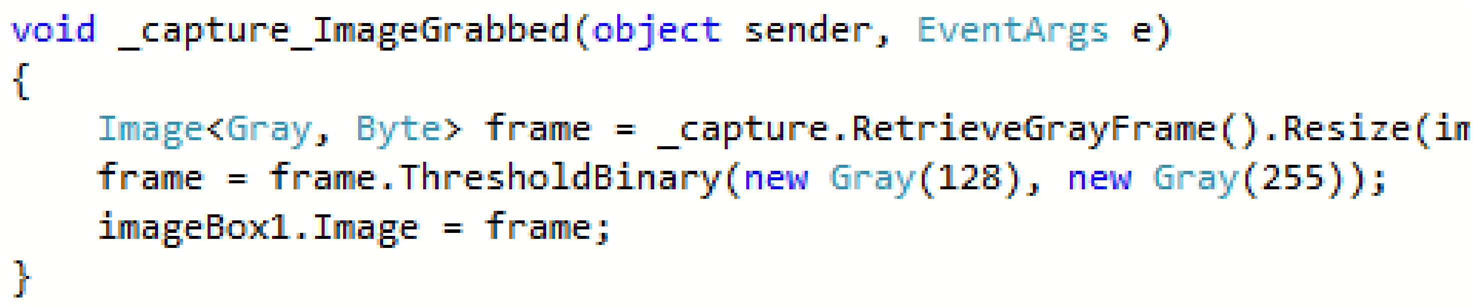
- Grayscale
  - In addition to retrieving a Bgr image in the ImageGrabbed event handler, you can also retrieve a grayscale image
  - To do this we change the Image type from Bgr to Gray and replace the RetreiveBgrFrame method call with RetreiveGrayFrame



```
void _capture_ImageGrabbed(object sender, EventArgs e)
{
    Image<Gray, Byte> frame = _capture.RetrieveGrayFrame()
    imageBox1.Image = frame;
}
```

Color Conversion

- Binary – Pure Black and White
  - To produce a pure black and white image, you first start with a Gray image and then call the ThresholdBinary method. The method accepts the min and max threshold values for the binary conversion

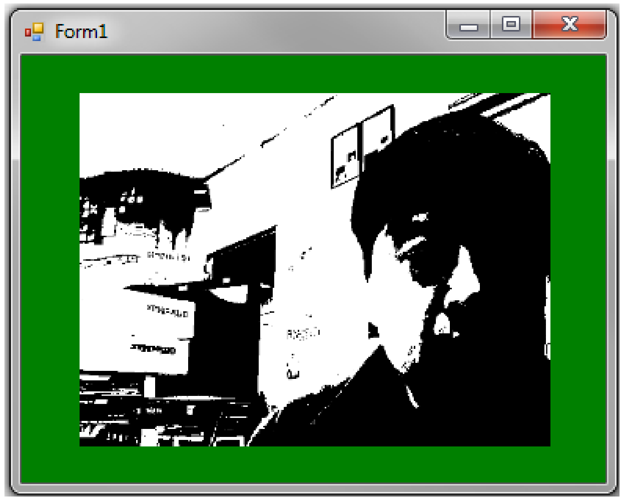


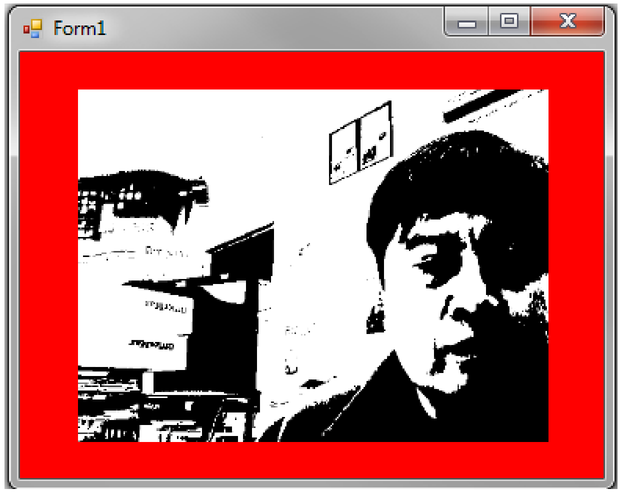
```
void _capture_ImageGrabbed(object sender, EventArgs e)
{
    Image<Gray, Byte> frame = _capture.RetrieveGrayFrame().Resize(im
    frame = frame.ThresholdBinary(new Gray(128), new Gray(255));
    imageBox1.Image = frame;
}
```

Pixel Counter

Pixel Counter

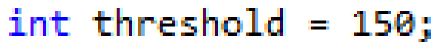
- Objective: Create a Windows Forms application using Emgu that counts the number of white pixels in the image.
  - If > 50% set background RED
  - If <= 50% set background GREEN





Pixel Counter

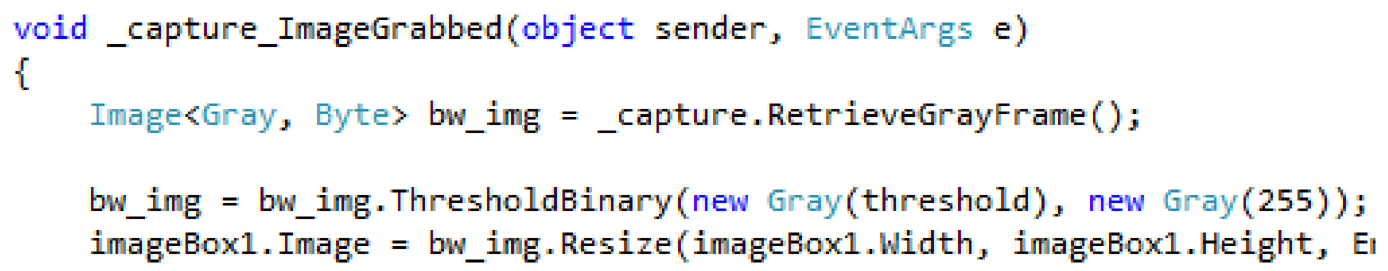
- Create a new Win Forms application and repeat the steps outlined in the Emgu Basics section.
  - Create one new class variable that will be used for the lower threshold in your binary image conversion method call



```
int threshold = 150;
```

Pixel Counter

- In the ImageGrabbed method, retrieve a Gray frame and convert it to a binary image using the threshold variable you just created as the lower bound
- Display that image to the screen



```
void _capture_ImageGrabbed(object sender, EventArgs e)
{
    Image<Gray, Byte> bw_img = _capture.RetrieveGrayFrame();

    bw_img = bw_img.ThresholdBinary(new Gray(threshold), new Gray(255));
    imageBox1.Image = bw_img.Resize(imageBox1.Width, imageBox1.Height, E
```

Pixel Counter

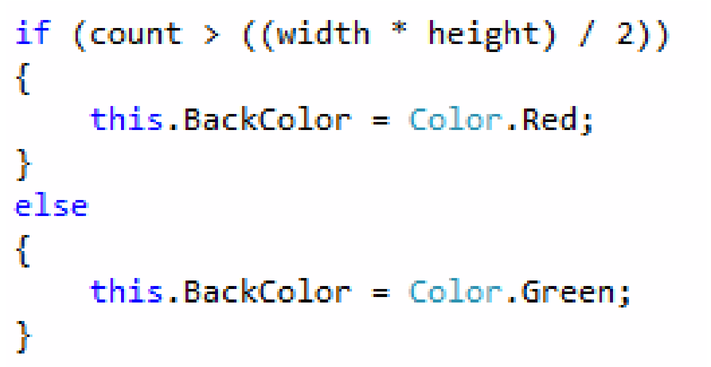
- Iterate through the rows and columns
  - Count the number of white pixels

```
int width = bw_img.Width;
int height = bw_img.Height;
int count = 0;

for (int h = 0; h < height; h++)
{
    for (int w = 0; w < width; w++)
    {
        if (bw_img.Data[h, w, 0] == 255)
        {
            count++;
        }
    }
}
```

Pixel Counter

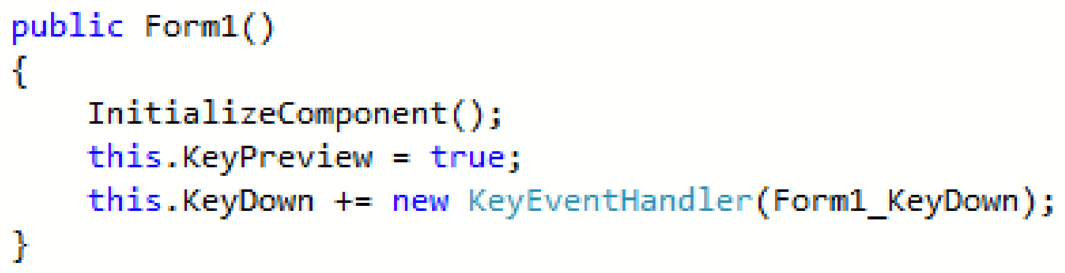
- Set the background color of the form based on the total number of white pixels counted



```
if (count > ((width * height) / 2))
{
    this.BackColor = Color.Red;
}
else
{
    this.BackColor = Color.Green;
}
```

Pix Count – Interactive Threshold

- The binary threshold variable can be made interactive by adding a simple key event handler
  - In the Form1 method
    - Set the KeyPreview attribute to true
    - Create a new KeyEventHandler



```
public Form1()
{
    InitializeComponent();
    this.KeyPreview = true;
    this.KeyDown += new KeyEventHandler(Form1_KeyDown);
}
```

Pix Count – Interactive Threshold

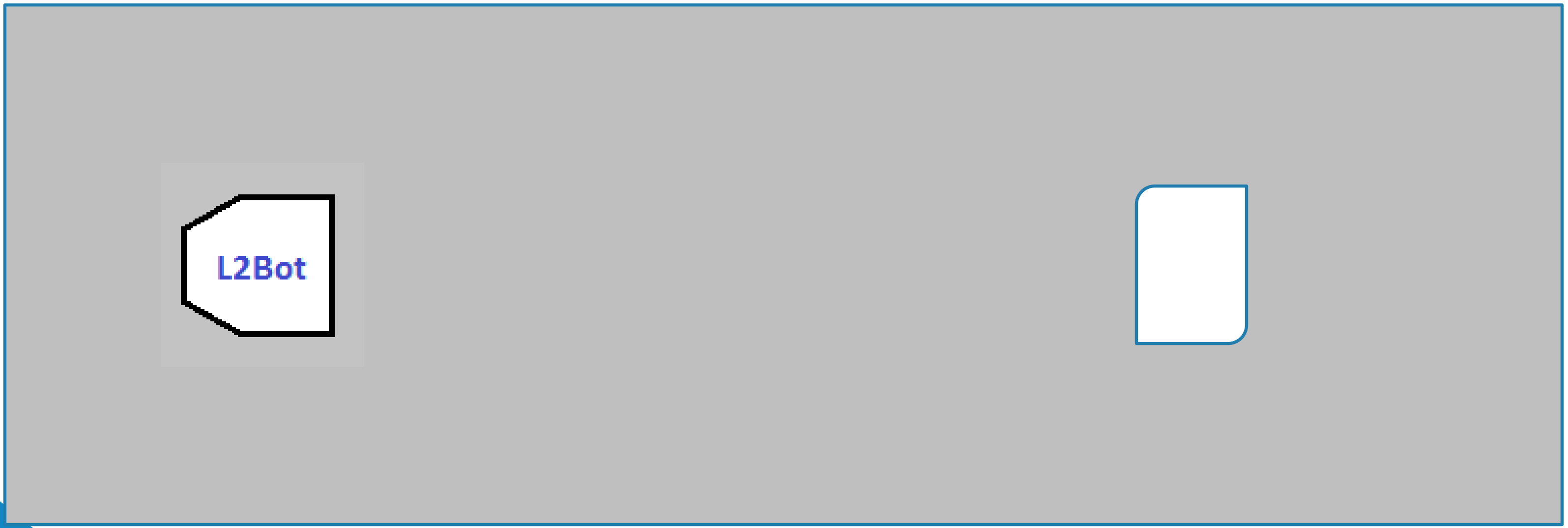
- In the KeyDown event handler method
  - Determine if the event was caused by the up or down key
  - Increase or decrease the value of the minimum binary threshold

```
void Form1_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Up)
    {
        threshold += 5;
        if (threshold >= 255)
            threshold = 255;
    }
    else if (e.KeyCode == Keys.Down)
    {
        threshold -= 5;
        if (threshold <= 0)
            threshold = 0;
    }
}
```

L2Bot Stop at White

Stop at White

- Objective: Create a Windows Forms application using Emgu that drives the L2Bot forward until it encounters a white sheet of paper
  - Hint: Start with the OpenCV_Pixel_Counter program

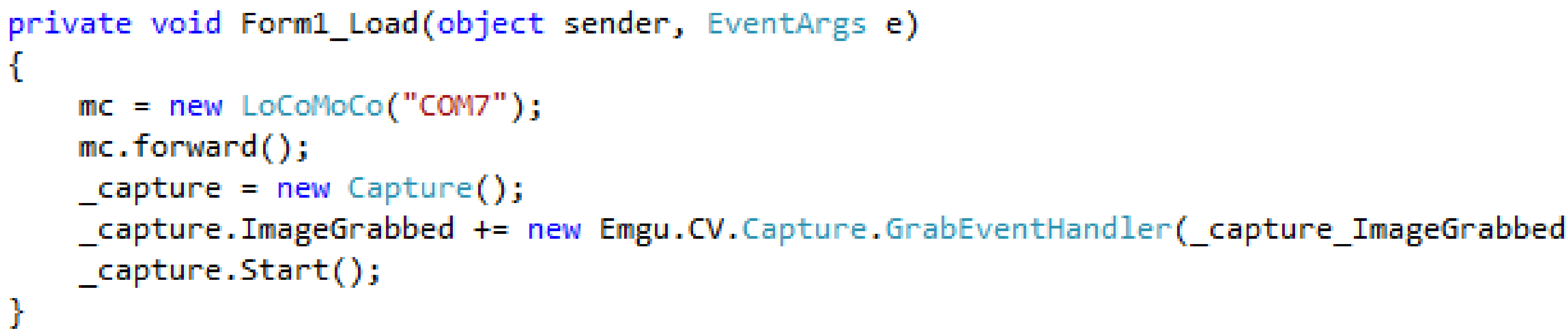


Stop at White

- Add the LoCoMoCo class to the OpenCV_Pixel_Counter program
- After counting the pixels, determine if they are greater than some threshold
- If greater, send a stop command to the motor controller

Stop at White

- Create a new instance of the LoCoMoCo class in the From1_Load method above the creation of your Emgu initialization code
- Command the robot to drive forward

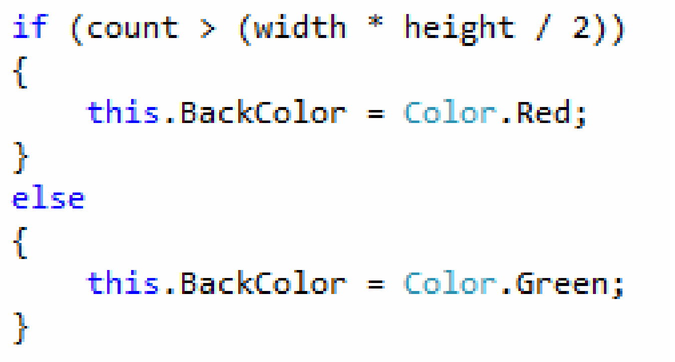


```
private void Form1_Load(object sender, EventArgs e)
{
    mc = new LoCoMoCo("COM7");
    mc.forward();
    _capture = new Capture();
    _capture.ImageGrabbed += new Emgu.CV.Capture.GrabEventHandler(_capture_ImageGrabbed
    _capture.Start();
}
```

Stop at White

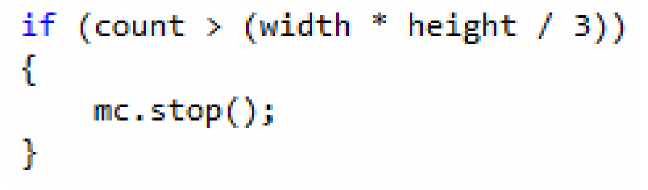
- After evaluating the number of white pixels within the ImageGrabbed event handler, you will determine if the robot should stop or continue

Pixel_Counter



```
if (count > (width * height / 2))
{
    this.BackColor = Color.Red;
}
else
{
    this.BackColor = Green;
}
```

Stop_At_White



```
if (count > (width * height / 3))
{
    mc.stop();
}
```

L2Bot Line Following

L2Bot Line Following

- Objective: Create a Win Forms application that follows a white line using Emgu and the L2Bot

L2Bot Line Following

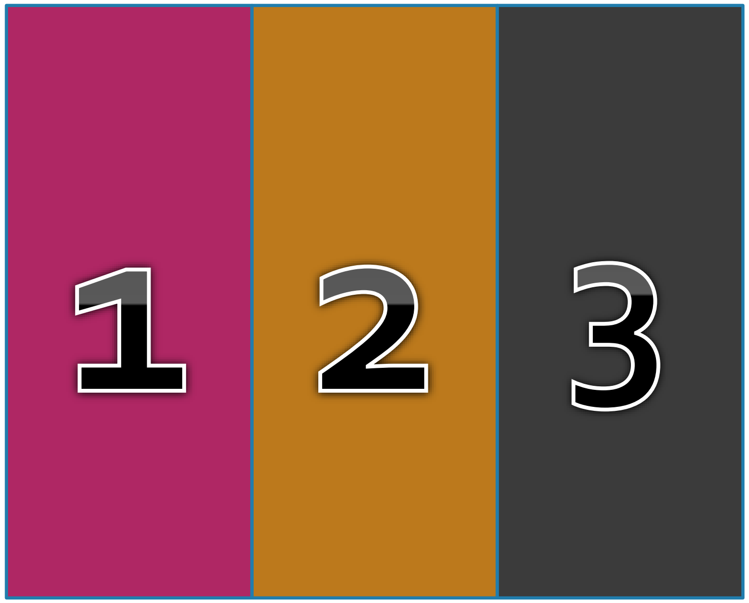
- Combining the Emgu image capture/manipulation applications and L2Bot control programs with some additional logic can produce a line following robot
- To start with, create an Emgu application (following the same steps you used in the past) and then add the LoCoMoCo class to the project. Alternatively, you may wish to start with the OpenCV_Pixel_Counter program as a base.

L2Bot Line Following

- To follow any line the computer must be able to:
  - Distinguish the line from the surroundings
  - Determine where the line is in relation to the robot
- Distinguishing the line from the surroundings
  - Since the line the robot will follow is white and the floor color is dark, the line can easily be distinguished by converting the image to a pure black and white image
    - image.ThresholdBinary(loThresh, hiThresh);

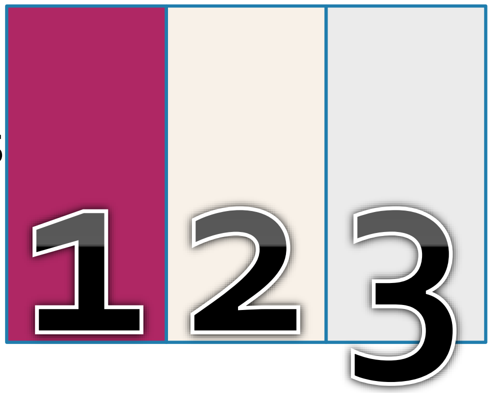
L2Bot Line Following

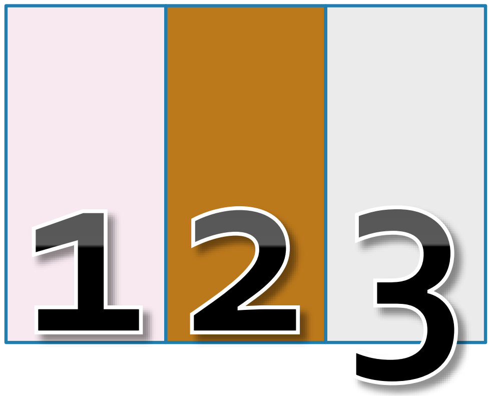
- Determining line location
  - There are many ways to determine where the line is in relation to the robot. Some are very simple and some require some complex strategies and math
  - The simplest method involves dividing the image up into three columns and then counting the number of white(line) pixels in each column



L2Bot Line Following

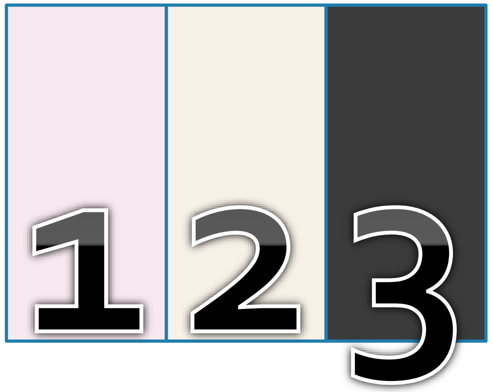
- If the number of white pixels in column 1 is greater than the number of white pixels in columns 2 & 3 – Turn LEFT





- If the number of white pixels in columns 2 is greater than the number of white pixels in columns 1 & 3 – Go FORWARD

- If the number of white pixels in column 3 is greater than the number of white pixels in columns 1 & 2 – Turn RIGHT



L2Bot Line Following

- Once you have a Win Forms application that contains the essential files for both the motor controller and Emgu you can begin to add the line following logic
- From within the ImageGrabbed event handler:
  - Retrieve a gray frame
  - Convert the gray frame to a binary image
  - Get the width and height of the captured image
  - Create variables to hold the amount of white pixels in each column
  - Iterate through all the image pixels to find white pixels and increment the corresponding column variable
  - Command the L2Bot based on the column with the greatest number of white pixels

L2Bot Line Following

- Create variables to hold the amount of white pixels in each column
- Iterate through all the image pixels to find white pixels and increment the corresponding column variable

```
int left_col = 0;
int middle_col = 0;
int right_col = 0;
int col_width = width / 3;

for (int h = 0; h < height; h++)
{
    for (int w = 0; w < width; w++)
    {
        if (bw_img.Data[h, w, 0] == 255)
        {
            if (w < col_width)
            {
                left_col++;
            }
            else if (w < (col_width * 2))
            {
                middle_col++;
            }
            else
            {
                right_col++;
            }
        }
    }
}
```

L2Bot Line Following

- Command the L2Bot based on the column with the greatest number of white pixels

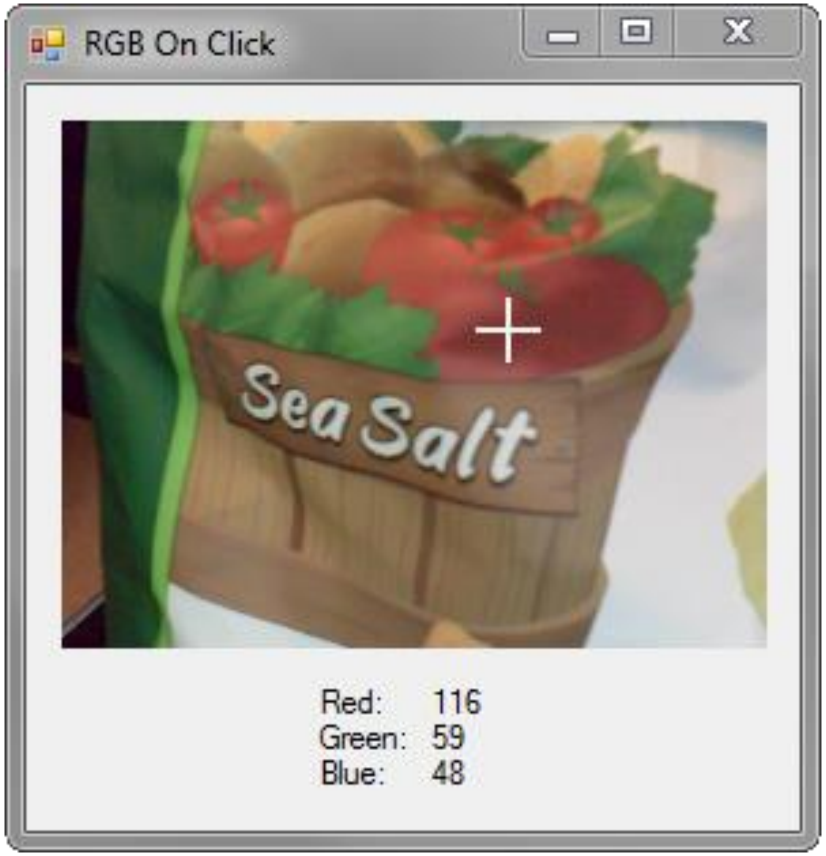
```
int current_motor_command = 0;
// Send motor command
if ((left_col > middle_col) && (left_col > right_col))
{
    current_motor_command = 1;
}
else if ((right_col > middle_col) && (right_col > left_col))
{
    current_motor_command = 2;
}
else
{
    current_motor_command = 3;
}

if (current_motor_command != previous_motor_command)
{
    previous_motor_command = current_motor_command;
    switch (current_motor_command)
    {
        case 1:
            mc.turnleft();
            break;
        case 2:
            mc.turnright();
            break;
        case 3:
            mc.forward();
            break;
    }
}
```

Obtaining BGR Values From An Image

BGR Values

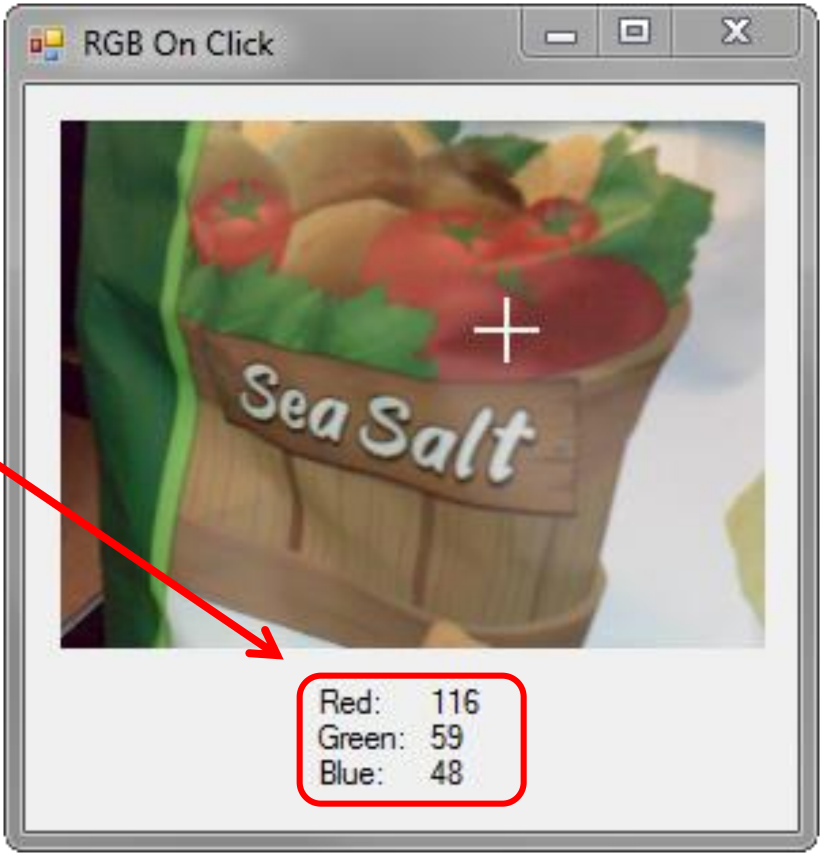
- Objective: Get the BGR (Blue, Green, Red) values of a pixel when selected using the mouse



```
Red:    116
Green:  59
Blue:   48
```

BGR Values

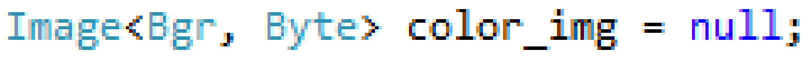
- Begin by adding 6 labels to the form that contains your Emgu ImageBox
- Change the text attribute of the three labels in the first column to display the Red, Green, and Blue Labels
- Change the label name attribute of the 3 labels in the second column to reflect the containers the color values will be displayed in



```
Red:    116
Green:  59
Blue:   48
```

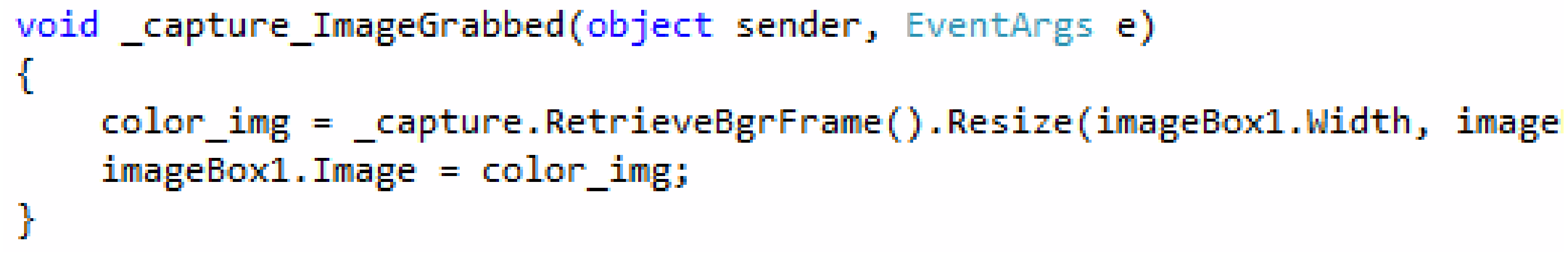
BGR Values

- Create a class Image variable



```
Image<Bgr, Byte> color_img = null;
```

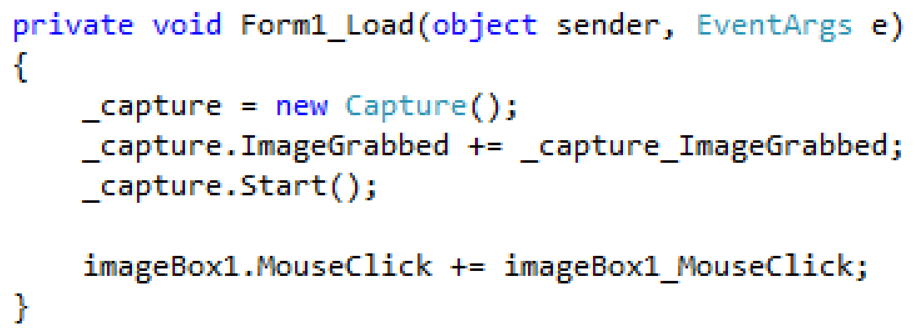
- In the ImageGrabbed event handler, save the grabbed image to the class Image variable and display the image to the screen



```
void _capture_ImageGrabbed(object sender, EventArgs e)
{
    color_img = _capture.RetrieveBgrFrame().Resize(imageBox1.Width, image
    imageBox1.Image = color_img;
}
```

BGR Values

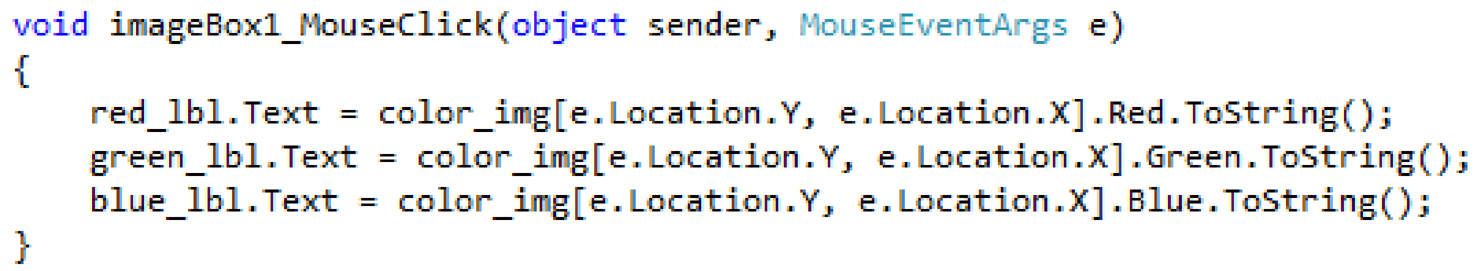
- After your standard Emgu initialization code, add a mouse click listener for the ImageBox



```
private void Form1_Load(object sender, EventArgs e)
{
    _capture = new Capture();
    _capture.ImageGrabbed += _capture_ImageGrabbed;
    _capture.Start();

    imageBox1.MouseClick += imageBox1_MouseClick;
}
```

- In the MouseClick event handler, retreive the color from the selected pixel and display the value to the screen

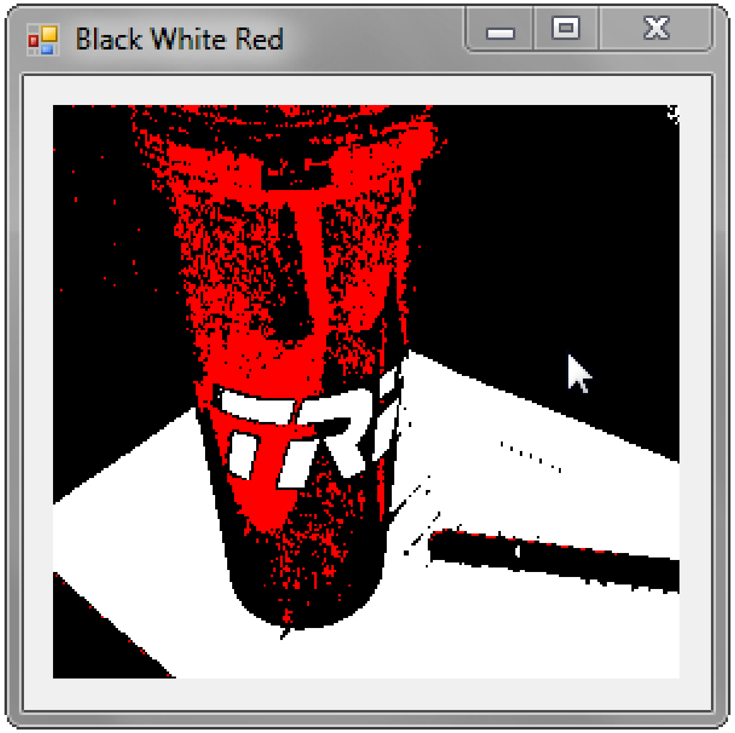


```
void imageBox1_MouseClick(object sender, MouseEventArgs e)
{
    red_lbl.Text = color_img[e.Location.Y, e.Location.X].Red.ToString();
    green_lbl.Text = color_img[e.Location.Y, e.Location.X].Green.ToString();
    blue_lbl.Text = color_img[e.Location.Y, e.Location.X].Blue.ToString();
}
```

Creating a Black, White, and Red Image

Black, White, Red

- Objective: Display a black and white image but maintain any red pixels



Black, White, Red

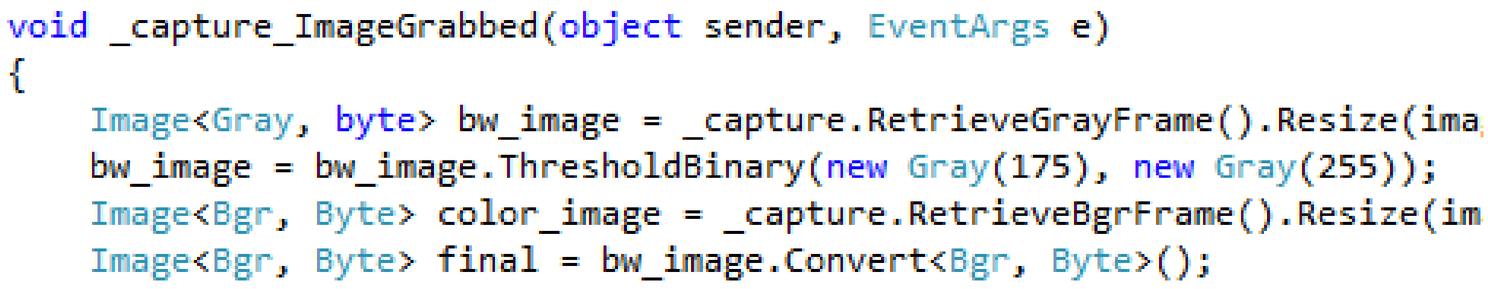
- You can easily create the image by following these steps:
  - Retrieve a gray frame
  - Convert the gray frame to a binary image
  - Retrieve a color frame
  - Create a new color image that will be used to create the black, white, and red image
  - Iterate through the color image pixels, if the pixel is red set the final images pixel color to red. Else, set the pixel to the corresponding binary image pixel value

Black, White, Red

- There are two ways to get/set pixel values in Emgu
  - Image[Row, Column].Blue
  - Image[Row, Column].Green
  - Image[Row, Column].Red
    - Pros – Easier to follow/use
    - Cons – Slow to execute
  - Image.Data[Row, Column, 0] => Blue Component
  - Image.Data[Row, Column, 1] => Green Component
  - Image.Data[Row, Column, 2] => Red Component
    - Pros – Faster
    - Cons – Harder to follow

Black, White, Red

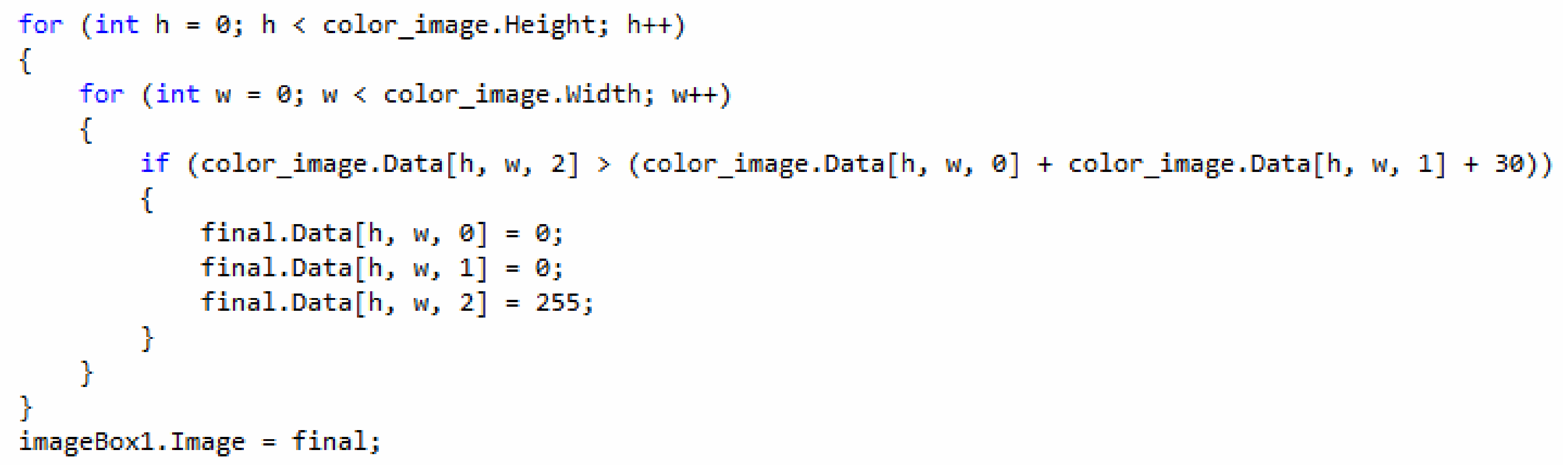
- Retrieve a gray frame
- Convert the gray frame to a binary image
- Retrieve a color frame
- Create a new color image that will be used to create the black, white, and red image



```
void _capture_ImageGrabbed(object sender, EventArgs e)
{
    Image<Gray, byte> bw_image = _capture.RetrieveGrayFrame().Resize(ima
    bw_image = bw_image.ThresholdBinary(new Gray(175), new Gray(255));
    Image<Bgr, Byte> color_image = _capture.RetrieveBgrFrame().Resize(im
    Image<Bgr, Byte> final = bw_image.Convert<Bgr, Byte>();
```

Black, White, Red

- Iterate through the color image pixels, if the pixel is red set the final images pixel color to red. Else, set the pixel to the corresponding binary image pixel value



```
for (int h = 0; h < color_image.Height; h++)
{
    for (int w = 0; w < color_image.Width; w++)
    {
        if (color_image.Data[h, w, 2] > (color_image.Data[h, w, 0] + color_image.Data[h, w, 1] + 30))
        {
            final.Data[h, w, 0] = 0;
            final.Data[h, w, 1] = 0;
            final.Data[h, w, 2] = 255;
        }
    }
}
imageBox1.Image = final;
```

L2Bot Line Following Stop at Color

L2Bot Line Following w/ Cone

- Objective: Create a Win Forms application that follows a white line using Emgu and the L2Bot. The application must also stop the robot when a piece of colored paper is encountered and restart when the paper is removed.

L2Bot Line Following w/ Cone

- Combine what you have learned in the following programs to successfully complete the challenge:
  - Stop at White
  - L2Bot Line Following
  - Black, White, and Red