**Project Report**

Project Name:

# **Converized Bar Inspection Checkout**

**Introduction:**

* **This project is used for image inspection**
* **In this project all work start after selecting or capturing image.**
* **development purpose developer can select image manually and perform inspection operation.**
* **In that project there are two main tabs:**
  + **Inspection and Reports**
* **Inside inspection tab multiple functionality available.**
* **Inside Report tab show the inspection report in date wise.**

**Working of Project:**

* Build or start project
* After that show dashboard window.
* **Inside that dashboard window. there are two main tabs / buttons.**
  + **Inspection**
  + **Report**

**Inside Inspection Tab**

* **Click on image button and select image folder for development or testing purpose.**
* **inside that folder store multiple image.**
* **After selecting folder then select camera and select part name.**
* **After completing this process then click start inspection button to start the image inspection.**
* **After that click next button to inspect one by one image.**
* **After capturing image show image Measurements, the format of mm by default. But user can customize measurement in mm, pixel or Inches**
* **After processing camera one and camera two image final result will be display in screen. in the form of Ok or Not Ok.**
* **After stop the inspection. Excel report open automatically. inside that excel store all measurement in mm or pixel.**

**Report:**

* **Inside report section report will be display in date wise.**

**Not Understanding Points:**

1. **onFrameEvent\_cam2()**
2. **HTuple**
3. **delegate**
4. **HSamrtWindowControl //understand**
5. **HWindow**
6. **NetworkInterface**
7. **Mutex**
8. **uEye // api used for camera live image**
9. HKeepAspectRatio
10. **Ping and pingReply**
11. ReadHoldingRegisters
12. WriteSingleRegister
13. EnumerateFiles
14. uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
15. Marshal
16. InspectCam2()
17. HObject / iconic variable
18. **initcommunication()**
19. **DeviceListAcq()**
20. **MV\_CC\_openDevice\_NET(); //using MyCamera class object call this method. And store result value inside nRet variable.**
21. **Open()**

**Basic Understanding Point:**

* In every programming language program execution start from main function. Because main is an entry point function.
* In C# Main() function available inside program.cs file.
* So first execute program.cs file.

**Inside Program.cs Inside Main()**

1. NetworkInterface[] nics = NetworkInterface.GetAllNetworkInterfaces();

* nics is a variable inside that variable store all value of network interface array.
* A NetworkInterface array that contains objects that describe the available network interfaces, or an empty array if no interfaces are detected.

1. String sMacAddress = string.Empty; //empty string
2. bool isGeniuene = false; //Boolean type with false value
3. foreach (NetworkInterface adapter in nics)

{

//not understand properly

IPInterfaceProperties properties = adapter.GetIPProperties();

sMacAddress = adapter.GetPhysicalAddress().ToString();

// if (sMacAddress == "509A4C13810D") //check match or not

{

isGeniuene = true;

}

}

* **adapter is a variable inside that variable access one by one nics value using for each loop.**

1. if (isGeniuene) //check condition

{

bool mutexCreated = true; //create Boolean variable

//not proper understand

System.Threading.Mutex mutex = new System.Threading.Mutex(true, "VisiMaster", out mutexCreated);

if (mutexCreated) // check condition

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

//Application.Run(new Login());

//if (Login.islogin== true)

{

Application.Run(new MasterForm()); //calling

}

}

else

{

System.Diagnostics.Process current = System.Diagnostics.Process.GetCurrentProcess();

foreach (System.Diagnostics.Process process in System.Diagnostics.Process.GetProcessesByName(current.ProcessName))

{

if (process.Id != current.Id)

{

MessageBox.Show("Another Instance of VisiMaster is Already Running.", "VisiMaster Already running", MessageBoxButtons.OK, MessageBoxIcon.Stop);

//SetForegroundWindow(process.MainWindowHandle);

break;

}

} //end foreach

} //end else

} //end if

else

{

MessageBox.Show("Please insert License file to proceed...!");

}

**Inside MasterForm.cs**

1. **InitilizedComponent(); //call inside that all property about design form.**
2. **HalconLicense = CheckHalconLicense();** //method and check condition

If condition is true then return true otherwise return false.

And return value store inside HalconLicense Boolean variable.

1. **if (HalconLicense)** //check conditions
2. **Inside if**

hSmartWindowControl1.HKeepAspectRatio = true;

hSmartWindowControl2.HKeepAspectRatio = true;

//HWindow is an instance of halcon window

//inside win\_result1 and 2 is used to store halcon window result

win\_result1 = hSmartWindowControl1.HalconWindow;

win\_result2 = hSmartWindowControl2.HalconWindow;

// hSmartWindowControl1.SetFullImagePart(null);

// hSmartWindowControl2.SetFullImagePart(null);

// Cam1ResultDict, Cam2ResultDict and FinalResultDict is an object of CurrentDictionary class. CuncurentDictionary class present inside namespace System.Collection. Concurrent

Cam1ResultDict = new ConcurrentDictionary<int, bool>();

Cam2ResultDict = new ConcurrentDictionary<int, bool>();

FinalResultDict = new ConcurrentDictionary<int, bool>();

ReadCalibration\_files(); //call function

1. **Inside ReadCalibration\_files(); //method**
   1. **Start try block //inside try block**
2. HTuple hv\_DirName; //create variable

HOperatorSet.GetCurrentDir(out hv\_DirName); //get current working directory and store inside hv\_DirName.

string path = hv\_DirName + "\\Calibration\\";// store path //not understand

var CamPara = Directory.EnumerateFiles(path, "\*.cal"); //EnumerateFile match path search pattern

var CamPose = Directory.EnumerateFiles(path, "\*.dat");

string a = CamPara.ToString(); //store value in a string

string b = CamPose.ToString(); //store value in b string

1. foreach (string currentFile in CamPara) //extract CamPara Dictionary value inside current file string.

{

a = currentFile; //CamPara currentFile store inside a.

}

foreach (string currentFile in CamPose)

{

b = currentFile; //CamPose currentFile store inside b.

}

1. try // read CamPara and CamPose and store output.

{

HOperatorSet.ReadCamPar(a, out Campara);

HOperatorSet.ReadPose(b, out Pose);

}

catch (HalconException ex) //Display Exception

{

MessageBox.Show("Unable read Calibration Files" + ex.Message);

}

// End Try Block

* 1. catch (Exception ex) //start catch block

{

MessageBox.Show(ex.ToString()); //show exception

}

**// Back to MasterForm.cs** // line no 156

**//Check Camera using if else statement**

1. if (usedcamera == "IDS") //start IDS camera if block

{

Cam2 = initCamera(Cam2, 1); // call initCamera() method.

**//Inside initCamera**

try // start try block

{

Cam = new Camera();// Cam is an object of camera class.

uEye.Defines.Status status = uEye.Defines.Status.NoSuccess; //status **-1** indicate **error** and **0** indicate **success.**

status = Cam.Init(camID); // store camera status inside status.

if (status != uEye.Defines.Status.Success) //check conditions

{

return null;

}

status = Cam.Parameter.Load("Cam" + camID + ".ini");//............camera parameter file..........

if (status != uEye.Defines.Status.Success) //status success nasel tar

{

MessageBox.Show("Parameter Load failed for Camera " + camID);

}

status = Cam.Memory.Allocate(); //allocate memory means success.

return Cam; // return Cam object and return value store inside Cam2.

}

// end of the try block

//start catch block

catch (NullReferenceException e)

{

MessageBox.Show("initcamera error\n" + e.Message);

return null;

}

catch (SystemException ex)

{

return null;

}

**// Back Inside IDS if return value store inside Cam2.**

if (Cam2 != null) // Cam2 is not equal then go inside if

{

Cam2.EventFrame += new System.EventHandler(this.onFrameEvent\_cam2);

// call onFrameEvent\_cam2() method

**// Inside onFrameEvent\_cam2()**

**onFrameEvent\_cam2:**

* after getting camera image onFrame\_Cam2 event will occurs.

Try **//start try block**

{

int FrameCountFromPLC = CounterCam2;

HObject Image;

HTuple height, width;

System.IntPtr tempMemory;

Cam2.Memory.GetActive(out tempMemory);

//HOperatorSet.GenImageInterleaved(out Image, tempMemory, "bgr", 2056, 1542, 0, "byte", 0, 0, 0, 0, -1, 0);

// HOperatorSet.Rgb1ToGray(Image, out Image);

//**GenImage1** create an image from pointer to pixel.

HOperatorSet.GenImage1(out Image, "byte", 600, 720, tempMemory);

if (isInspection == 1)

{

Image.DispObj(win\_result2);

if (isSetPartDoneCam2)

{

HOperatorSet.GetImageSize(Image, out width,out height);

//HOperatorSet class grouping a halcon operators.

//GetImageSize return the size of image height and width.

HOperatorSet.SetPart(win\_result2, 0, 0, height, width);

isSetPartDoneCam2 = false;

}

Transper\_Image Transper\_Image = new Transper\_Image(Image, FrameCountFromPLC);

Thread t = new Thread(new ParameterizedThreadStart(InspectCam2)); **// pass method call method not understand properly method present in line no 1355.**

* Inside inspection cam2 method all about the logic are present to inspect the image. And display result image Ok or NOk.
* **For example:**
  + **Measurement in Pixel, mm and inches**
* Inside that method one call **InspectSideCam()** and pass all required parameters.

**Inside InspectSideCam():**

* create one local iconic variable.
* Create local control **HTuple** variable.
* Initialize local and output iconic variables

**Dispose():**

* Dispose method is used to deallocate or free the system memory resources.

**Check Part Name:**

* Object. Part name is equal to specified part name or not.

**Catch():**

* Show the exceptions.

**finally ():**

**InvokeRequired():**

* Gets a value indicating whether the caller must call Invoke(Delegate, Object[]) when calling an object that implements this interface.
* true if the caller must call Invoke(Delegate, Object[]) otherwise, false.

**BeginInvoke():**

* accept delegate() method and array of objects. **Return** An IAsyncResult that represents the result of the BeginInvoke(Delegate) operation.

t.Start(Transper\_Image);

}

} // Try Block End

} **// end of Cam2 if**

} // end IDS if block

1. else if (usedcamera == "FLIR")

{

InitImageEventListener(); //call this method

**Inside InitImageEventListener():**

* one call **ImageEventListener1(win\_result1, this, 1);** and pass three parameter.
* Win\_result1 , MasterForm object and camera index. And store result inside **imageEventListner1.**

}

1. else if (usedcamera == "HIKVISION")

{

initcommunication(); //call initcommunication() method

DeviceListAcq(); // call DeviceListAcq()

Init(); // call Init() method

}

**Inside initcommunication():**

* **Ping**
* Allows an application to determine whether a remote computer is accessible over the network.
* Send request to plc server. And check server connected or not.
* **PingReply**
* class to return information about the operation, such as its status and the time taken to send the request and receive the reply.
* Its provide the reason for the failure if no message received.
* **Send(IpAddress , timeout)**
* The Send methods return instances of the PingReply class directly.
* **IpAddress** An System.Net.IPAddress that identifies the computer that is the destination for the ICMP echo message.
* **Timeout**
* An System.Int32 value that specifies the maximum number of milliseconds (after sending the echo message) to wait for the ICMP echo reply message.
* **Check Conditions inside try:**
  + **If** Connection is success, then create one thread object and call **Maintain\_Connection** method.
  + **Else** status false then show one dialog message. PLC Disconnected.Are you sure you want to Continue. And provide Three MessageBoxButtons(YesNoCancel).
  + and check button using if else statement.
  + **If** press **No** then kill the current process or terminate. Otherwise no any action.
* **Catch():**
  + Inside catch handle **NullReferenceException** and  **SystemException**.

**Inside Maintain\_Connection():**

* **isPLC\_Connected:**
  + **isPLC\_Connected** is a Boolean variable its check condition inside while loop. Condition is true then go inside loop.
* **Try:**
* Create two UInt16[] instance name as **CycleCompleted** and **temp.**
* **Call** ReadHoldingRegisters() and pass two parameter startAddress and number of point. And return value store under CycleCompleted.
* **Use** switch statement and display error message using error code.

**DeviceListAcq():**

* Inside DeviceListAcq() method check which type of camera connected and create connection.
* Types of camera.
  + Wired
  + USB
* **MV\_CC\_DEVICE\_INFO\_LIST:**
* MV\_CC\_DEVICE\_INFO\_LIST is the one of the structure inside that structure store one unsigned integer and one IntPtr [] pointer/ reference. **stDevList** is the instance of **MV\_CC\_DEVICE\_INFO\_LIST** structure.
* **Return value store inside nRet. nRet is the integer variable.**
* **Marshal class:**
* Provides a collection of methods for allocating unmanaged memory, copying unmanaged memory blocks, and converting managed to unmanaged types, as well as other miscellaneous methods used when interacting with unmanaged code.

**Init():**

* Inside Init() method check camera initialized or not.
* If camera is connected then call **Open()** method.
* Otherwise show **Exception** Camera initialization failed.Please check Camera connection.

**MasterForm\_Load Inside MasterForm.cs**

* **WindowState:** 
  + A FormWindowState that represents whether form is minimized, maximized, or normal. The default is **FormWindowState.Normal**.
* **Location:**
* Location is used to set windows position.
* The first number (X) is the distance from the left border of the container; the second number (Y) is the distance from the upper border of the container area, measured in pixels.
* **Size:**
* A Size that represents the size of the form.
* **Screen:**
  + Represents a display device or multiple display devices on a single system.
* **PPM\_timer = new System.Timers.Timer(10000):**
* Generates an event after a set interval, with an option to generate recurring events.
* PPM\_timer is an instance of timer class.
* **Elapsed:**
* Occurs when the interval elapses. After completed set time event will occur.
* **AutoReset:**
* Gets or sets a Boolean indicating whether the Timer should raise the Elapsed event only once (false) or repeatedly (true).
* **Start():**
  + **Used to start timer event.**
* **Login\_Data\_Ob = LoginData.getInstance();**
  + Call getInstance() method.
* **lbl\_user\_name.Text = Login\_Data\_Ob.User\_Name;**
  + **inside lbl\_user\_name.Text set to login object dot User\_Name.**