

```

1  using System;
2  using System.Drawing;
3  using System.Drawing.Imaging;
4  using System.Net;
5  using System.Net.Sockets;
6  using System.Threading;
7  using System.Windows.Forms;
8
9  namespace Xi410Ethernet
10 {
11     public partial class Xi410EthernetForm : Form
12     {
13         #region Constants
14         const int DEFAULT_PORT = 50101; //The Port the Camera is sending to
15         const int HEIGHT = 384; //the Width of the image
16         const int WIDTH = 242; //the Height of the image
17         const int IMG_WIDTH = 240; //there are 242 total lines, but only 240 of them
18         //contains image information
19         const int UDP_PACKAGE_LENGTH = 770; //the length of one udp package
20         const int META_DATA_INDEX = 240; //Each UDP package includes a line index,
21         //the 240 line includes the meta data
22         const int LAST_EMPTY_ROW_INDEX = 241; //Each UDP package includes a line
23         //index, the 241 line is empty and shows that a full frame arrived
24         const int FLAG_INDEX_IN_METADATA = 11; //the 11th byte in the metadata
25         //contains information of the flag.
26         const int IS_TEMPERATURE_MODE_IN_METADATA = 33;
27         const int IS_TEMPERATURE_MODE_BIT = 3;
28         #endregion
29
30         #region Private Member
31         private bool keepReading = false; //the start methods sets this to true and
32         //continue to read images from the device until
33         private UdpClient client = null;
34         private int framecounter = 0; //counts the number of frame that arrived. Used
35         //To calculate the frames per second.
36         private DateTime startPlayTime = DateTime.Now; //The Time the receiving of
37         //Images starts. Used to calculate the frames per second.
38         #endregion
39
40         #region UI
41         /// <summary>
42         /// The standart WinFormConstructor that initializes all UI Components
43         /// </summary>
44         public Xi410EthernetForm()
45         {
46             InitializeComponent();
47             PortLabel.Text = "Port " + DEFAULT_PORT;
48         }
49
50         /// <summary>
51         /// This gets called when the Form is shown and fully operational
52         /// </summary>
53         private void Xi410EthernetForm_Shown(object sender, EventArgs e)
54         {
55             Start();
56         }
57
58         /// <summary>
59         /// This Gets called when the form closes
60         /// </summary>
61         private void Xi410EthernetForm_FormClosing(object sender,
62             FormClosingEventArgs e)
63         {
64             //When the Form closes, no more pictures should be displayed
65             Stop();
66             if (client != null)
67             {
68                 client.Close();
69                 client = null;
70             }
71         }
72
73         /// <summary>

```

```

66     /// This methods shows a given Bitmap in the picturebox.
67     /// This is
68     /// </summary>
69     /// <param name="bitmap"></param>
70     private void video_NewFrame(Bitmap bitmap, double temperature)
71     {
72         //if a backgroundthread wants to access an UI element it need to invoke
73         //the uithread (asks the uithread to do it for him)
74         if (this.InvokeRequired)
75         {
76             try
77             {
78                 this.Invoke(new Action(() => video_NewFrame(bitmap,
79                     temperature)));
80             }
81             catch (ObjectDisposedException)
82             {
83                 //This can occur when the the form is closed but the video is
84                 //not stopped.
85             }
86         }
87         else
88         {
89             pictureBox1.BackgroundImage = bitmap; // this shows the imaage in
90             //the picture box.
91             frameCountLabel.Text = "FPS: " + (framecounter / (DateTime.Now -
92                 startPlayTime).TotalSeconds).ToString("0.0"); //
93             framecounter++;
94             CurrentTemperatureLabel.Text = "Current Temperature: " +
95             temperature.ToString("0.0") + "°C";
96         }
97     }
98 }
99
100 #endregion
101
102 #region Image Aquisition
103 private void Start()
104 {
105     Stop(); //Stop image aquisition when one is already active
106     framecounter = 0; //reset the framecounter. Needed to calculate the
107     //frames per seconds
108     startPlayTime = DateTime.Now; //remember the starttime. Needed to
109     //calculate the frames per seconds
110
111     //Start a backgroundThread which constantly checks the udp - port for
112     //new packages, collects them in an array
113     //then converts them to a bitmap and sends it to a picturebox.
114     Thread t = new Thread(Run);
115     t.Start();
116 }
117
118 /// <summary>
119 /// this stop the while loop in Run. The current processed Image will be
120 /// finished, but no new one will be started
121 /// </summary>
122 private void Stop()
123 {
124     keepReading = false;
125 }
126
127 /// <summary>
128 /// The opens a UDP-Client and in a loop recieves and processes the incoming
129 /// data
130 /// </summary>
131 private void Run()
132 {
133     try
134     {
135         int port = DEFAULT_PORT;
136         IPEndPoint remoteEP = new IPEndPoint(IPAddress.Any, port); //listen
137         //to any IP Address on the port
138         client = new UdpClient(port); //Create a new UdpClient
139         double[] newImg = new double[WIDTH * HEIGHT]; //Initiate an array

```

```

127         that stores the temperature of a frame
double[] lastImg = new double[WIDTH * HEIGHT]; //If the flag is
close, the new img shows wrong temperatures, so we show the last
valid one
128     int lastValidFrameCounter = 0; //remember the last valid framecounter
129     byte[] metaData = new byte[UDP_PACKAGE_LENGTH - 2]; //Initiate an
array that hold the meta information of a frame, the first 2 bytes
of the package indicate the line and frame and do not belong to the
metadata
130     keepReading = true; //keep reading
131     while (keepReading) //Continue reading the the udp client until
keepReading is set to false, i.e. the program is closed or the stop
button is clicked
132     {
133         byte[] data = client.Receive(ref remoteEP); //Wait for the next
udp package to arrive. When the firewall blocks this
application, this takes forever
134         if (data.Length != UDP_PACKAGE_LENGTH) continue; //check that the
package has length 770
135         int offset = data[0] * HEIGHT; //Each package has a row counter
on byte 0, a framecounter on byte 1 and the 2*384 bytes for the
temperatures of the pixels
136         for (int i = 0; i < HEIGHT; i++) //iterate over the width
137         {
138             newImg[i + offset] = (data[i * 2 + 3] * 256 + data[i * 2 +
2] - 1000) / 10; // use the temperature format t =
(byte1*256+byte2 -1000)/10
139         }
140         if (data[0] == META_DATA_INDEX) Array.Copy(data, 2, metaData, 0,
768); // MetaData comes in the 241th row
141         if (data[0] == LAST_EMPTY_ROW_INDEX) //this page is empty and
tells us a new img has arrived
142         {
143
144             //The Temperature Flag indicates that the image contains
temperature and not adu values
//If not set, asks the user to set it via Pix Connect
145             if (!IsTemperatureFlag(metaData))
146             {
147                 keepReading = false;
148                 MessageBox.Show("The Xi410 is set in ADU mode and not in
Temperature mode. Please go to the PixConnect and
change that.");
149             }
150             byte flagstate = metaData[FLAG_INDEX_IN_METADATA]; //get the
flag status
151             if (flagstate == 0 && newImg[HEIGHT * HEIGHT / 2 + WIDTH /
2] < 100) //take the image only if the new temperture is valid
152             {
153                 Array.Copy(newImg, lastImg, WIDTH * HEIGHT); //save the
image as last image
154                 lastValidFrameCounter = data[1]; //remeber the last valid
framecounter
155             }
156             ReadyNewFrame(lastImg);
157         }
158     }
159     client.Close();
160     client = null;
161 }
162 //If the Connection breaks
163 catch (SocketException ex)
164 {
165     if (client != null)
166     {
167         client.Close();
168         client = null;
169     }
170     System.Diagnostics.Debug.WriteLine(ex.ToString());
171 }
172 }
173 }
174 }
175 }

```

```

176 private static bool IsTemperatureFlag(byte[] metaData)
177 {
178     //First Chech if MetaData are valid
179     if (metaData == null || metaData.Length < UDP_PACKAGE_LENGTH - 2)
180     {
181         return false;
182     }
183     //Check if the 3rd bit in the metadatas 33th byte is set
184     return ((metaData[IS_TEMPERATURE_MODE_IN_METADATA]) & (1 <<
185         IS_TEMPERATURE_MODE_BIT)) > 0;
186 }
187
188 /// <summary>
189 /// this methods transforms the incoming temperature data to a Bitmap and
190 /// then calls video_NewFrame to show it in the pictureBox
191 /// </summary>
192 /// <param name="img">the whole frame with temperatures on </param>
193 private void ReadyNewFrame(double[] img)
194 {
195     //Get 3 sigma min and max value of the image;
196     int img_length = IMG_WIDTH * HEIGHT;
197
198     //Calculate the mean value
199     double sum = 0;
200     for (int i = 0; i < img_length; i++)
201     {
202         sum += img[i];
203     }
204
205     double mean = (double)sum / img_length;
206
207     //Calculate the Variance
208     sum = 0;
209     for (int i = 0; i < img_length; i++)
210     {
211         sum += (mean - img[i]) * (mean - img[i]);
212     }
213     double variance = sum / img_length;
214     variance = Math.Sqrt(variance);
215     variance *= 3; // 3 Sigma
216
217     //Calculate min and max
218     double min = mean - variance;
219     double max = mean + variance;
220
221     //Create a two-dimensional array with gray values
222     double[,] rawImage = new double[IMG_WIDTH, HEIGHT];
223     for (int y = 0; y < rawImage.GetLength(1); y++)
224     {
225         for (int x = 0; x < rawImage.GetLength(0); x++)
226         {
227             double d = img[x * HEIGHT + y];
228             if (d > max)
229             {
230                 rawImage[x, y] = 1;
231             }
232             else if (d < min)
233             {
234                 rawImage[x, y] = 0;
235             }
236             else
237             {
238                 rawImage[x, y] = (d - min) / (max - min);
239             }
240         }
241     }
242     //Convert gray image to bitmap
243     Bitmap bitmap = ToBitmap(rawImage);
244
245     //Lauch New Frame Event with bitmap and metadatas of the frame
246     video_NewFrame(bitmap, img[img.Length / 2]);

```

```

247     }
248
249     /// <summary>
250     /// Transforms a double array of Gray data, i.e. values between 0 and 1,
    into a bitmap.
251     /// </summary>
252     /// <param name="rawImage">The Raw gray image</param>
253     /// <returns>The Bitmap of the gray image</returns>
254     private static unsafe Bitmap ToBitmap(double[,] rawImage)
255     {
256         int width = rawImage.GetLength(1);
257         int height = rawImage.GetLength(0);
258
259         Bitmap image = new Bitmap(width, height);
260         BitmapData bitmapData = image.LockBits(
261             new Rectangle(0, 0, width, height),
262             ImageLockMode.ReadWrite,
263             PixelFormat.Format32bppArgb
264         );
265         ColorARGB* startingPosition = (ColorARGB*)bitmapData.Scan0;
266
267
268         for (int i = 0; i < height; i++)
269         {
270             for (int j = 0; j < width; j++)
271             {
272                 double color = rawImage[i, j];
273                 byte rgb = (byte)(color * 255);
274
275                 ColorARGB* position = startingPosition + j + i * width;
276                 position->A = 255;
277                 position->R = rgb;
278                 position->G = rgb;
279                 position->B = rgb;
280             }
281         }
282
283         image.UnlockBits(bitmapData);
284         return image;
285     }
286
287     /// <summary>
288     /// This defines a structure of 32ARGB Color, one byte each for alpha, red,
    green, red
289     /// A (from 0 to 255) represents alpha (translucency)
290     /// R (from 0 to 255) represents red
291     /// G (from 0 to 255) represents green
292     /// B (from 0 to 255) represents blue
293     /// </summary>
294     public struct ColorARGB
295     {
296         public byte B;
297         public byte G;
298         public byte R;
299         public byte A;
300
301         public ColorARGB(Color color)
302         {
303             A = color.A;
304             R = color.R;
305             G = color.G;
306             B = color.B;
307         }
308
309         public ColorARGB(byte a, byte r, byte g, byte b)
310         {
311             A = a;
312             R = r;
313             G = g;
314             B = b;
315         }
316
317         public Color ToColor()

```

```
318         {
319             return Color.FromArgb(A, R, G, B);
320         }
321     }
322 #endregion
323
324 }
325 }
```