

```

1  using Xi80Ethernet;
2  using System;
3  using System.Drawing;
4  using System.Drawing.Imaging;
5  using System.Net;
6  using System.Net.Sockets;
7  using System.Runtime.InteropServices;
8  using System.Text;
9  using System.Threading;
10 using System.Windows.Forms;
11 using System.Threading.Tasks;
12
13 namespace Xi80Ethernet
14 {
15     public partial class Xi80EthernetForm : Form
16     {
17
18         #region Constants
19         const int DEFAULT_PORT = 50101; //The Port the Camera is sending to
20         const int RAW_HEIGHT = 84; //the Width of the image
21         const int RAW_WIDTH = 80; //the Height of the image
22         const int IMG_WIDTH = 80; //there are 242 total lines, but only 240 of them
           contains image information
23         const int IMG_HEIGHT = 80;
24         const int BYTES_PRO_PIXEL = 2;
25         const int META_DATA_SIZE = 160;
26         const int META_DATA_POSITION = 322;
27         const int UDP_PACKAGE_LENGTH = 482; //the length of one udp package
28         const int META_DATA_INDEX = 78; //Each UDP package includes a line index, the
           240 line includes the meta data
29         const int LAST_EMPTY_ROW_INDEX = 81; //Each UDP package includes a line
           index, the 241 line is empty and shows that a full frame arived
30         const int FLAG_INDEX_IN_METADATA = 11; //the 11th byte in the metadata
           contains information of the flag.
31         const int IS_TEMPERATURE_MODE_IN_METADATA = 33;
32         const int IS_TEMPERATURE_MODE_BIT = 3;
33         #endregion
34
35         #region Private Member
36         private bool keepReading = false; //the start methds sets this to true and
           continue to read images from the device until
37         private int framecounter = 0; //counts the number of frame that arived. Used
           To calculate the frames per second.
38         private DateTime startPlayTime = DateTime.Now; //The Time the recieving of
           Images starts. Used to calculate the frames per second.
39         #endregion
40
41         #region UI
42         /// <summary>
43         /// The standart WinFormConstructor that initializes all UI Components
44         /// </summary>
45         public Xi80EthernetForm()
46         {
47             InitializeComponent();
48             PortLabel.Text = "Port " + DEFAULT_PORT;
49         }
50
51         /// <summary>
52         /// This Gets called when the form closes
53         /// </summary>
54         private void Xi80EthernetForm_FormClosing(object sender,
           FormClosingEventArgs e)
55         {
56             //When the Form closes, no more pictures should be displayed
57             Stop();
58         }
59
60         /// <summary>
61         /// This gets called when the form is shown. We start listening to the
           Ethernetport immediately.
62         /// </summary>
63         private void Xi80EthernetForm_Shown(object sender, EventArgs e)
64         {

```

```

65         StartEthernet();
66     }
67
68     /// <summary>
69     /// This methods shows a given Bitmap in the picturebox.
70     /// This is
71     /// </summary>
72     /// <param name="bitmap"></param>
73     private void video_NewFrame(Bitmap bitmap, double temperature)
74     {
75         //if a backgroundthread wants to access an UI element it need to invoke
76         //the uithread (asks the uithread to do it for him)
77         if (this.InvokeRequired)
78         {
79             try
80             {
81                 this.Invoke(new Action(() => video_NewFrame(bitmap,
82                     temperature)));
83             }
84             catch (ObjectDisposedException)
85             {
86                 //This can occur when the the form is closed but the video is
87                 //not stopped.
88             }
89         }
90         else
91         {
92             pictureBox1.BackgroundImage = bitmap; // this shows the image in
93             //the picture box.
94             frameCountLabel.Text = "FPS: " + (framecounter / (DateTime.Now -
95                 startPlayTime).TotalSeconds).ToString("0.0"); //
96             framecounter++;
97             CurrentTemperatureLabel.Text = "Current Temperature: " +
98             temperature.ToString("0.0") + "°C";
99         }
100     }
101
102     #endregion
103
104     #region Image Aquisition
105     private void ReadyNewRawFrame(short[] rawFrame)
106     {
107         double[] tempImg = new double[IMG_HEIGHT * IMG_WIDTH];
108         for (int i = 0; i < IMG_HEIGHT * IMG_WIDTH; i++)
109         {
110             tempImg[i] = (double)(rawFrame[i] - 1000) / 10d;
111         }
112         ReadyNewFrame(tempImg);
113     }
114
115     private void ReadyNewRawFrame(byte[] img)
116     {
117         double[] tempImg = new double[IMG_HEIGHT * IMG_WIDTH];
118         for (int i = 0; i < IMG_HEIGHT * IMG_WIDTH; i++)
119         {
120             tempImg[i] = (double)(img[2 * i] + img[2 * i + 1] * 2 << 8 - 1000) /
121             10d;
122         }
123         ReadyNewFrame(tempImg);
124     }
125
126     /// <summary>
127     /// this stop the while loop in Run. The current processed Image will be
128     /// finished, but no new one will be started
129     /// </summary>
130     private void Stop()
131     {
132         keepReading = false;
133     }
134
135     #region Ethernet

```

```

130
131 private void StartEthernet()
132 {
133     Stop();
134     framecounter = 0;
135     startPlayTime = DateTime.Now; //remember the starttime. Needed to
        calculate the frames per seconds
136     Thread t = new Thread(Run);
137     t.Name = "EthernetThread";
138     t.IsBackground = true;
139     t.Start();
140 }
141
142
143 /// <summary>
144 /// The opens a UDP-Client and in a loop recieves and processes the incoming
        data
145 /// </summary>
146 private void Run()
147 {
148
149     int port = DEFAULT_PORT;
150     IPEndPoint remoteEP = new IPEndPoint(IPAddress.Any, port); //listen to
        any IP Address on the port
151     UdpClient client = new UdpClient(port); //Create a new UdpClient
152     double[] newImg = new double[RAW_WIDTH * RAW_HEIGHT]; //Initiate an array
        that stores the temperature of a frame
153     double[] lastImg = new double[RAW_WIDTH * RAW_HEIGHT]; //If the flag is
        close, the new img shows wrong temperatures, so we show the last valid
        one
154     int lastValidFrameCounter = 0; //remember the last valid framecounter
155     byte[] metaData = new byte[META_DATA_SIZE]; //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
156     keepReading = true; //keep reading
157     while (keepReading) //Continue reading the the udp client until
        keepReading is set to false, i.e. the programm is closed or the stop
        button is clicked
158     {
159         byte[] data = client.Receive(ref remoteEP); //Wait for the next udp
        package to arrive. When the firewall blockes this application, this
        takes forever
160         if (data.Length != UDP_PACKAGE_LENGTH) continue; //check that the
        package has length 770
161         int offset = data[0] * RAW_WIDTH; //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
162         for (int i = 0; i < RAW_WIDTH*3; i++) //iterate over the width
163         {
164             newImg[i + offset] = (data[i * 2 + 3] * 256 + data[i * 2 + 2] -
                1000) / 10d; // use the temperature format t = (byte1*256+byte2
                -1000)/10
165         }
166         if (data[0] == META_DATA_INDEX)
167         {
168             Array.Copy(data, META_DATA_POSITION, metaData, 0,
                META_DATA_SIZE); // MetaData comes in the 241th row
169         }
170         if (data[0] == LAST_EMPTY_ROW_INDEX) //this page is empty and tells
        us a new img has arrived
171         {
172
173             //The Temperature Flag indicates that the image contains
            temperature and not adu values
174             //If not set, asks the user to set it via Pix Connect
175             if (!IsTemperatureFlag(metaData))
176             {
177                 keepReading = false;
178                 MessageBox.Show("The Xi80 is set in ADU mode and not in
                    Temperature mode. Please go to the PixConnect and change
                    that.");
179             }
180             byte flagstate = metaData[FLAG_INDEX_IN_METADATA]; //get the flag

```

```

181         status
            if (flagstate == 0 && newImg[RAW_HEIGHT * RAW_HEIGHT / 2 +
RAW_WIDTH / 2] < 100)//take the image only if the new tempeture
is valid
182         {
183             Array.Copy(newImg, lastImg, RAW_WIDTH * RAW_HEIGHT);//save
the image as last image
184             lastValidFrameCounter = data[1];//remeber the last valid
framecounter
185         }
186         ReadyNewFrame(lastImg);
187     }
188 }
189 client.Close();
190 }
191 #endregion
192
193 private static bool IsTemperatureFlag(byte[] metaData)
194 {
195     //First Chech if MetaData are valid
196     if (metaData == null || metaData.Length < META_DATA_SIZE)
197     {
198         return false;
199     }
200     //Check if the 3rd bit in the metadatas 33th byte is set
201     return ((metaData[IS_TEMPERATURE_MODE_IN_METADATA]) & (1 <<
IS_TEMPERATURE_MODE_BIT)) > 0;
202 }
203
204 /// <summary>
205 /// this methods transforms the incoming temperature data to a Bitmap and
then calls video_NewFrame to show it in the picturebox
206 /// </summary>
207 /// <param name="img">the whole frame with temperatures on </param>
208 private void ReadyNewFrame(double[] img)
209 {
210     //Get 3 sigma min and max value of the image;
211     int img_length = IMG_WIDTH * IMG_HEIGHT;
212
213     //Calculate the mean value
214     double sum = 0;
215     for (int i = 0; i < img_length; i++)
216     {
217         sum += img[i];
218     }
219
220     double mean = (double)sum / img_length;
221
222     //Calculate the Variance
223     sum = 0;
224     for (int i = 0; i < img_length; i++)
225     {
226         sum += (mean - img[i]) * (mean - img[i]);
227     }
228     double variance = sum / img_length;
229     variance = Math.Sqrt(variance);
230     variance *= 3; // 3 Sigma
231
232     //Calculate min and max
233     double min = mean - variance;
234     double max = mean + variance;
235
236     //Create a two-dimensional array with gray values
237     double[,] rawImage = new double[IMG_WIDTH, IMG_HEIGHT];
238     for (int y = 0; y < rawImage.GetLength(1); y++)
239     {
240         for (int x = 0; x < rawImage.GetLength(0); x++)
241         {
242             double d = img[x * IMG_HEIGHT + y];
243             if (d > max)
244             {
245                 rawImage[x, y] = 1;
246             }

```

```

247         else if (d < min)
248         {
249             rawImage[x, y] = 0;
250         }
251         else
252         {
253             rawImage[x, y] = (d - min) / (max - min);
254         }
255     }
256 }
257
258 //Convert gray image to bitmap
259 Bitmap bitmap = ToBitmap(rawImage);
260
261 //Lauch New Frame Event with bitmap and metadatas of the frame
262 video_NewFrame(bitmap, img[img.Length / 2]);
263 }
264
265 /// <summary>
266 /// Transforms a double array of Gray data, i.e. values between 0 and 1,
267 /// into a bitmap.
268 /// </summary>
269 /// <param name="rawImage">The Raw gray image</param>
270 /// <returns>The Bitmap of the gray image</returns>
271 private static unsafe Bitmap ToBitmap(double[,] rawImage)
272 {
273     int width = rawImage.GetLength(1);
274     int height = rawImage.GetLength(0);
275
276     Bitmap image = new Bitmap(width, height);
277     BitmapData bitmapData = image.LockBits(
278         new Rectangle(0, 0, width, height),
279         ImageLockMode.ReadWrite,
280         PixelFormat.Format32bppArgb
281     );
282     ColorARGB* startingPosition = (ColorARGB*)bitmapData.Scan0;
283
284     for (int i = 0; i < height; i++)
285     {
286         for (int j = 0; j < width; j++)
287         {
288             double color = rawImage[i, j];
289             byte rgb = (byte)(color * 255);
290
291             ColorARGB* position = startingPosition + j + i * width;
292             position->A = 255;
293             position->R = rgb;
294             position->G = rgb;
295             position->B = rgb;
296         }
297     }
298
299     image.UnlockBits(bitmapData);
300     return image;
301 }
302
303 /// <summary>
304 /// This defines a structure of 32ARGB Color, one byte each for alpha, red,
305 /// green, red
306 /// A (from 0 to 255) represents alpha (translucency)
307 /// R (from 0 to 255) represents red
308 /// G (from 0 to 255) represents green
309 /// B (from 0 to 255) represents blue
310 /// </summary>
311 public struct ColorARGB
312 {
313     public byte B;
314     public byte G;
315     public byte R;
316     public byte A;
317
318     public ColorARGB(Color color)

```

```
318         {
319             A = color.A;
320             R = color.R;
321             G = color.G;
322             B = color.B;
323         }
324
325     public ColorARGB(byte a, byte r, byte g, byte b)
326     {
327         A = a;
328         R = r;
329         G = g;
330         B = b;
331     }
332
333     public Color ToColor()
334     {
335         return Color.FromArgb(A, R, G, B);
336     }
337 }
338
339
340 #endregion
341
342
343
344     }
345 }
346 }
```