AC 2008-1015: A CORRELATION DETECTOR SIMULATION

James Reising, University of Evansville

JAMES A. REISING is an Associate Professor of Electrical Engineering at the University of Evansville, Evansville, Indiana, where he has taught since 1980. Prior to that time he was employed by Eagle-Picher Industries at the Miami Research Laboratories and the Electro-Optic Materials Department. He is a senior member of IEEE.

Mark Randall, University of Evansville

Mark E. Randall is a Staff Engineer for the University of Evansville College of Engineering and Computer Science. He is presently working on a Master of Science Degree in Computer Science and Engineering from the University of Evansville. Mark has two undergraduate degrees: a Bachelor of Science degree in Electronics Technology from Indiana State University and a Bachelor of Science degree in Electrical Engineering from the University of Evansville. Mark has competed in several IEEE SoutheastCon student hardware contests and was the co-designer of the University of Evansville's 2006 winning entry "Gizmo".

A Correlation Detector Simulation

Abstract

In some detection processes, such as radar, sonar, and seismology, the cross-correlation of a known signal and a time-delayed version of the same signal with additive noise can be used to determine the time delay between the transmitted pulse and the return pulse. Such a procedure is relatively easy to rationalize intuitively. (The scheme may also be viewed as filtering the return signal with a matched filter, but the convolution of the signal with the impulse response of the matched filter may be harder for students, especially those who have studied neither linear systems nor digital filters, to grasp.)

A computer simulation of the cross-correlation detection process was originally assigned as a graphical programming project in a computer software class. The simple model used in the assignment assumed a sinusoidal signal of adjustable duration as the transmitted pulse and a delayed version of the same pulse with added white noise as the received pulse. Two noise models were available to the user: zero-mean uniformly distributed white noise and zero-mean Gaussian white noise of adjustable variance. The amplitude of the added noise was to be adjustable by the user. The transmitted pulse and the received signal were displayed in separate plots, and the cross-correlation of the transmitted pulse and the received signal was displayed in a third plot. Both signals and the cross-correlation were represented by finite length sequences.

A simple view of the detection process is included as part of the assignment. The transmitted signal is a pattern to be compared to the received signal at different time offsets. Since the noise is random and many samples are taken during one period of the sinusoidal pulse, the noise tends to cancel as the terms in the cross-correlation are added. The maximum “match” of the pattern occurs when the pattern is offset by the time delay represented in the received signal. The graphical presentation showed that, even in cases where a visual inspection of the received signal showed no hint of a pulse in the received signal, the cross-correlation clearly indicated the delay time of the received pulse.

The program allowed users to experiment with different noise amplitudes and delay times, and the visual presentation was quite dramatic. The concept seemed useful enough that the authors developed a program with a bit more functionality for use in a random processes course, where the topic of matched filters was part of the course material. In addition to a simple time delay, the pulse in the received signal was attenuated.

Introduction

The original project was assigned in a computer software course covering C#.NET. Most of the students in the course were computer engineering majors. The focus of the programming aspect of the problem was the graphical user interface and graphics. The problem itself was chosen to include topics from a random processes course. The computer program provided a tool to enhance student understanding of a topic that had appeared difficult for students to grasp in the past.

The program described in this paper was written by Mark Randall as part of a graduate version of the programming class.

Model Assumptions

The model used as the basis for the computer program assumed a situation in which a sinusoidal pulse was transmitted. A received signal was modeled as the sum of an attenuated, delayed version of the transmitted pulse with additive white noise.

The transmitted pulse is sinusoidal with amplitude that may be varied by the user in arbitrary units. The number of cycles transmitted and the sampling frequency in samples per cycle may also be varied by the user.

Either of two noise models is user-selectable: uniformly distributed zero-mean or zero-mean Gaussian noise. The user may also vary the noise amplitude in the same arbitrary units used for the transmitted pulse.

The received signal is the sum of the noise and the transmitted pulse, delayed by the sample periods selected by the user (constrained so the "received" pulse cannot overlap the transmitted pulse) with amplitude attenuated by the square of the reciprocal delay.

Both the transmitted signal and the received signal are represented by sequences of length N (set to 2000 in the program). The sequence representing the transmitted pulse is zero-padded to length N.

The detector output is computed as the cross-correlation[1] of the received signal and the transmitted pulse sequences: $R(k) = \frac{1}{N} \sum_{m=0}^{N-1-k} x_m y_{k+m}$, where x_m is a sample of the transmitted signal and y_{k+m} is a sample of the return signal, $0 \leq k \leq N - 1$, and N is the number of samples in each sequence.

Programming Considerations

The program consists of a main form with three plots. User controls allow changes in the following parameters: the number of cycles, amplitude, and frequency of the transmitted pulse; the delay of the received signal; and the type and amplitude of the noise. Additional controls allow the user to have the program generate a new noise sample or display a histogram of the noise in a pop-up window. The S/N ratio in the simulated signal plus noise is also displayed. The topmost plot displays the transmitted pulse. The middle plot displays the simulated received signal plus noise, and the bottom plot shows the detector output. The screen at startup is as shown in Figure 1below:

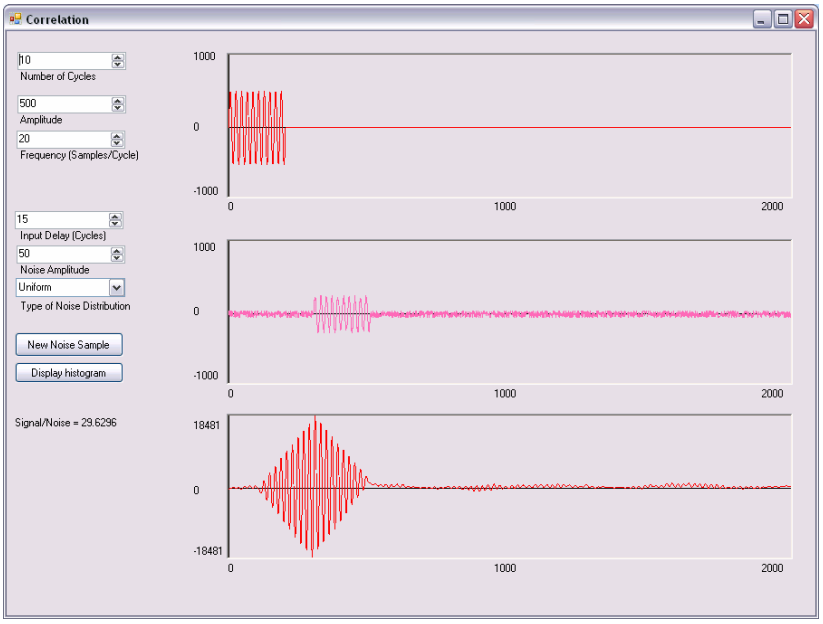

Figure 1

All signals and the noise are represented as arrays of point objects (having an X and a Y component). The array representing the transmitted pulse is computed as $\sin(2\pi\frac{n}{T})$, where T represents the period of the sinusoid in samples and n ranges from 0 to the duration of the transmitted pulse in samples. The array is then zero-padded to be of length N. The Y values of this array are later scaled by the transmitted pulse amplitude entered by the user.

As in most programming languages, the built-in pseudorandom number generator produces uniformly distributed values between zero and one. The random noise array is constructed using this function for the noise model selected by the user.

Zero-mean uniform noise samples are generated by scaling and shifting the numbers produced by the built-in generator to create an array of numbers uniformly distributed on (-1, 1). This array is later scaled by the noise amplitude entered by the user.

An array of unit variance zero-mean Gaussian distributed random numbers is generated using the built-in random number generator by means of the following method described in *Numerical Recipes in C*[2]:

> If x_1 and x_2 are sequences of uniformly (0,1) random numbers, both $y_1 = \sqrt{-2\ln x_1}\cos 2\pi x_2$ and $y_2 = \sqrt{-2\ln x_1}\sin 2\pi x_2$ are sequences of Gaussian random numbers with unity standard deviation and zero mean.

This array is later scaled by the noise amplitude entered by the user.

The Y values of the transmitted pulse array are then shifted by the delay (in samples) entered by the user and scaled by the reciprocal of the square of the delay. This shifted, scaled version of the transmitted pulse is then added to the noise array to produce the simulated received signal array.

Finally, the detector output is computed as the cross correlation of the transmitted pulse array and the received signal array.

The arrays of point objects are used to produce plots of the transmitted pulse, received signal, and detector output.

Another feature added to the program to aid in visualization is the option of displaying a histogram of the noise model. A screen shot of the histogram for Gaussian noise is shown in Figure 2 below:

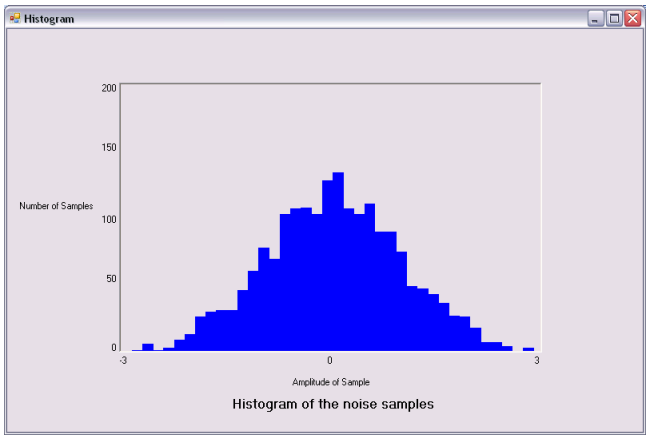

Figure 2

A screen shot of the program output for a case where the received pulse is hardly identifiable in the plot of the received signal is shown below in Figure 3:

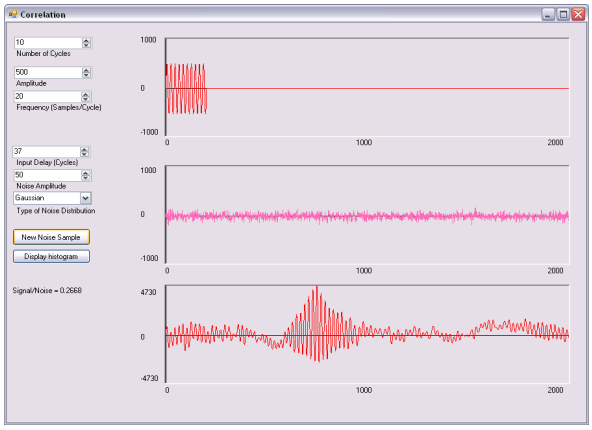

Figure 3

The position of the received pulse is clearly visible in the detector output plot. While the top two plots are shown with fixed units, the bottom plot is auto-scaled in the program so the peak amplitude of the output takes up almost the entire vertical range of the plot. Note that the signal to noise power ratio for the simulated received signal is displayed to the left of the detector output plot to indicate the sensitivity of the correlation method.

The code for the two forms is listed in the appendix. The program executable for .NET 2.0 is posted at http://csserver.evansville.edu/~reising/project3.exe .

Conclusion

Informal comments from students indicate that including topics from other courses in programming assignments makes the work more interesting. The finished product is useful as a learning tool as well. Intended future enhancements to the program include adding distortion to the signal component of the received signal.

Bibliography

1. Mitra, Sanjit K. *Digital Signal Processing 3rd Edition*. New York: McGraw-Hill, 2006.
2. Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C The Art of Scientific Computing 2nd Edition*. New York: Cambridge University Press, 1992.

Appendix

```
//Form1.cs
//By Mark Randall
// rev by J. Reising
//This program was written to create a correlation simulator (Sonar, Radar).

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

enum SAMPLES :int {max=2000};

namespace project3
{
   public partial class Form1 : Form
   {
```

```
   Pen penRed = new Pen(Color.Red);                // define red pen
   Pen penBlue = new Pen(Color.Blue);              // define blue pen
   Pen penBlack = new Pen(Color.Black); // define black pen
   Pen penPink = new Pen(Color.HotPink);           // define pink pen
   int cx;              // integer variables to hold dimensions of client area
   int cy;              // in pixels

   int duration = 200;  // Initial value for length of pulse in samples

   int inputamp = 500; // Initial value for the simulated transmitted pulse amplitude
   int noiseamp = 50; // Initial value for the noise amplitude

   int inputperiod = 20; // Initial value for the simulated transmitted pulse period in samples

   int sigmav = 1;   //variable  to hold value of sigma initialized to 1
   static public bool uniform = true, gaussian = false; // Defines the type of noise noise model

   PointF[] ipt = new PointF[(int)(int)SAMPLES.max];              // array of points for
transmitted pulse
   static public PointF[] noise = new PointF[(int)SAMPLES.max]; //array of points for noise
   PointF[] npt = new PointF[(int)SAMPLES.max];           // array of points for noise + signal
points

   int delay = 300;    // Initial value for delay of received pulse in samples
   int[] noiseAmplitude = new int[(int)SAMPLES.max];
   int step = 0;
   //set is a bool that is set after a noise dist. has been created setting set
   //to false cause a new random noise array to be created
   bool set = false;
   float maxdecay=0; //variable used to store max amp. of decayed signal
   float decay = 0;  //temp variable used when finding max amp. of decayed signal



   public Form1()
   {
      InitializeComponent();
   }

   // Panels size and size change handling
   protected override void OnResize(EventArgs e)
   {
      //The following lines of code allow the user to resize the window without distorting the
graphs
      panel1.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .273),
(int)(this.ClientSize.Height * .038));
```

```
        panel2.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .273),
(int)(this.ClientSize.Height * .357));
        panel3.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .273),
(int)(this.ClientSize.Height * .655));
        label8.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .228),
(int)(this.ClientSize.Height * .1538));
        label9.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .228),
(int)(this.ClientSize.Height * .2638));
        label10.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .228),
(int)(this.ClientSize.Height * .0338));
        label11.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .228),
(int)(this.ClientSize.Height * .360));
        label12.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .228),
(int)(this.ClientSize.Height * .580));
        label13.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .228),
(int)(this.ClientSize.Height * .47));
        label23.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .228),
(int)(this.ClientSize.Height * .665));
        label25.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .228),
(int)(this.ClientSize.Height * .776));
        label24.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .228),
(int)(this.ClientSize.Height * .88));

        label14.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .27),
(int)(this.ClientSize.Height * .29));
        label15.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .6),
(int)(this.ClientSize.Height * .29));
        label16.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .93),
(int)(this.ClientSize.Height * .29));
        label17.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .93),
(int)(this.ClientSize.Height * .61));
        label18.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .6),
(int)(this.ClientSize.Height * .61));
        label19.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .27),
(int)(this.ClientSize.Height * .61));
        label20.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .93),
(int)(this.ClientSize.Height * .91));
        label21.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .6),
(int)(this.ClientSize.Height * .91));
        label22.Location = new System.Drawing.Point((int)(this.ClientSize.Width * .27),
(int)(this.ClientSize.Height * .91));

        this.Invalidate();          // force repaint of client area on size change

    }
```

```
  // Paints the panel containing the simulated transmitted pulse.
  private void panel1_Paint(object sender, PaintEventArgs e)
  {
     Graphics grfx = e.Graphics;               // get graphics context
     panel1.Width = (int)(this.ClientSize.Width * .7);  // Defines the Width
     panel1.Height = (int)(this.ClientSize.Height * .25); // Defines the Height
     cx = this.panel1.Size.Width;
     cy = this.panel1.Size.Height;
     // move origin to center of client area
     grfx.TranslateTransform(0, (float)cy / 2);
     // define brush and color for text
     SolidBrush txtBrush = new SolidBrush(Color.Crimson);
     // Scale coordinates to convenient units & invert y
     grfx.ScaleTransform((float)cx / (int)SAMPLES.max, -(float)cy / (int)SAMPLES.max);
     grfx.DrawLine(penBlack, 0, 0, (int)SAMPLES.max, 0);                // x-axis
     grfx.DrawLine(penBlack, 0, ((int)SAMPLES.max/2), 0, -((int)SAMPLES.max/2));
     // y-axis
     //The next lines of code simple are used to automaticly set the limits on the graph
     label9.Text = Convert.ToString(-(int)SAMPLES.max / 2);
     label10.Text = Convert.ToString((int)SAMPLES.max / 2);
     label15.Text = Convert.ToString((int)SAMPLES.max / 2);
     label16.Text = Convert.ToString((int)SAMPLES.max);
     //The next 2 for loops create the input signal using the function SIN(2*PI*i)/period,
where i varies
     //from 0 to the length of the pulse in samples
     //The points are placed in the ipt X,Y arrays
     for (int i = 0; i < duration; i++)
     {
        ipt[i].X = (i);
        ipt[i].Y = inputamp * (float)(Math.Sin((2 * Math.PI * ipt[i].X) *
(1/((float)(inputperiod)))));
        // Equation of the transmitted pulse
     }
     for (int i = duration; i < (int)SAMPLES.max; i++)
     {
        ipt[i].X = i;
        ipt[i].Y = 0;
        // zero-padding the transmitted pulse array to required length
     }

     grfx.DrawLines(penRed, ipt);                              // draw the curve through the points

     panel2.Refresh();    //refresh noise pannel
     panel3.Refresh();    //refresh correlation pannel
  }
```

```
//_____________________________________________________________________________
    // Paints the panel containing the pulse with added noise.
    private void panel2_Paint(object sender, PaintEventArgs e)
    {

       Graphics grfx = e.Graphics;               // get graphics context
       panel2.Width = (int)(this.ClientSize.Width * .7); // Defines the Width
       panel2.Height = (int)(this.ClientSize.Height * .25); // Defines the Height
       cx = this.panel2.Size.Width;
       cy = this.panel2.Size.Height;
       // move origin to center of client area
       grfx.TranslateTransform(0, (float)cy / 2);
       // define brush and color for text
       SolidBrush txtBrush = new SolidBrush(Color.Crimson);
       // Scale coordinates to convenient units & invert y

       //The next lines of code simple are used to automaticly set the limits on the graph
       label12.Text = Convert.ToString(-(int)SAMPLES.max / 2);
       label11.Text = Convert.ToString((int)SAMPLES.max / 2);
       label18.Text = Convert.ToString((int)SAMPLES.max / 2);
       label17.Text = Convert.ToString((int)SAMPLES.max);

       grfx.ScaleTransform((float)cx / (int)SAMPLES.max, -(float)cy / (int)SAMPLES.max);
       grfx.DrawLine(penBlack, 0, 0, (int)SAMPLES.max, 0);                    // x-axis
       grfx.DrawLine(penBlack, 0, ((int)SAMPLES.max/2), 0, -((int)SAMPLES.max/2));
            // y-axis
       // calculate noise

       int i = 0;
       if (set != true)
       {
          Random rangen = new Random();

          if (uniform)// Uniform noise model
          {
             for (int n = 0; n < (int)SAMPLES.max; n++)
             {
                npt[n].X = (n);
                noise[n].Y = (2 * Convert.ToSingle(rangen.NextDouble()) - 1);
                //uniform (-1,1) distribution of amplitudes
             }
          }
          if (gaussian) // Gaussian noise model
          {

             double x1 = 0, x2 = 0, y1 = 0, y2 = 0, z1 = 0, z2 = 0;
```

```
            for (int n = 0; n < (int)SAMPLES.max; n += 2)
            {
            x1 = Convert.ToSingle(rangen.NextDouble());
            x2 = Convert.ToSingle(rangen.NextDouble());
            y1 = (Math.Sqrt((-2) * Math.Log(x1))) * (Math.Cos(2 * Math.PI * x2));
            y2 = (Math.Sqrt((-2) * Math.Log(x1))) * (Math.Sin(2 * Math.PI * x2));
            z1 = y1 * (double)sigmav;
            z2 = y2 * (double)sigmav;
               npt[n].X = (n);
               npt[n].Y = (float)z1;
               npt[n + 1].Y = (float)z2;
               noise[n].Y = npt[n].Y;
               noise[n + 1].Y = npt[n + 1].Y;
               //unit variance, zero-mean gaussian distribution of amplitudes
            }
         }
         set = true;

      }
     //The following loop adds the delayed input to the noise it also finds the Max value of the
delayed input
      // this is needed to calculate the S/N ration.
      for (int n = 0; n < (int)SAMPLES.max; n++)
      {
         npt[n].X = (n);
         npt[n].Y = noiseamp * noise[n].Y;   //scales amplitude of noise points
         if (n >= delay && i < duration)
         {
            //produces attenuated version of transmitted pulse
            decay = (ipt[i].Y * Convert.ToSingle(40000.00/ (delay * delay)));
            //and adds it to the noise
            npt[n].Y +=decay;
            if (maxdecay < decay)
               maxdecay = decay;
            i++;
         }

      }
      if (gaussian)  //Print s/n raion for gaussian distribution (A^2/2)/sigma^2
      {
         label7.Text = "Signal/Noise = " + Convert.ToString(Math.Round((maxdecay *
maxdecay / 2) / ((float)(numericUpDown4.Value * numericUpDown4.Value)),4));
      }
      else   //Print s/n ratio for uniform distribution (A^2/2)/(NoiseAmp^2/3)
      {
```

```
            label7.Text = "Signal/Noise = " + Convert.ToString(Math.Round((maxdecay *
    maxdecay / 2) /( ((float)(numericUpDown4.Value * numericUpDown4.Value))/3.0),4));

        }

        maxdecay = 0;  //set maxdecay to zero
        grfx.DrawLines(penPink, npt);                                   // draw the curve
    through the points

    }

//_____________________________________________________________________
    // Paints the panel containing the time correlation of the transmitted and received signals.
    private void panel3_Paint(object sender, PaintEventArgs e)
    {
        float temp = 0;
        int s = 0;
        Graphics grfx = e.Graphics;              // get graphics context
        panel3.Width = (int)(this.ClientSize.Width * .7); // Defines the Width
        panel3.Height = (int)(this.ClientSize.Height * .25); // Defines the Height
        cx = this.panel2.Size.Width;
        cy = this.panel2.Size.Height;
        // move origin to center of client area
        grfx.TranslateTransform(0, (float)cy / 2);
        // define brush and color for text
        SolidBrush txtBrush = new SolidBrush(Color.Crimson);
        // Scale coordinates to convenient units & invert y
        grfx.ScaleTransform((float)cx / (int)SAMPLES.max, -(float)cy / (int)SAMPLES.max);
        grfx.DrawLine(penBlack, 0, 0, (int)SAMPLES.max, 0);                     // x-axis
        grfx.DrawLine(penBlack, 0, ((int)SAMPLES.max/2), 0, -((int)SAMPLES.max/2));
              // y-axis
        // calculate noise
        PointF[] opt = new PointF[(int)SAMPLES.max];                    // array of points

        for (int k = 0; k <= ((int)SAMPLES.max-1); k++)
        {
            opt[k].X = k;
            for (int m = 0; m <= (((int)SAMPLES.max-1) - k); m++)
            {
                temp += ((npt[k + m].Y) * (ipt[m].Y))/(int)SAMPLES.max;
            }

            //find max value of  new array
            if(Math.Abs(temp) >= s) s = (int)Math.Abs(temp);

            opt[k].Y = temp;
```

```
        }
        //The next lines of code simple are used to automaticly set the limits on the graph
        label24.Text = Convert.ToString(-s);
        label23.Text = Convert.ToString(s);
        label21.Text = Convert.ToString((int)SAMPLES.max / 2);
        label20.Text = Convert.ToString((int)SAMPLES.max);
        s = s + 1;  // makes s = 1 as a minimum, avoiding potential divide by zero
        //the next for loop normalizes the correlation array and multiples these number by the
max size of the graph
        //this was done for plotting purposes to make sure that all of the graph could alway be
seen.
        for (int k = 0; k < ((int)SAMPLES.max-1); k++)
        {
           opt[k].Y = (opt[k].Y / s) * ((int)SAMPLES.max/2);
        }

        grfx.DrawLines(penRed, opt);                    // draw the curve through the points

    }

    // Increases or decreases the number of cycles of the simulated transmitted pulse
    // Refreshes the panels containing each plot
    private void numericUpDown1_ValueChanged(object sender, EventArgs e)
    {

        duration = (int)(numericUpDown1.Value*inputperiod);
        numericUpDown5.Minimum = duration / inputperiod;
        numericUpDown5.Maximum = ((int)SAMPLES.max - (duration)) / inputperiod;
        panel1.Refresh();
        panel2.Refresh();
        panel3.Refresh();

    }

    // Increases or decreases the amplitude of the simulated transmitted pulse
    // Refreshes the panels containing each plot
    private void numericUpDown2_ValueChanged(object sender, EventArgs e)
    {
        numericUpDown2.Maximum = (int)SAMPLES.max / 2;
        inputamp = (int)numericUpDown2.Value;
        panel1.Refresh();
        panel2.Refresh();
        panel3.Refresh();
    }

    // Increases or decreases the frequency of the simulated transmitted pulse
```

```
// Refreshes the panels containing each plot
private void numericUpDown3_ValueChanged(object sender, EventArgs e)
{
   inputperiod = (int)numericUpDown3.Value;
   duration = (int)(numericUpDown1.Value * inputperiod);
   numericUpDown5.Minimum = duration / inputperiod;
   numericUpDown5.Maximum = ((int)SAMPLES.max - (duration)) / inputperiod;

   panel1.Refresh();
   panel2.Refresh();
   panel3.Refresh();
}

// Increases or decreases the noise amplitude of the added noise pulse
// Refreshes the panels containing each plot
private void numericUpDown4_ValueChanged(object sender, EventArgs e)
{
   numericUpDown4.Maximum = (int)SAMPLES.max/2;
   noiseamp = (int)numericUpDown4.Value;
   panel1.Refresh();
   panel2.Refresh();
   panel3.Refresh();
   noiseAmplitude[step] = noiseamp;
   step++;
}

// Increases or decreases the input delay of the added noise pulse
// Refreshes the panels containing each plot
private void numericUpDown5_ValueChanged_1(object sender, EventArgs e)
{
   numericUpDown5.Minimum = duration/inputperiod;
   numericUpDown5.Maximum = ((int)SAMPLES.max - (duration))/inputperiod;
   delay = (int)numericUpDown5.Value*inputperiod;

   panel1.Refresh();
   panel2.Refresh();
   panel3.Refresh();
}

// ComboBox for the user to choose between a Uniform or Gaussian noise distribution
// Refreshes the panels containing each plot
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
   if (comboBox1.Text == "Gaussian")
   {
      set = false;
```

```
            gaussian = true;
            uniform = false;
            panel1.Refresh();
            panel2.Refresh();
            panel3.Refresh();

         }
         else
         {
            set = false;
            gaussian = false;
            uniform = true;
            panel1.Refresh();
            panel2.Refresh();
            panel3.Refresh();
         }
      }

      private void Form1_Load(object sender, EventArgs e)
      {
         // set some control start values
         numericUpDown4.Value = noiseamp;
         numericUpDown3.Value = inputperiod;
         numericUpDown2.Value = inputamp;
         numericUpDown1.Value = 10;              // default no. of cycles
         duration = (int)(numericUpDown1.Value * inputperiod);
         this.Invalidate();         // force repaint of client area on size change
      }

      // Displays the histogram of the noise (int)SAMPLES.max in an independent window
      private void button1_Click(object sender, EventArgs e)
      {
         Histogram histogramForm = new Histogram();
         histogramForm.Show();
      }

      private void button2_Click(object sender, EventArgs e)
      {
         set = false;
         panel1.Refresh();
         panel2.Refresh();
         panel3.Refresh();
      }
   }
}
```

```
//Histogram.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace project3
{
   public partial class Histogram : Form
   {
      public Histogram()
      {
         InitializeComponent();
      }

      Pen penRed = new Pen(Color.Red);              // define red pen
      Pen penBlue = new Pen(Color.Blue);            // define blue pen
      Pen penBlack = new Pen(Color.Black); // define black pen
      Pen penPink = new Pen(Color.HotPink);         // define pink pen

      SolidBrush brushblue = new SolidBrush(Color.Blue);

      int cx;              // integer variables to hold dimensions of client area
      int cy;              // in pixel

      // Paints the panel containing the histogram of the noise samples to examine the noise
sample distribution.
      private void panel1_Paint(object sender, PaintEventArgs e)
      {
         Graphics grfx = e.Graphics;             // get graphics context
         cx = this.panel1.Size.Width;
         cy = this.panel1.Size.Height;
         // move origin to center of client area
         grfx.TranslateTransform(0, cy);
         // define brush and color for text
         SolidBrush txtBrush = new SolidBrush(Color.Blue);
         // Scale coordinates to convenient units & invert y
         grfx.ScaleTransform((float)cx / 1000, -(float)cy / 200);

         float max = 0;
         float min = 0;
         float width = 0;
```

```
        if (Form1.gaussian)
        {
           max = 3;
           min = -3;
        }
        else
        {
           max = 1;
           min = -1;
        }
        width = max - min;

        label9.Text = Convert.ToString ( min);
        label11.Text = Convert.ToString (max);

        float sectionw = width / 40; // Splits the width into that 40 distinct sections
        int[] section = new int[40];
        for (int t = 0; t < 40; t++)
        {
           section[t] = 0;
        }

        for (int i = 0; i < 40; i++)
        {
           for (int t = 0; t < (int)SAMPLES.max; t++)
           {
              if ((Form1.noise[t].Y >= min) && (Form1.noise[t].Y < min + sectionw))
                 // Finds the range corresponding to each noise value

                 section[i] += 1; // Increase the amplitude of the section, for each value falling in
that range
           }
           min += sectionw;
        }

        for (int t = 0; t < 40; t++)

        {
           grfx.FillRectangle(brushblue, t * 25, 0, cx / 20, section[t]); // Fills the histogram bands
in blue
        }

     }

     private void Histogram_Load(object sender, EventArgs e)
```

```
        {
            this.Invalidate();          // force repaint of client area on size change
        }
        // Panels size and size change handling
        protected override void OnResize(EventArgs e)
        {
            this.Invalidate();          // force repaint of client area on size change
        }

    }

}
```