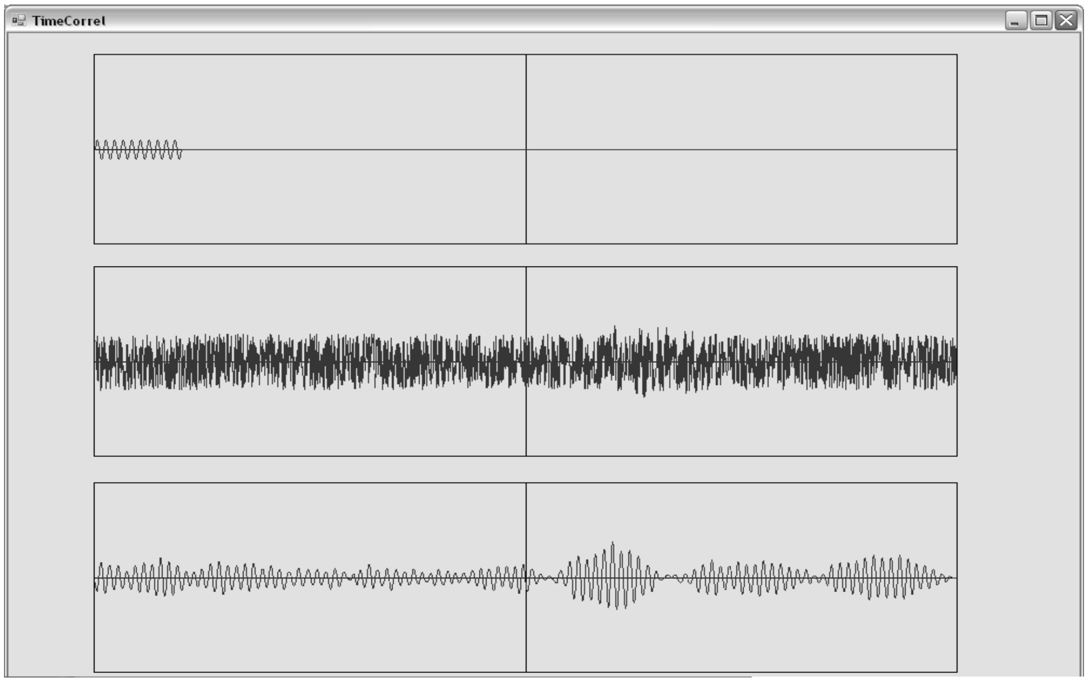
Correlation Detector Simulation

Some range detection applications use a system that sends out a pulse and waits for a return echo of the pulse. The time between the transmission of the original pulse and the reception of the echo may be used to compute the range of whatever reflected the pulse if the properties of the medium through which the pulse propagates are known. The return signal, however, has added noise, which may obscure the reflected pulse. One method of enhancing the received signal is time correlation of the transmitted pulse and the received signal.

For this project you are to write a Windows application that simulates a correlation detector. The results should be presented to the user in graphical form. The screen shot below illustrates what part of the graphical output might look like. There are three labels used to plot the results of a simulation for a particular choice of parameters.

Each of the three plots have x and y axis scales of -1000 to 1000. A coordinate transform was used in the program to make the plotting more convenient. The top plot shows a simulated transmitted pulse starting at x = -1000. Note that a scale of 0 to 2000 could just as well been used for x; it would have been a better choice. Your x scale should start at 0. The transmitted pulse plotted was $100\sin(2\pi\frac{n}{20})$ with n ranging from 0 to 199, producing about ten cycles.



The second plot shows the same pulse starting at x = 200 with added uniform noise. The noise was simulated as an array of points determined as

```
// calculate noise
     Random rangen = new Random();
     for (int n = 0; n < 2000; n++)
     {
        npt[n].X = (n - 1000);
        npt[n].Y = 300 * (2 * Convert.ToSingle(rangen.NextDouble()) - 1);
    }
```

The code snippet illustrates the use of one of the methods in the System.Random class. NextDouble() returns a double between 0 and 1. The part of the statement in parentheses produces random numbers between -1 and 1, and the multiplier of 300 is the amplitude of the noise in the units used for the plot. This noise is added to the delayed version of the transmitted pulse to produce the simulated received signal in the plot.

The third plot shows the time correlation of the transmitted and received signals computed as follows: $R(k) = \sum_{m=0}^{N-1-k} x_m y_{k+m}$ where x_m is a sample of the transmitted signal and y_{k+m} is a sample of the return signal. Note that although the noise amplitude was three times the transmitted pulse amplitude, the time correlation makes it much easier to determine the delay of the returned pulse. Both arrays x and y are of size N, and k ranges from 0 to N-1.

Your program should allow the user to choose the **duration of the transmitted pulse**, the **time delay** of the pulse in the received signal, and the **amplitude** and **type** of noise added to the delayed pulse to simulate the received signal. The example above used random noise uniformly distributed between -1 and +1 as the basic noise model, multiplied by an amplitude factor (300 in the code snippet shown). Your program should allow the user to select either the uniform noise model or a gaussian noise model. If the gaussian model is used, the program should allow the user to select the **standard deviation σ** as well.

The gaussian noise model for this program assumes that the noise at any point is a multiple (amplitude factor) of a zero-mean gaussian random variable with standard deviation σ. Sample values of the gaussian random variable may be generated from random numbers uniformly distributed between 0 and 1, such as returned by the Random.NextDouble() method in the code snippet above. The following method is described in *Numerical Recipes in C*, 2nd Edition.

> 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.

Zero-mean gaussian random sequences of stadard deviation σ may then be generated from y_1 and y_2 as $z_1 = \sigma y_1$ and $z_2 = \sigma y_2$.

The plots in your program should have appropriately labeled tic marks on the axes. The x axes should all start with zero at the left, while the y axes should have zero in the center of the vertical extent of the plot.

One more feature your program should include is the display of a histogram of the noise samples, so the user may examine the noise sample distribution. The histogram should be implemented as a method in a class of its own. The method should accept as parameters an array of points, and an array of bin centers. Output should be returned as an array of counts and bin centers. This method could then be used by future programs by including the class as a library. The arrays returned could then be used to produce a histogram. A histogram for the gaussian model should extend from -4σ to 4σ .