Signal Constellation Design Tool: A Case study in User Interface Synthesis

Gabriel Robins

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024, USA
gabriel@vaxb.isi.edu

Abstract

Signal constellation design is a major subtask of constructing an efficient communication system; it essentially entails trading-off error frequency against information throughput, a chief occupation of modem designers. We propose and implement an interactive tool for designing and simulating arbitrary signal constellations. 'To construct the user interface we have utilized Interface Builder, a new interactive tool that greatly facilitates the synthesis of arbitrary user interfaces through an object-oriented methodology. Using the Interface Builder package and the Signal Constellation Design Tool as the target prototype, we show how an order-of-magnitude improvement can be achieved in the effort required to produce a complex user interface. Our secondary goal is to try to dispel some of the mystique surrounding user interface synthesis on state-of-the-art workstations by describing in detail the construction of an interactive tool for computer-assisted learning.

Keywords: User interfaces, User interface tools, Human computer instruction, Man-machine interaction, Computer-assisted learning, Simulation tools, Object-oriented systems.

1. Introduction

Signal constellation design is a major part of constructing an efficient communication system. This task essentially entails trading off error frequency against information throughput, a chief occupation of modem designers. We propose and implement an interactive tool for designing and simulating arbitrary signal constellations. While the actual code that simulates signal constellations is rather trivial in itself, the user interface to this code is quite complex. To design and construct this user interface we have used Interface Builder, a new interactive tool that greatly facilitates the synthesis of user interfaces through an object-oriented methodology. Using the Interface Builder package and the Signal Constellation Design Tool as the target prototype, we show how an order-of-magnitude improvement can be achieved in the effort required to produce a complex user interface, and then draw some conclusions regarding the synthesis of user interfaces in general.

Our secondary goal is to try to dispel some of the mystique surrounding user interface synthesis on state-of-the-art workstations. Many otherwise informed researchers have very little experience in user-interface design, and consequently view user interface design as some sort of a black art, best left to specialized hackers to dabble in. By user interface design I mean a collection of functionality (running on a bit-mapped display workstation with a mouse) that interacts with the user in a friendly manner via menus, scroll bars, control buttons, icons, mouse clicks, and key strokes.

We intend to show that, quite to the contrary of these myths, given the proper tools and methodology, the synthesis of complex user interfaces could be rather trivial. As a case in point, the user interface described in this document was implemented on a MacIntosh, requiring only several days of coding, including the time to read the manuals and learn how to use the software. As a by-product of our inquiry, we have synthesized an interactive tool for computer-assisted learning.

LGE 1016

The first half of this document explains signal constellation design in general and how Interface Builder was used to synthesis the user interface; numerous examples and illustrations are given. The rest of this document describes and illustrates the functionality and usage of the resulting signal constellation design tool. The annotated Common LISP source code is available upon request both in hardcopy and on a MacIntosh diskette.

2. Signal Constellation Design

In designing an efficient communication scheme for band-limited channels, invariably of chief concern are the effects of noise and other kind of interference on the system [Forney, Gallager, Lang, Longstaff, and Qureshi]. To combat such interference, and while still aiming to achieve high throughput, one must carefully design an appropriate signal constellation [Carlyle] [Schwartz] [Sklar].

The task of signal constellation design essentially entails trading off error frequency against information throughput and is a chief occupation of modem designers. We propose and implement an interactive tool to alleviate the task of designing and simulating arbitrary signal constellations. We would like our tool to graphically display the signal constellation in two dimensions, allowing the user to visually observe the progressing simulation under interactive modifications to the interference parameters of the system.

2.1. The Desired Functionality

In this section we describe in more detail the functionality that we would like our Signal Constellation Design Tool to exhibit. Later we explain how this functionality was actually achieved in the implementation.

First, we would like to allow the user to select any of a number of "canned" standard signal constellations. For example, the user may elect to simulate an N-in-a-circle signal constellation and observe its performance under various levels of noise and distortion. Such selections should be done via mouse and menu interaction. Next the user may wish to select a certain probability distribution that would control the generation of random signal points. For example, the user may wish to select a Gaussian distribution with a specified variance.

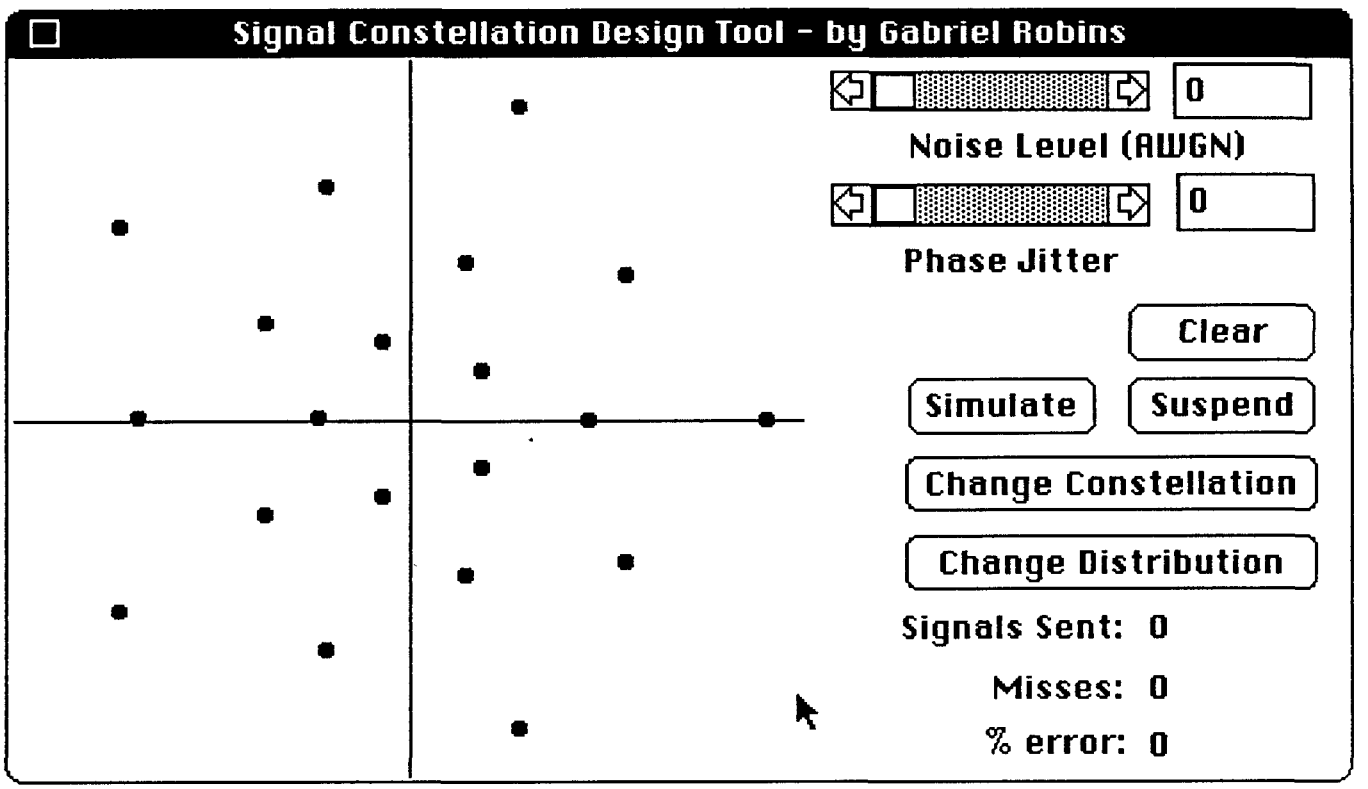
Once the user has selected a particular constellation to simulate/observe, as well as a probability distribution, that constellation should be drawn on the screen and the simulation may proceed. During the simulation, the user may interactively modify a number of system parameters, such as the phase jitter and the additive white Gaussian noise level. This would be accomplished by dragging "scroll-bars" identified with the corresponding parameters, or by directly typing in the desired values.

Using a random number generator, random signals are generated, according to the probability distribution function specified earlier, and are plotted on the signal constellation diagram. After a few minutes, a cumulative scatter-plot of the received signals will become apparent, giving the user an indication of how that signal constellation is performing under the distortion parameter values set previously. A cumulative running total of the number of errors encountered so far should be displayed, as is the empirically derived error-probability (the number of errors divided by the number of signals transmitted.)

The various commands should be also be accessible via clicking appropriate buttons, and alternatively also via menus and keystrokes. In addition, we wish to provide the user with some on-line help and information.

2.2. The Main Panel

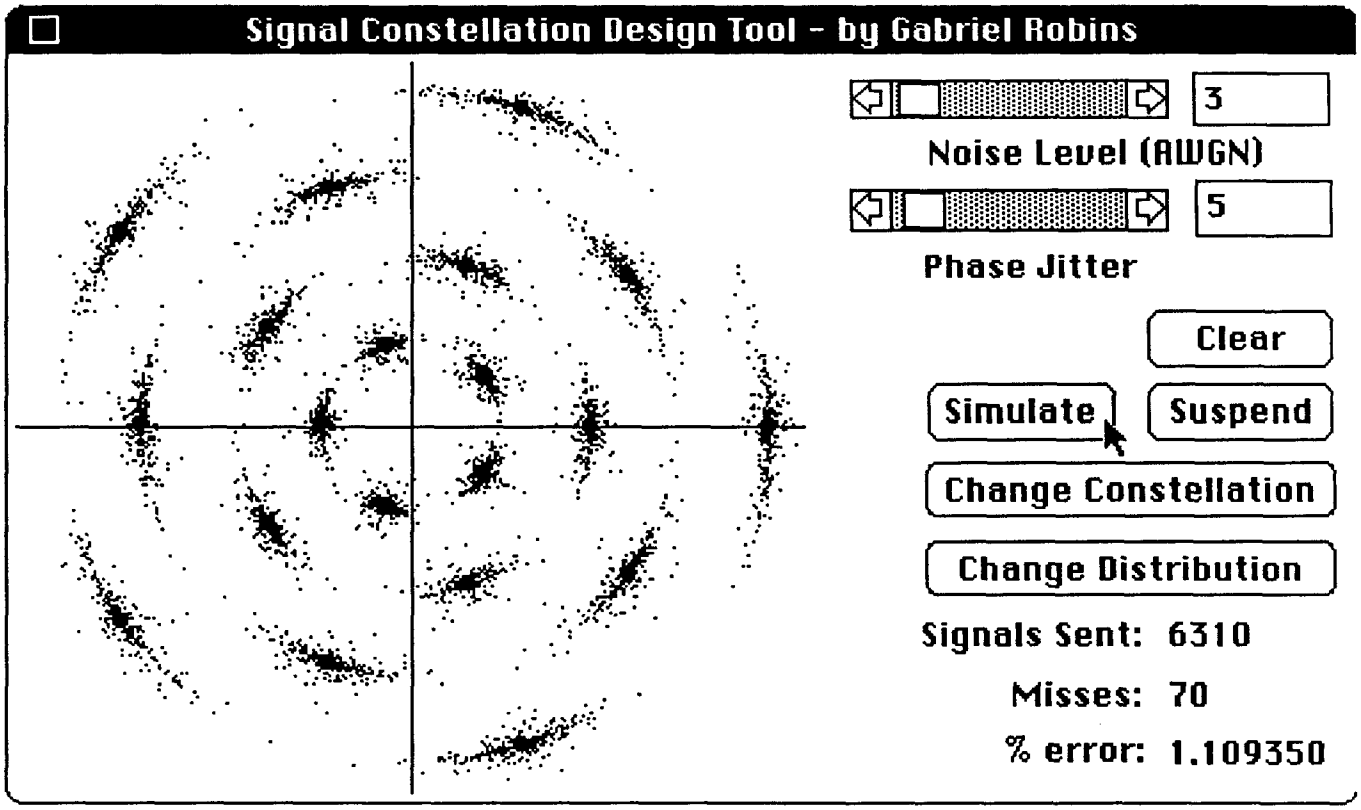
To make the appearance of the user interface more concrete, we give an illustration of how the main panel might appear:



To the left we see the main drawing area where the signal constellation appears; in this case the signal constellation itself consists of 20 points uniformly distributed on 4 concentric circles. At the top right we note the interference parameters, as well as the scroll bars and click boxes used to modify them. Below that we observe several "buttons" each of which will invoke a command if the user clicks it with the mouse. To the lower right we have the running statistics and error-ratio as the simulation progresses.

The user may invoke several operations simply by clicking the corresponding buttons. In addition, all of these commands are also available from the pull-down menus, as well as through keystrokes (i.e. single character keyboard inputs). We may also have at the top a pull-down menu bar, representing the various commands the user may invoke; the menu bar is not visible in this diagram.

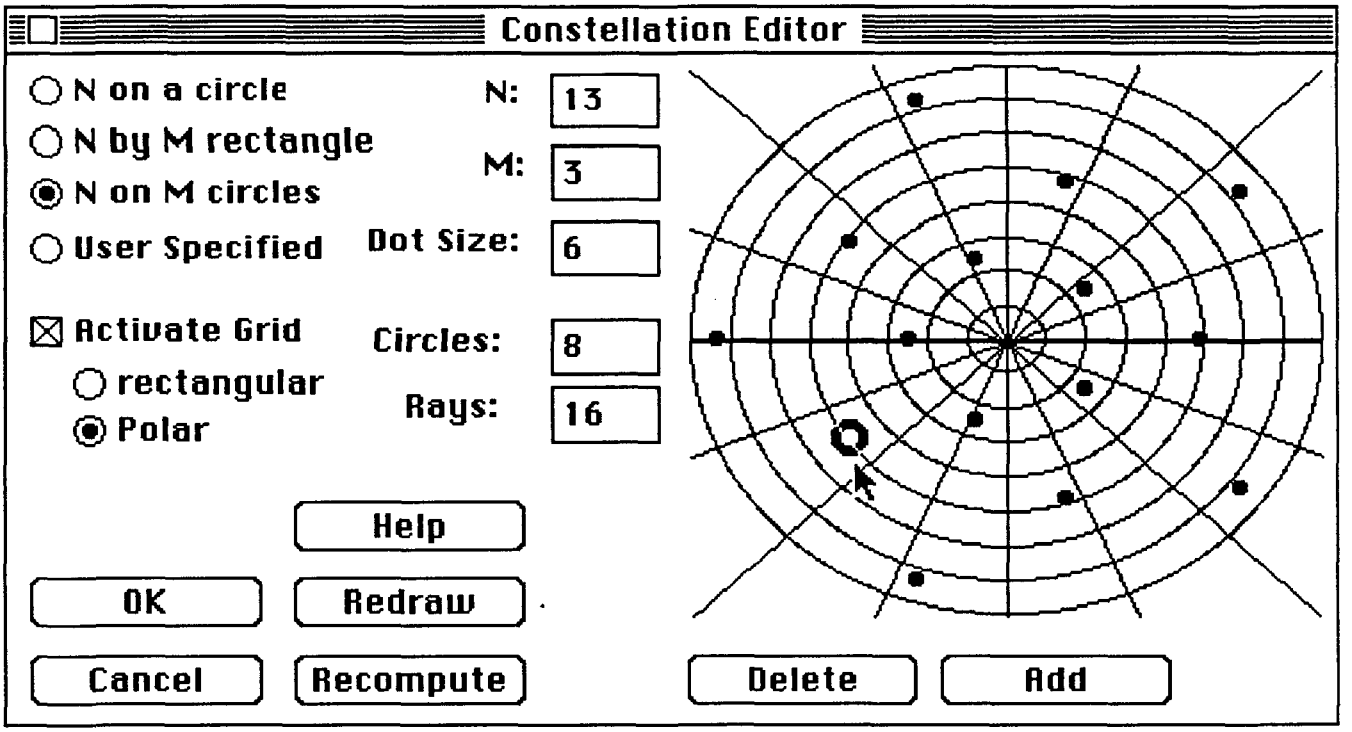
After a simulation has been underway for some time, the main panel might appear as follows:



The clouds around the signal points represent where the randomly generated signals fell around the actual signal constellation points. In this simulation, given the specified noise parameters, we are observing an error rate over over one percent, an undesirable situation.

2.3. The Constellation Editor

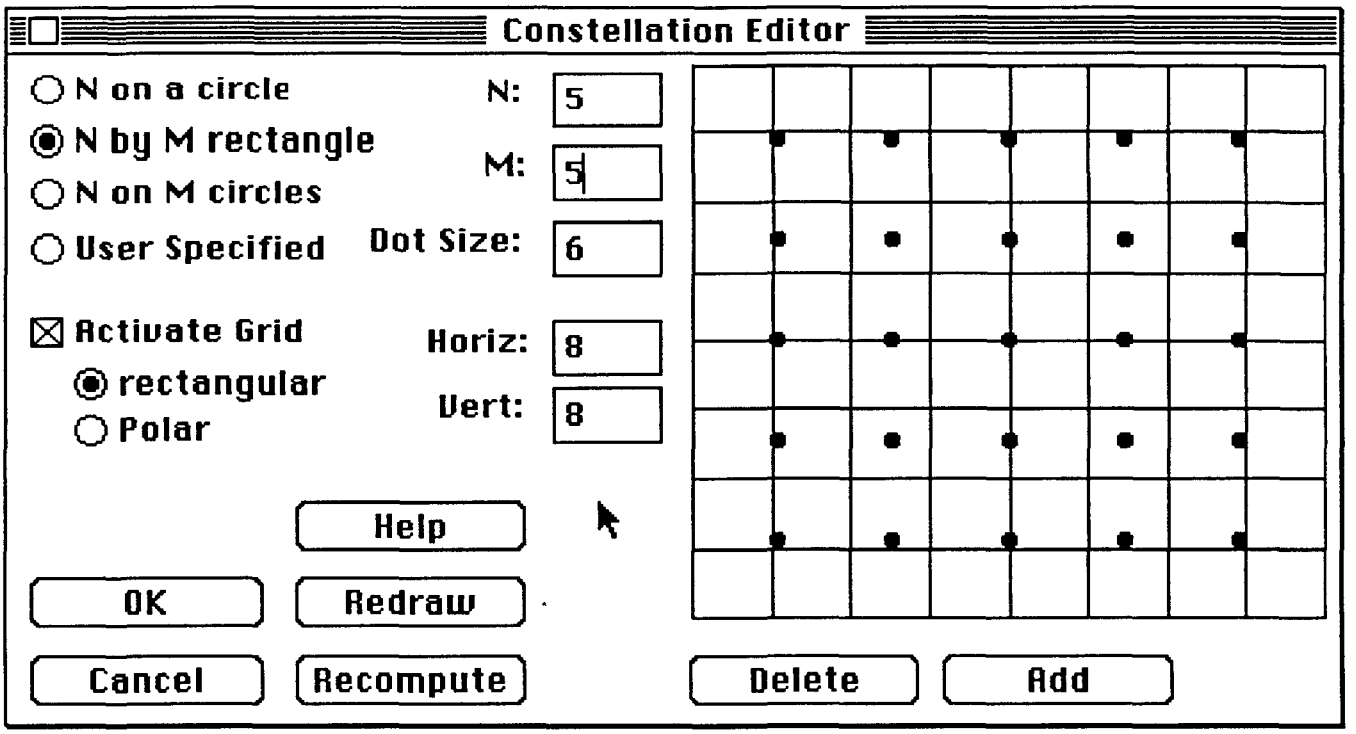
The panel that allows the user to select and edit a signal constellation is called the Constellation Editor and may appear as follows:



At the top left the user may select one of several "canned" signal constellations, parametrized by the variables M and N; these parameters are also user-specified: to change them, the user simply clicks in the corresponding box and types in the new value. An optional editing grid is available, and may be either rectangular or polar; the purpose of this grid is to make placement of individual constellation points more precise. The resolution of the grid may be controlled by the user; in the case of the rectangular grid, the number horizontal and vertical lines may be specified, and in the case of the polar grid, the number of circles and rays may be specified.

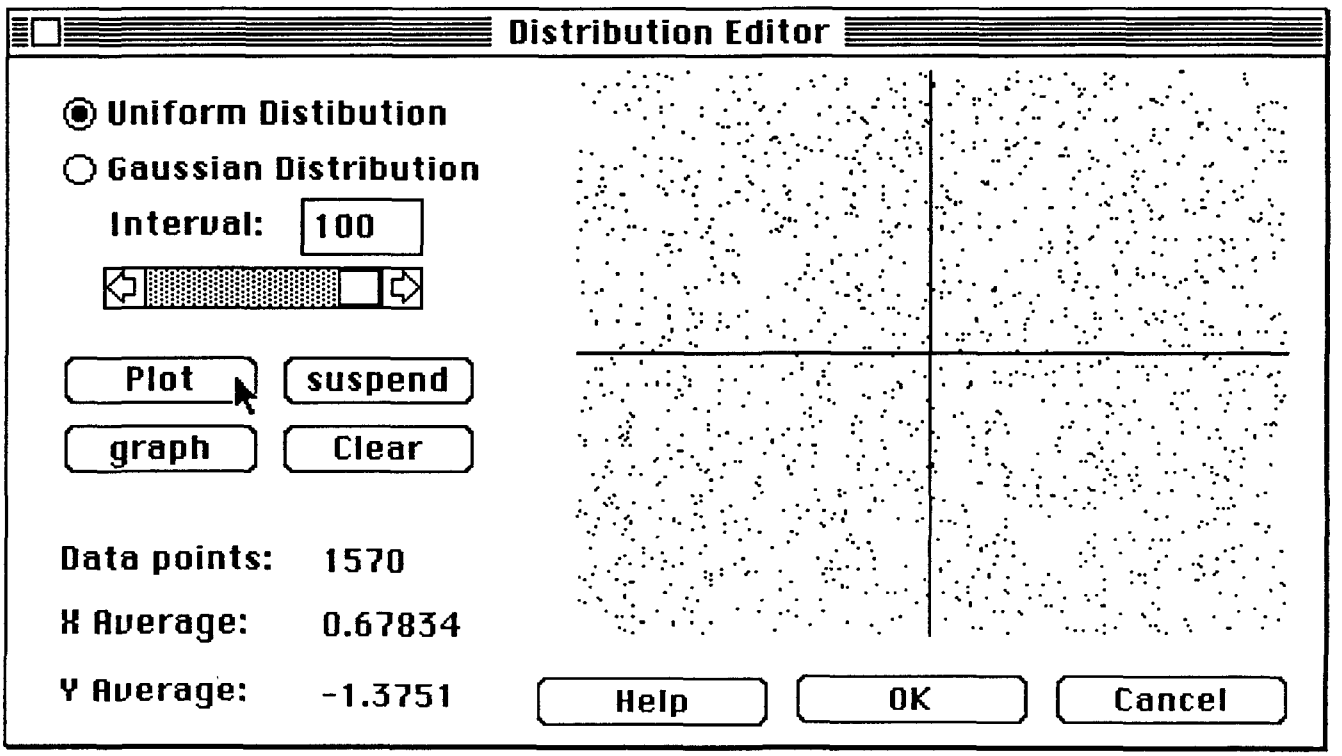
The user may add or delete constellation points, redraw the display, or obtain on-line help, simply by clicking the corresponding buttons. In addition, all of these commands are also available from the pull-down menus, as well as through keystrokes (i.e. single character keyboard inputs). Note that one of the points of the constellation is highlighted; this is accomplished when the mouse is clicked anywhere in the drawing area, whereupon the closest point to the click becomes highlighted. A "delete" command would subsequently remove the highlighted point, while an "add" command would wait for a new mouse click and a new point would be added to the constellation at the location of that click.

The on-line help consists of several screens of information and will be discussed later. "Ok" saves the current signal constellation and uses it from now on in all future calculations, while "Cancel" reverts back to the signal constellation previously in effect. Had the user selected a rectangular grid instead on a polar grid, the display might have appeared as follows:



2.4. The Distribution Editor

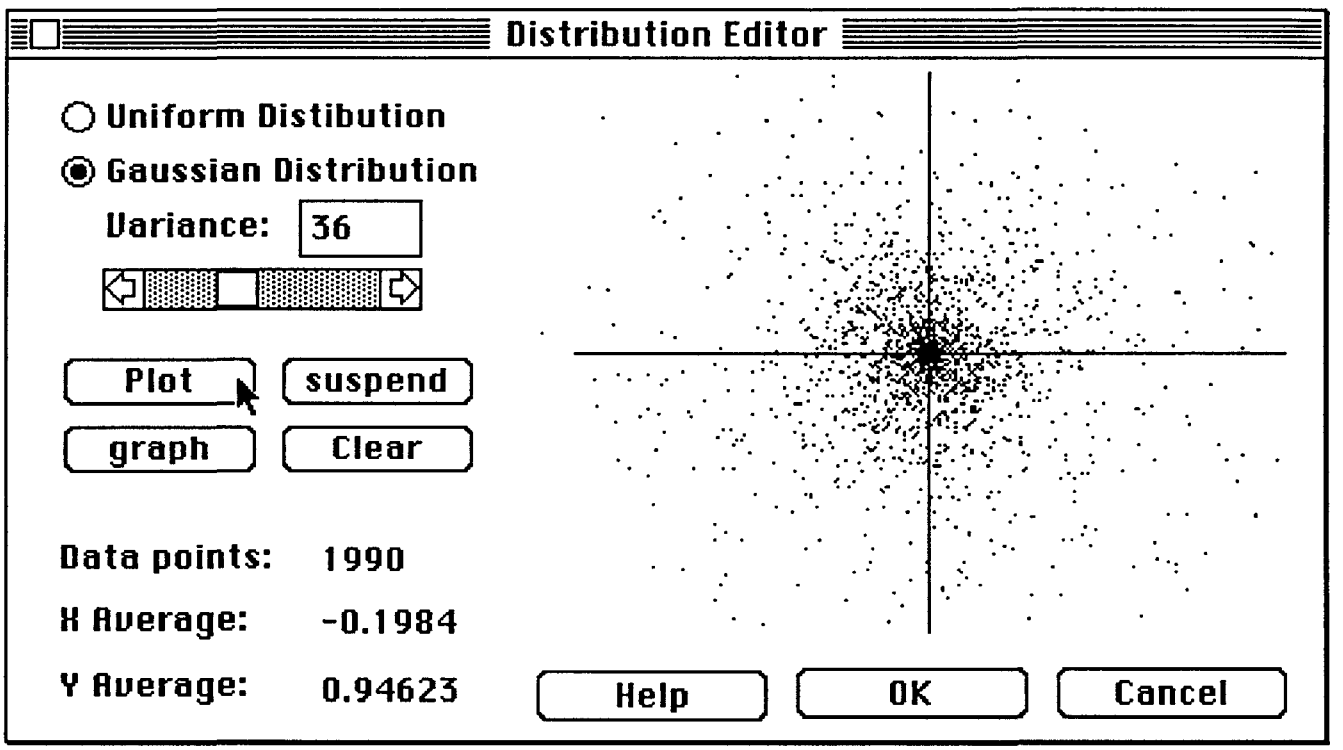
The panel that allows the user to select and edit a signal constellation is called the Distribution Editor and may appear as follows:



The user may select from either a uniform distribution on a given interval, or a Gaussian distribution with a given variance. The "Plot" command starts generating and plotting random points

according to the distribution specified by the user. The "Graph" command draws a graph of the probability density function in the X/Y plane. The "Help" command provides some on-line help/information, while the "Clear" command clears all the old points from the display. The average X and Y coordinates for the points generated so far are displayed to the lower left. "Ok" saves the current probability distribution and uses it from now on in all future calculations, while "Cancel" reverts back to the probability distribution previously in effect.

The following diagram illustrates a "Plot" of the Gaussian distribution:



By now the reader would agree that although simulating a given signal constellation may by itself indeed constitute a trivial programming task, the construction of a user interface that would behave as described above is by contrast quite a formidable programming task. In practical terms, the former could be easily accomplished in a couple of hours, while the later may take many weeks to construct. Using Interface Builder and an object-oriented programming methodology, all of these tasks were implemented on a MacIntosh in only several days of coding, including the time to read the manuals and learn how to use the software.

3. Using Interface Builder

The process of constructing the user interface using Interface Builder simply entails specifying inside an interactive environment the various menus, dialogue-boxes, scroll-bars, and menu-buttons, as well as where they should appear on the screen, and what should happened when each is clicked, selected, or dragged. The latter is accomplished by providing the relevant LISP code associated with each object. ExperInterface Builder performs all of the user-specified functions at the right times by usurping the workstation's "main event loop" and substituting the user-specified functionality as the default.

The result is an attractive user interface which is easy to build and modify. This tool was pioneered by ExperTelligence and recently has met with competition from Apple Inc., namely the HyperCard program [Goodman]. The importance of such tools has only recently been fully appreciated, although it has been known for quite some time that most of the effort associated with constructing computer software is invariably spent in programming the user interface; moreover, in

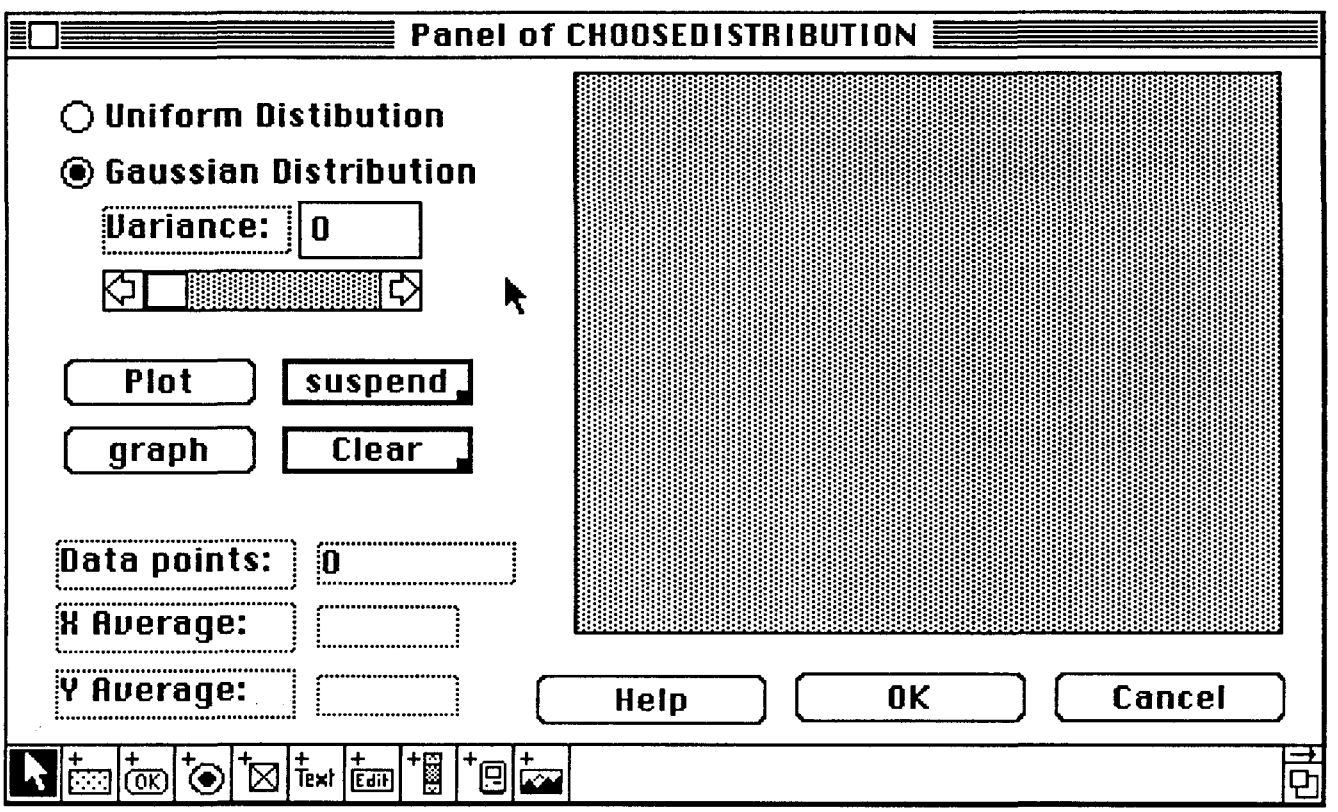
many cases the user interface directly determines the utility of a piece of software [Kaczmarek] [Robins].

3.1. The Methodology of Interface Builder

Interface Builder uses an object-oriented paradigm to create a user interface. Objects are rather general entities and may include windows, bitmaps, icons, records, scroll bars, buttons, text strings, regions, points, lines, files, and mouse clicks, among others. Objects communicate by sending *messages* to one another, and each object has a set of messages that it knows how to respond to; for example, a "redraw" message sent to an icon may cause the icon to redraw itself on the display. In addition to various useful default messages (or *methods*), a user may specify additional customized methods to be associated with an object. Messages may contain zero or more arguments and are essentially equivalent to function calls.

An Interface Builder *editor* is simply a panel consisting of a collection of objects, each with an associated set of methods. In addition to methods, an object may also have some local variables that may store arbitrary values, including other objects. When an object is defined it is specified as a child of some other object, and thus automatically inherits all the methods that apply to its parent; in addition, new methods may be added to the child, specializing it from its parent. An object may have multiple parents, in which case it inherits all of their methods. The astute reader will note that this schema necessitates a conflict-resolution or priority scheme when methods clash through inheritance, but we do not consider these details particularly relevant and therefore do not pursue them any further here.

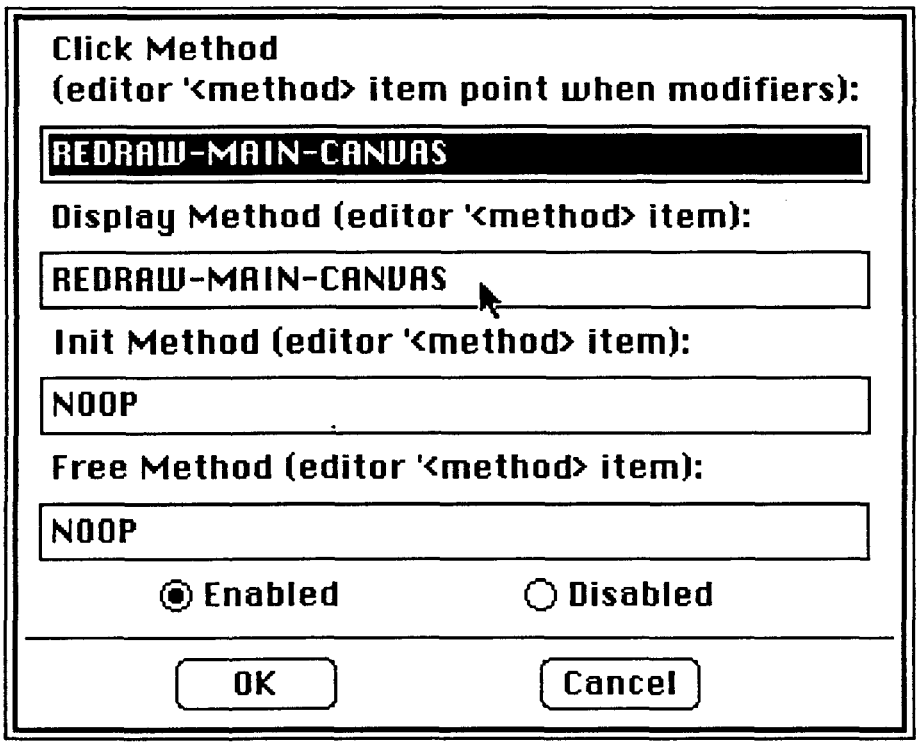
As a concrete example, let us consider the Distribution Editor described earlier, whose panel inside Interface Builder appears as follows:



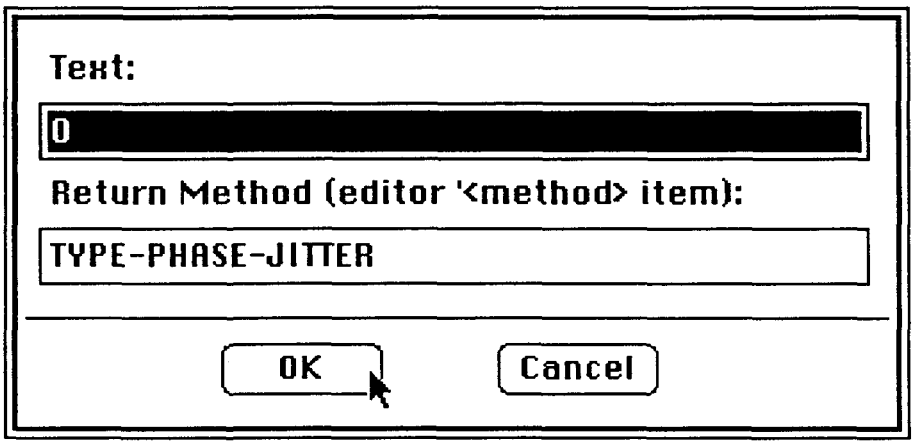
Each visible item is an object to which we may send various messages, and with which there is associated functionality that is invoked whenever during execution it is clicked, dragged, resized, etc. The icons at the lower left side are Interface Builder commands and are used to create the various types of objects that they represent pictorially. Once such an object is created, it may be further

modified, resized, and redefined.

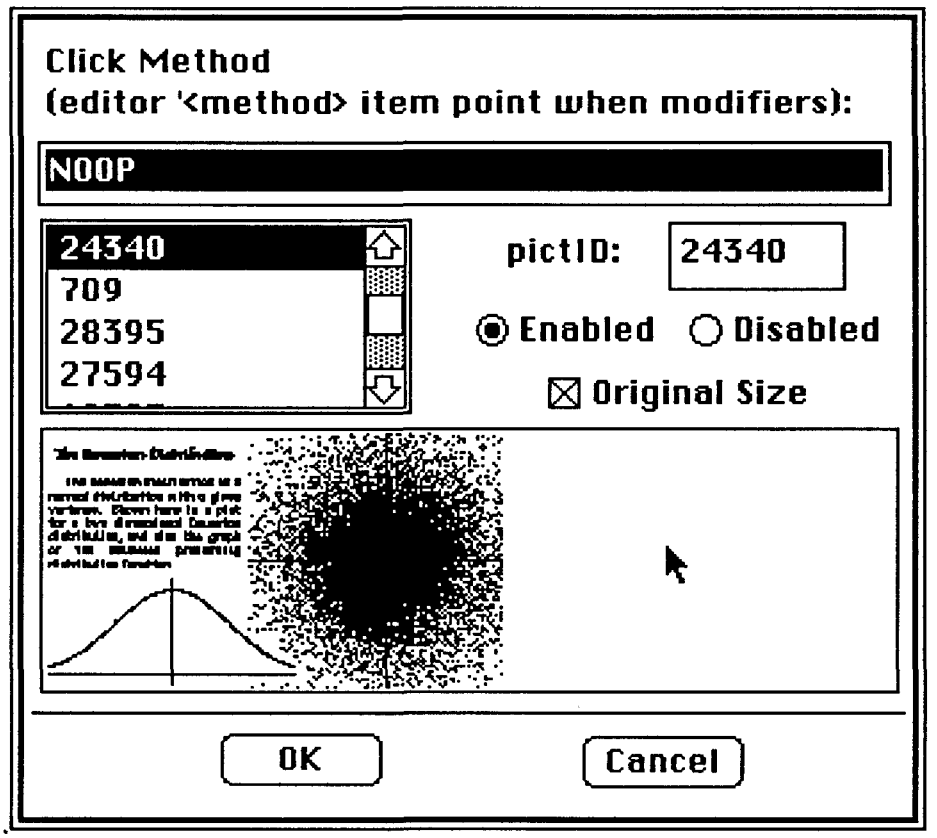
For example, the "Click" method of the main drawing area to the right may be specified by double-clicking on the main drawing area and filling in the required fields in the resulting dialogue panel as follows:



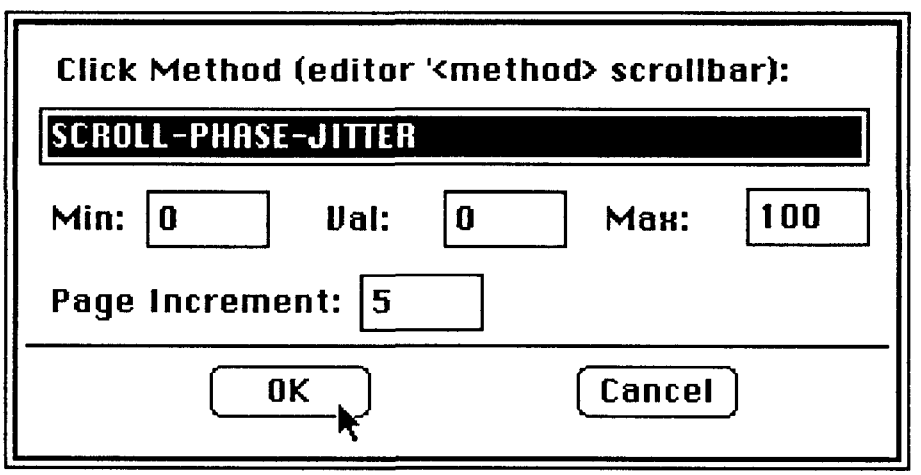
The function REDRAW-MAIN-CANVAS is a piece of code that will clear out and redraw that area. Similarly a text item may be specified to have a certain "click" method by creating it, double-clicking it, and filling in the appropriate fields in the resulting dialogue panel, as follows:



Arbitrary icons and bitmaps may also be included as part of the editor panel being constructed. This is done by a dialogue as in the following example, in which a previously created bitmap is designated as part of the display of the forth Distribution Editor help screen:

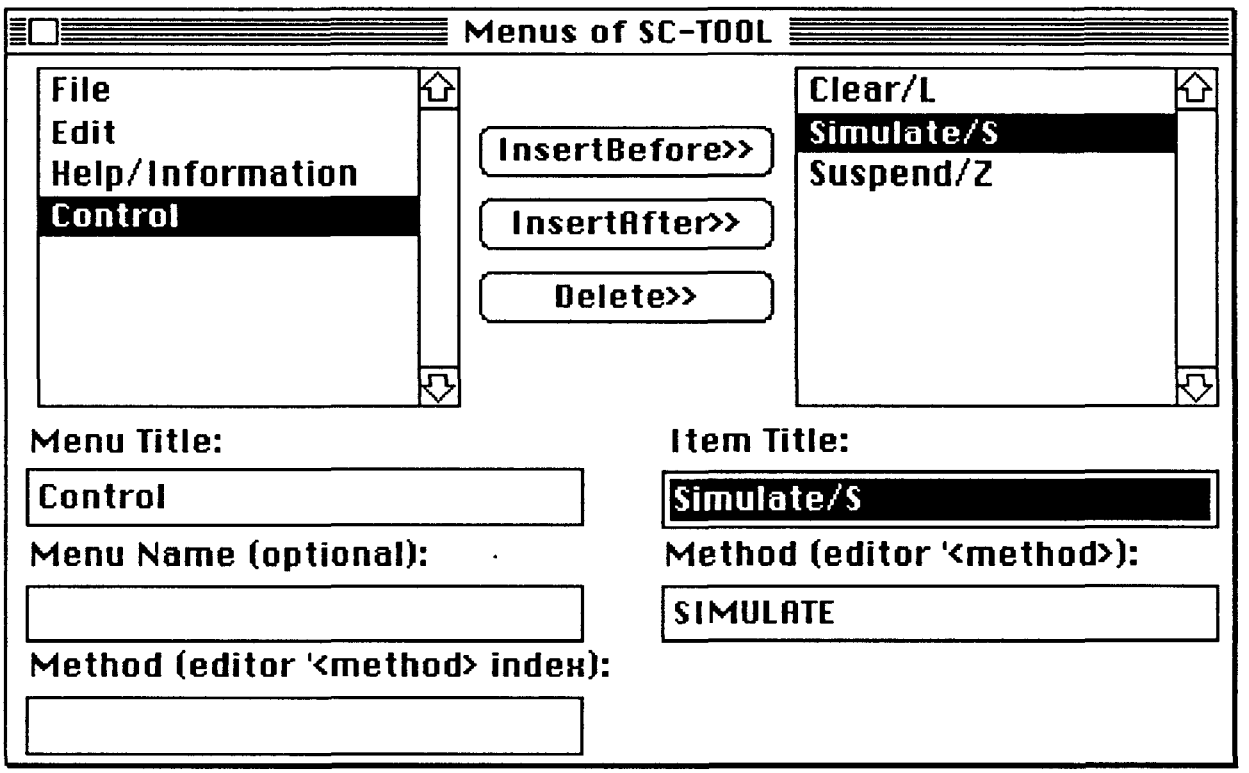


Scroll bars of arbitrary sizes may be similarly created and placed in arbitrary locations by specifying the appropriate "click" method, as well as minimum and maximum values for the scroll interval:

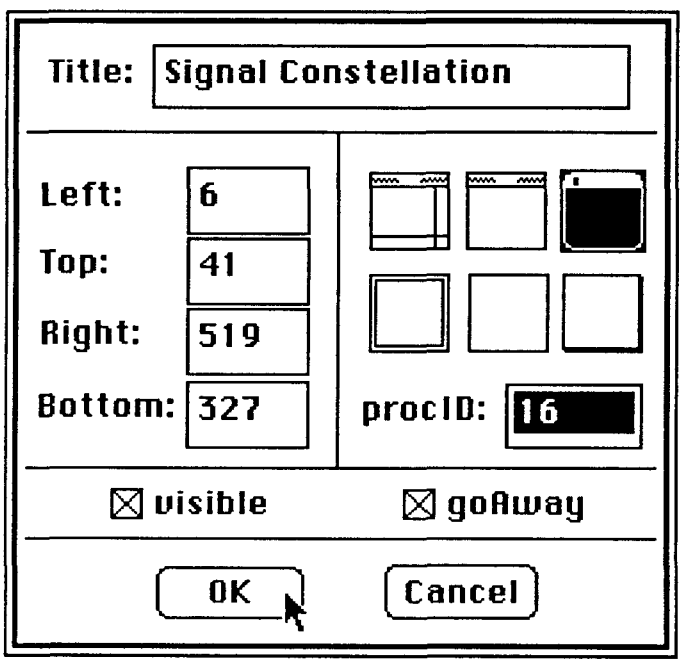


Each editor panel has associated with it a pull-down menu bar containing several menus, each containing several menu items. A menu item is an entry in a menu that when selected causes some code to be executed. Menus are also constructed interactively in Interface Builder. For each named menu entry the user specifies a function to be called when that entry is selected. In addition the user may optionally specify a keystroke (denoted by a slash and a letter) that will execute the same functionality *without* having to go through the menu system. This is useful to experienced users who would find it easier to memorize a keystroke rather than waste a longer time pulling down and clicking a menu item. The following example illustrates the process of defining a menu, and is part of

the main panel of the Signal Constellation Design Tool:

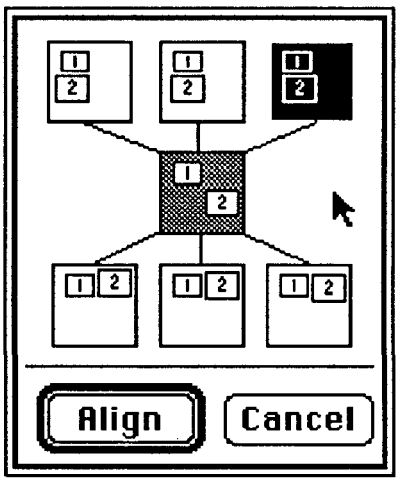


The shape and characteristics of the panel window may itself be modified; a window may be optionally movable, scrollable, resizable, closable, have a title, etc. Here is an example of how these attributes are interactively specified via a dialogue with Interface Builder:

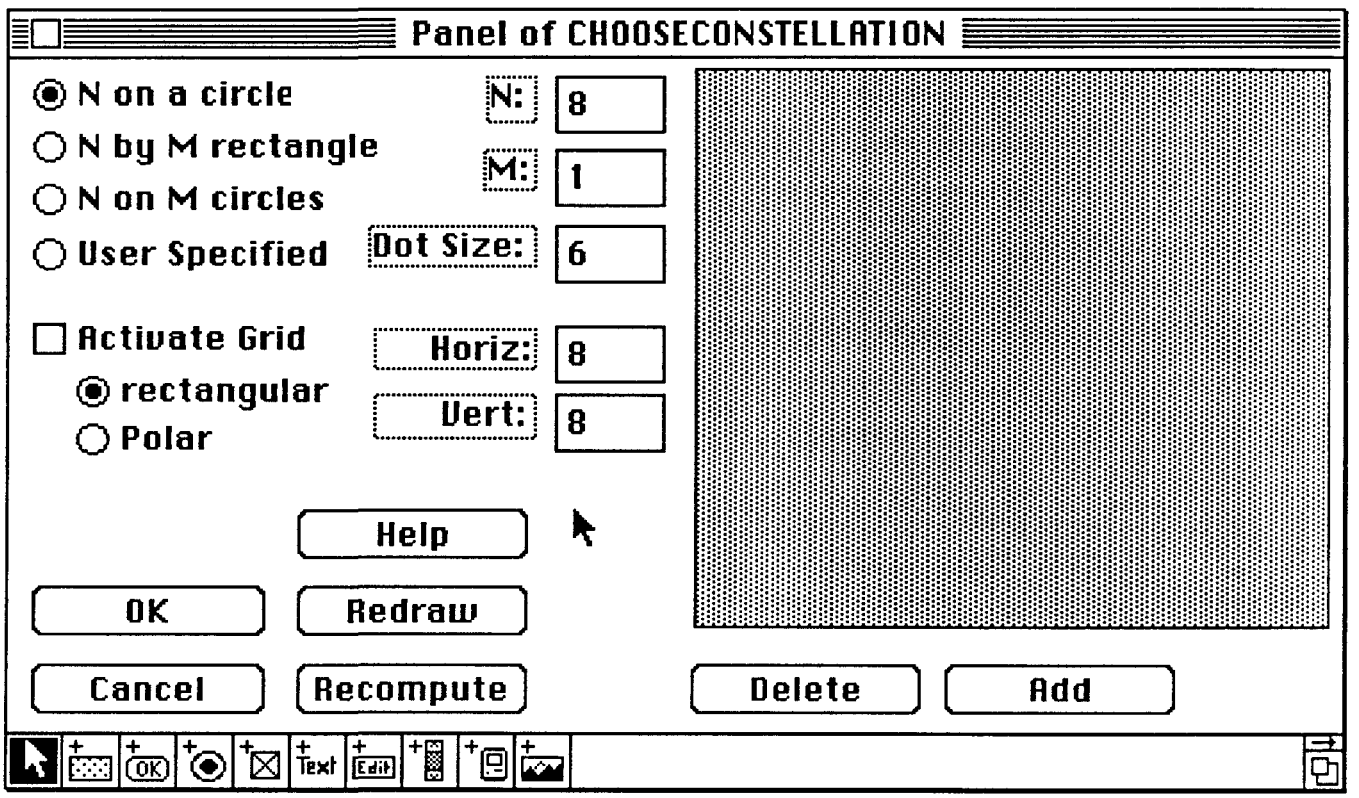


To make the placement of fields more precise, Interface Builder provides a facility for aligning fields and also making groups of fields the same size. Such alignment/resizing helps to make the resulting panel more uniform in appearance; the alignment command is invoked by clicking on the appropriate

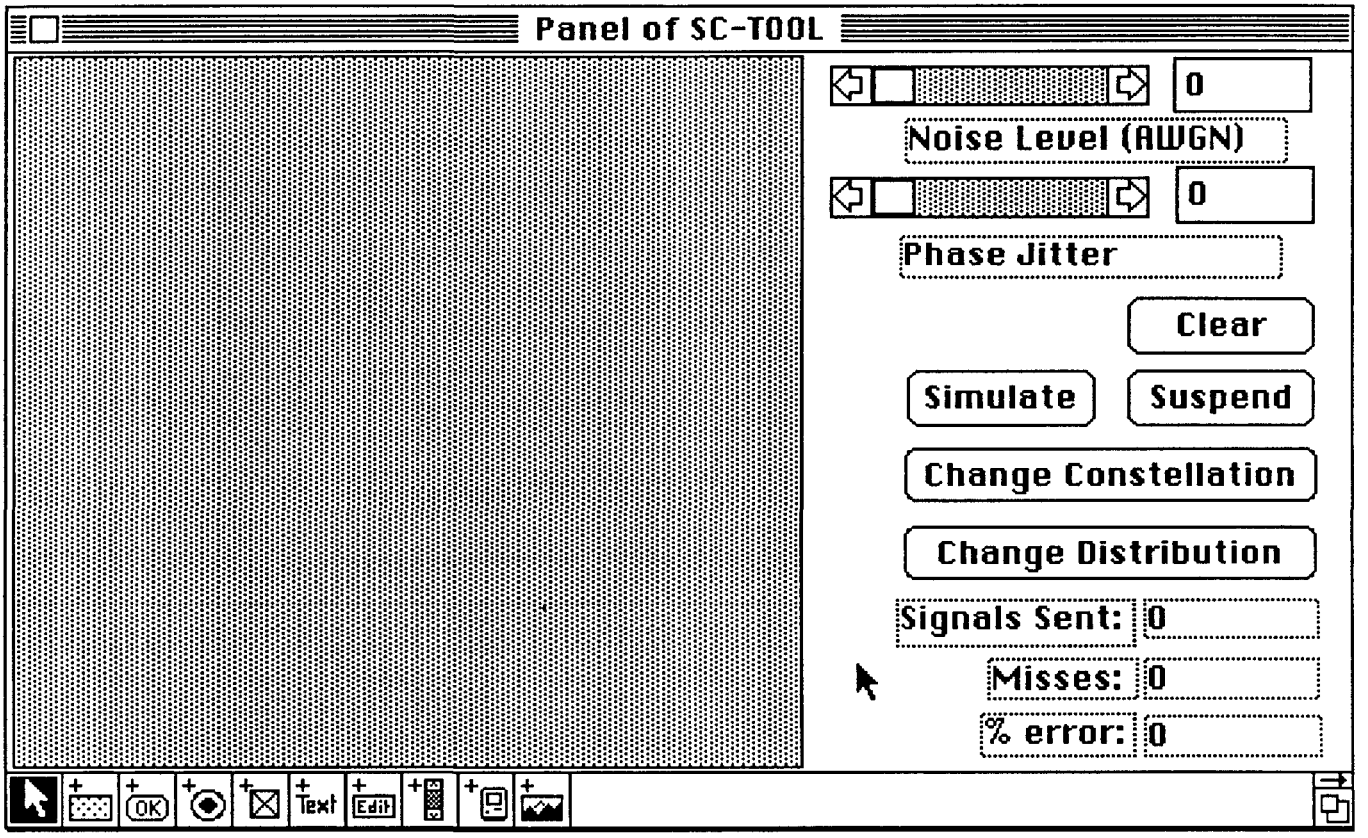
icon in the following dialogue panel:



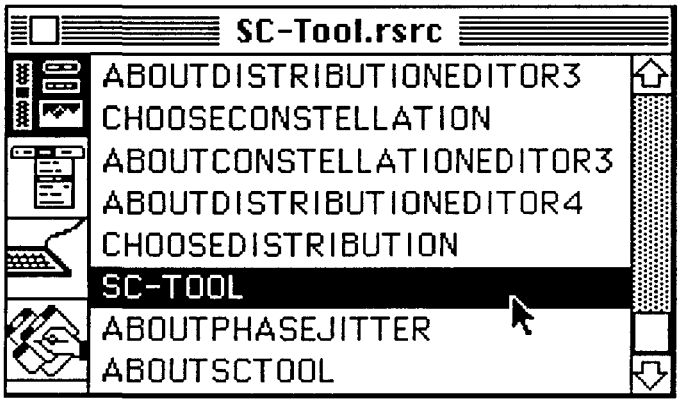
Proceeding in this manner we then construct inside Interface Builder the panel for the Constellation Editor, which appears as follows:



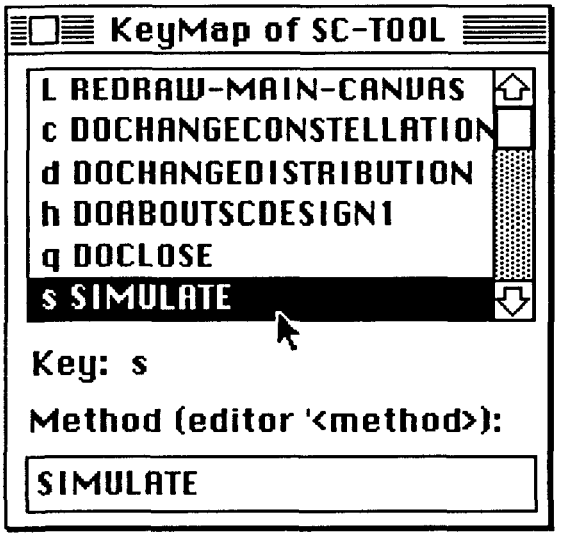
Finally we construct inside Interface Builder the main panel for the Signal Constellation Design Tool, which appears as follows:



After several editor panels have been constructed, we obtain the following Interface Builder display window, where every line corresponds to an editor panel. The icons on the left (in top to bottom order) correspond to control item editing, menu editing, keymap editing, and subeditor editing, respectively:



Control item and menu editing has been discussed previously. Keymap editing entails binding various functionality to keyboard keys. If a function F is bound to a keyboard key K, then during execution, whenever the key K is pressed, function F gets called. This provides an easy means to quickly invoke certain user-defined commands and functions. A common practice is to bind certain keys to important menu items in order to save experienced users the time to pull down a menu; instead, only a single key needs to be pressed. The dialogue panel in which such key bindings are specified in Interface Builder is given here:



4. Correctness and Functional Orthogonality

Since the underlying paradigm of Interface Builder is object-oriented in nature, a certain functional orthogonality exists in the finished software in the following sense. Messages sent to an object do not directly affect any other object, and moreover objects can only communicate by passing "messages" to one another (actually there is another way for objects to communicate, namely by assignment/reading of global variables, but this practice is not encouraged). This implies that flow of control is highly constrained and therefore the formation of side-effects, although possible, is nevertheless tightly controlled.

If a set of objects has been created and debugged and is found to operate correctly, adding new objects is not likely to affect any of the old objects or the correctness of their behavior. Moreover, the functionality of any of the objects may be invoked at any point in time via an appropriate message from any other object. Although at first glance this would seem to give rise to a certain "non-determinism" in execution, in practice, the programmer will be very informed about what code should/would execute under various circumstances, and my experience has shown that if the programmer has adhered to the standard object-oriented programming conventions, the "right thing" usually happens under even the most pathological circumstances.

The programmer's code does not have to worry about a "main-event-loop" and about dispatching certain pieces of code depending upon what event has transpired, because Interface Builder usurps the system's "main-event-loop" already and does all the necessary dispatching based on the programmer's specifications. This takes much of the complexity out of the application code, complexity that would otherwise have had to be duplicated from scratch in each application. Thus considerable programmer effort is saved by this scheme.

5. The On-Line Help Screens

As part of our user interface design, we provide a mechanism for presenting some interactive on-line help to the user. This help may be invoked via clicking a button, pulling a menu, or pressing a key. The help itself consists of one of more screens full of information, directions, and diagrams. The user may jump between these screens, or quit and return to the original mode before calling the help. Each screen (except the first) contains a "Previous" command button, that will expose and activate the screen that immediately precedes the current one in the logical continuation of the help. In addition, each screen (except the last) contains a "Next" command button, that will expose and activate the screen that immediately succeeds the current one in the logical continuation of the help sequence.

Having some on-line help is essential in many applications and often saves considerable (manual look-up) time for the user, especially if the help is also crossed referenced or indexed in some manner. A good example of useful on-line help facilities is contained in the MicroSoft Word 3.01 text processing program.

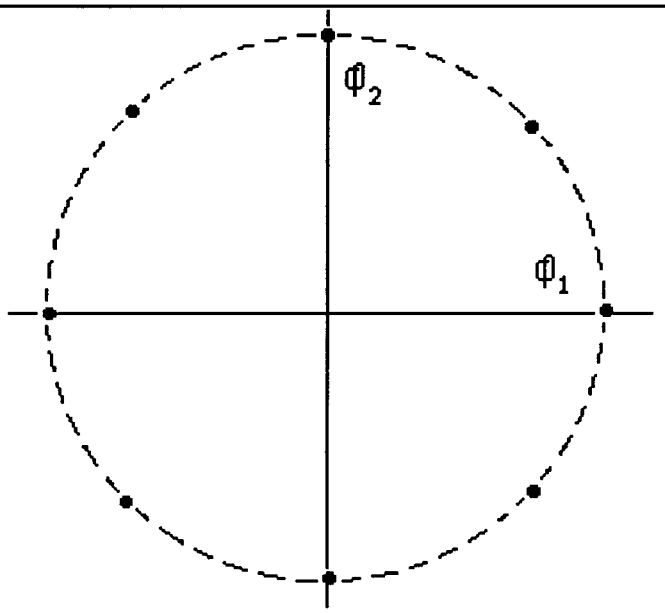
5.1. On-Line Help Screens for Signal Constellation Design

As an example of this discussion we depict here some of the help screens included in the user interface, beginning with the two on-line help screens for the "About signal constellation design" item:

Signal Constellation Design

Signal constellation design entails specifying a set of signal points in the plane in such a way as to minimize the probability of error during signal transmission subject to given interference parameters such as additive white Gaussian noise and phase jitter.

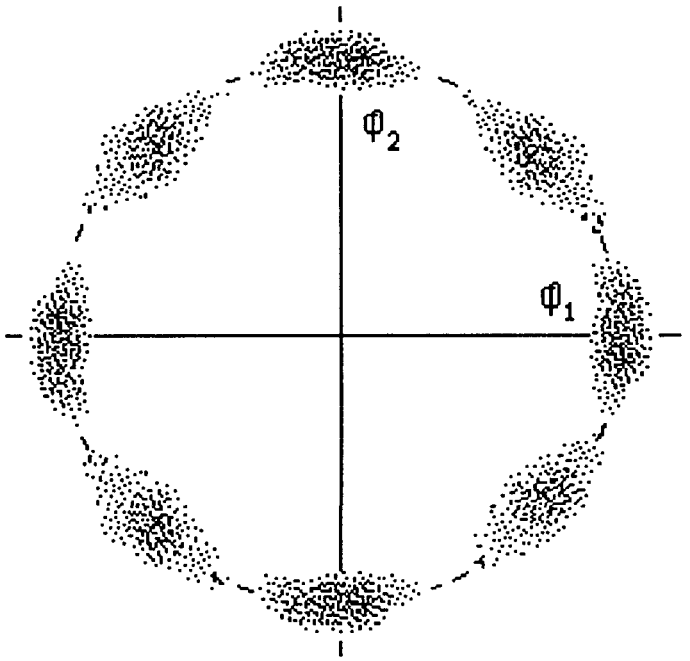
To the right is a typical signal constellation; this example consists of 8 signal points uniformly spaced around a unit circle.



Next Quit

Once the user has specified the signal constellation and set the values of the interference parameters, the simulation of this signal constellation begins: random signals are generated with added distortion produced in accordance with the proper probability distribution, and the resulting received signals are plotted against the original signals.

A record is kept of the error rate observed. After a while, the picture for our example may appear as in the diagram on the right:



Previous Quit

5.2. On-Line Help Screens for Additive White Gaussian Noise

Here are the three on-line help screens for the "additive white Gaussian noise" item:

Additive white Gaussian noise (AWGN)

The term *noise* refers to unwanted electrical signals that are superimposed on the transmitted signal and tend to obscure it; it limits the receivers ability to make correct symbol decisions, and thereby limits the rate of information transmission.

Usually we model noise as being of the additive white Gaussian type; that is, as a random process whose value n(t) at time t is statistically characterized by the Gaussian density function p(n):

$$p(n) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{n}{\sigma}\right)^2 \right)$$

where σ^2 is the variance and the mean is zero.

Next Quit

The *central limit theorem* of statistics states that under very general conditions the probability distribution of a sum of k statistically independent random variables approaches the Gaussian distribution as k approaches infinity, no matter what the individual distribution functions may be.

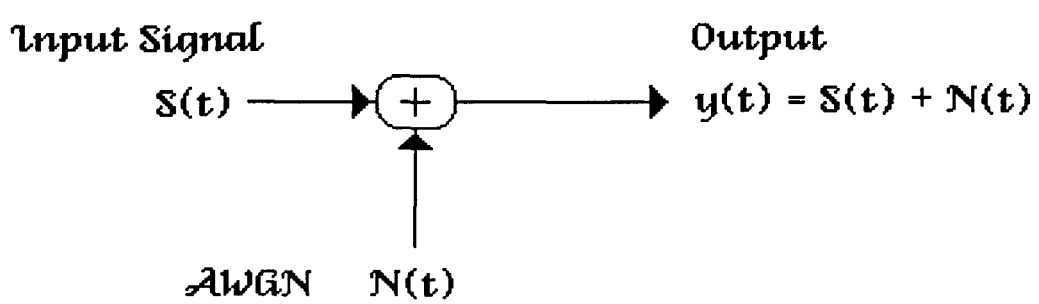
Therefore, even though individual noise mechanisms might have other than Gaussian distributions, the aggregate of many such mechanisms will tend toward the Gaussian distribution. We are therefore justified in modelling noise in our system using the Gaussian distribution.

The term *white* in AWGN refers to the fact that the power spectral density of thermal noise is the same for all frequencies of interest in most communication systems; that is, a thermal noise source emanates an equal amount of noise power per unit bandwidth at all frequencies.

Previous | Next | Quit

The term *additive* refers to the fact that the noise is added to, or superimposed on the signal during transmission; there are no multiplicative mechanisms at work. The noise affects each transmitted signal independently, and a communication channel of this nature is called a *memoryless* channel.

Diagrammatically, the situation appears as follows:



Previous | Quit

Note that such help screens may contain both text, equations, graphics, and active control objects such as buttons and scroll bars; the functionality of the latter is completely user-controlled, as is the general layout, placement, and appearance of these items.

5.3. On-Line Help Screens for Distribution Editor

Here are the four on-line help screens for the Distribution Editor:

The Probability Distribution Editor

○ Uniform Distibution
◉ Gaussian Distribution
Variance: 46

◉ Uniform Distibution
○ Gaussian Distribution
Interval: 54

The Distribution Editor allows the user to select and inspect a particular probability distribution, which will in turn determine the likelihood of certain points being selected for the various simulation parameters such as noise and phase jitter.

If a uniform distribution is selected, the user may specify the range from which values will be selected uniformly. If a Gaussian distribution is selected, the user may specify its variance.

Next Quit

Distribution Editor Commands

Plot
suspend

graph
Clear

Data points: 6170
X average: -1.0332
Y Average: 0.43128

Plot will randomly select and display points in the plane according to the specified distribution, while **Suspend** will halt the plotting process.

Graph will produce a graph of the selected distribution.

Clear will redraw the coordinate axis and erase all points previously plotted/graphed.

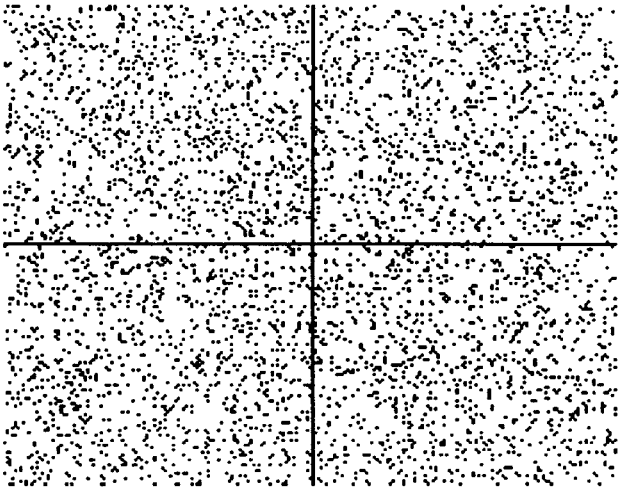
A running total is kept of the number of points plotted so far, as well as their average x and y coordinates.

Previous Next Quit

The Uniform Distribution

A uniform distribution is simply a random selection with equal probability of a point from a given interval. Shown here to the right is a two dimensional uniform distribution.

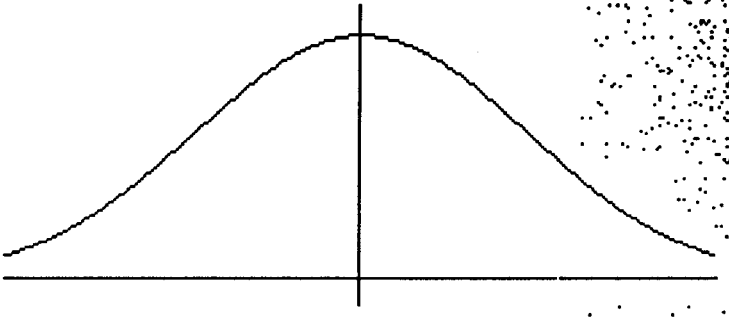
In such a distribution both the X and the Y components are chosen both uniformly and independently.

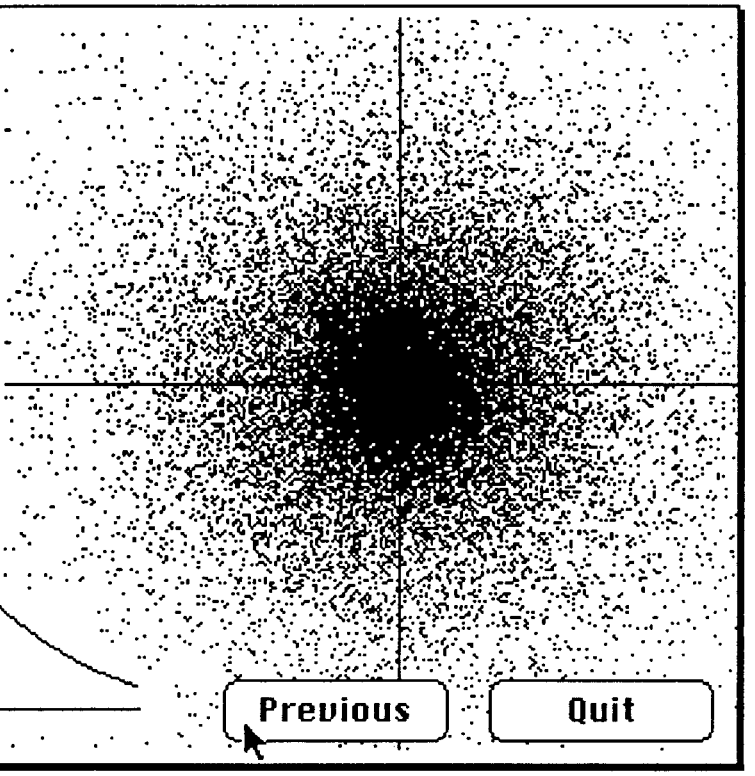


Previous Next Quit

The Gaussian Distribution

The Gaussian distribution is a normal distribution with a given variance. Shown here is a plot for a two-dimensional Gaussian distribution, and also the graph of the Gaussian probability distribution function.





Previous Quit

5.4. On-Line Help Screens for Constellation Editor

Here are the three on-line help screens for the Constellation Editor:

The Constellation Editor

○ N on a circle
○ N by M rectangle
◉ N on M circles
○ User Specified

N: 13
M: 2
Dot Size: 6

The Constellation Editor allows the user to select one of several canned/standard signal constellations, or specify an arbitrary one.

The parameters N and M are also user specified and allow considerable flexibility in parametrizing the standard signal constellations. The dot-size determines how large the dots (representing the signals) will be on the display.

Next Quit

Constellation Editor Commands

Redraw will clear and redraw the current signal constellation.

Recompute will recompute and redraw the current signal constellation.

Delete will remove the highlighted signal from the constellation.

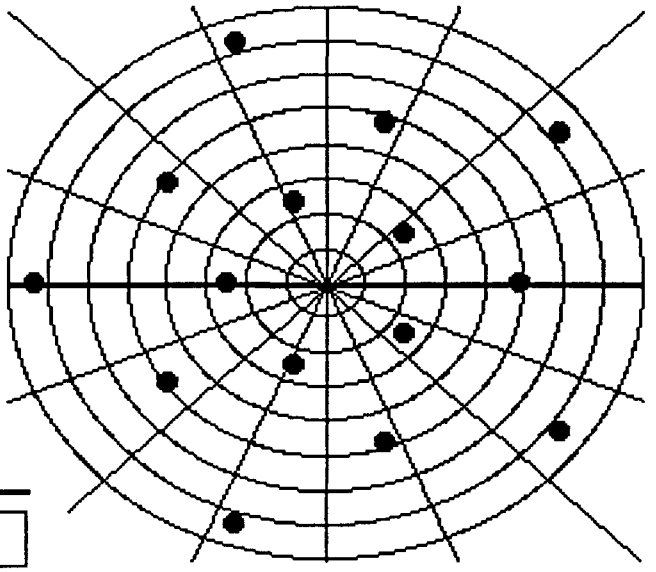
Add will add a signal to the signal constellation, in the location specified by the next mouse-click.

Clicking on the signal constellation itself will highlight the signal closest to the position of the mouse-click. Most of the commands are also available through the menus, as well as through key-strokes.

Previous Next Quit

The Editing Grid

For convenience the user may turn on an editing grid in either polar or rectangular coordinates. This should make the placement of signal dots more precise. The user also may control the resolution of the grid. To the right is an example of a signal constellation embedded in a polar grid.



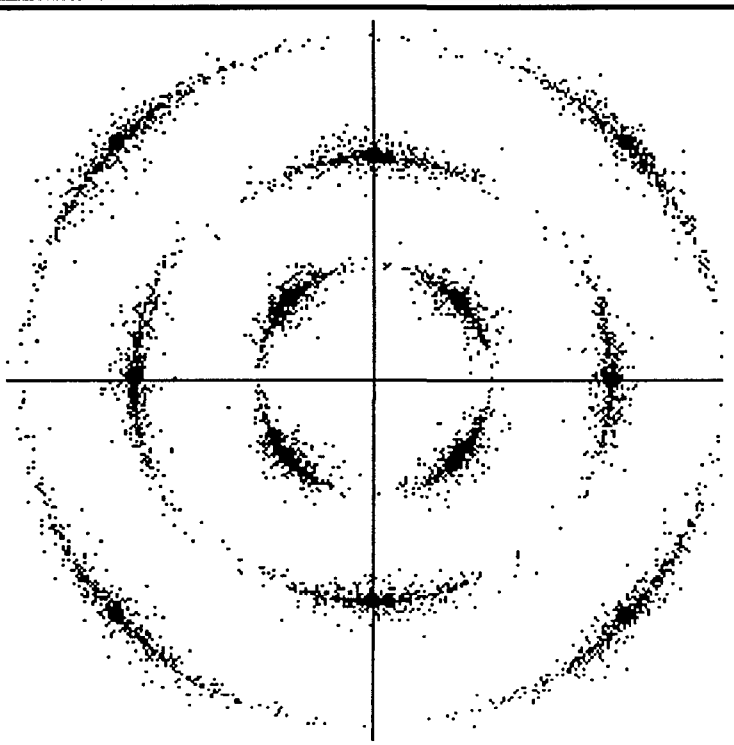
☒ Activate Grid
○ rectangular
◉ Polar

Circles: 8
Rays: 16

Previous Quit

5.5. On-Line Help Screen for Phase Jitter

Here is the on-line help screen for the "phase jitter" item (note that when there is only one screen-full of help, there are no "Next" or "Previous" command buttons):



Phase Jitter

Phase jitter is a type of noise that affects the phase angle of the transmitted signal. Phase jitter is superimposed (added) to the other noises affecting the signal, and may be set to some user-specified level independently of the other parameters. To the left is an example of a simulation with considerable phase jitter (but very little AWGN.)

OK

5.6. On-Line Help Screen for the Acknowledgements

Here is the on-line help screen for the "acknowledgements" item:

Acknowledgements

I thank Professor Jack Carlyle for igniting my interest in this subject, and for his helpful advice on numerous occasions. I thank ExperTelligence Inc. for providing the software which made possible the implementation of the signal constellation design tool: ExperCommon LISP® version 2.3B, and ExperInterface Builder®.

OK

5.7. On-Line Help Screen for the References

Here is the on-line help screen for the "references" item:

Bibliography

Carlyle, J., Analog Transmission/Reception for Digital Communication, CS214 Class Notes, Computer Science Department, University of California, Los Angeles, Winter, 1988,

Forney, D., Gallager, R., Lang, G., Longstaff, F., Qureshi, S., Efficient Modulation for Band-Limited Channels, IEEE Journal on Selected Areas in Communication, Vol. SAC-2, No. 5, September, 1985.

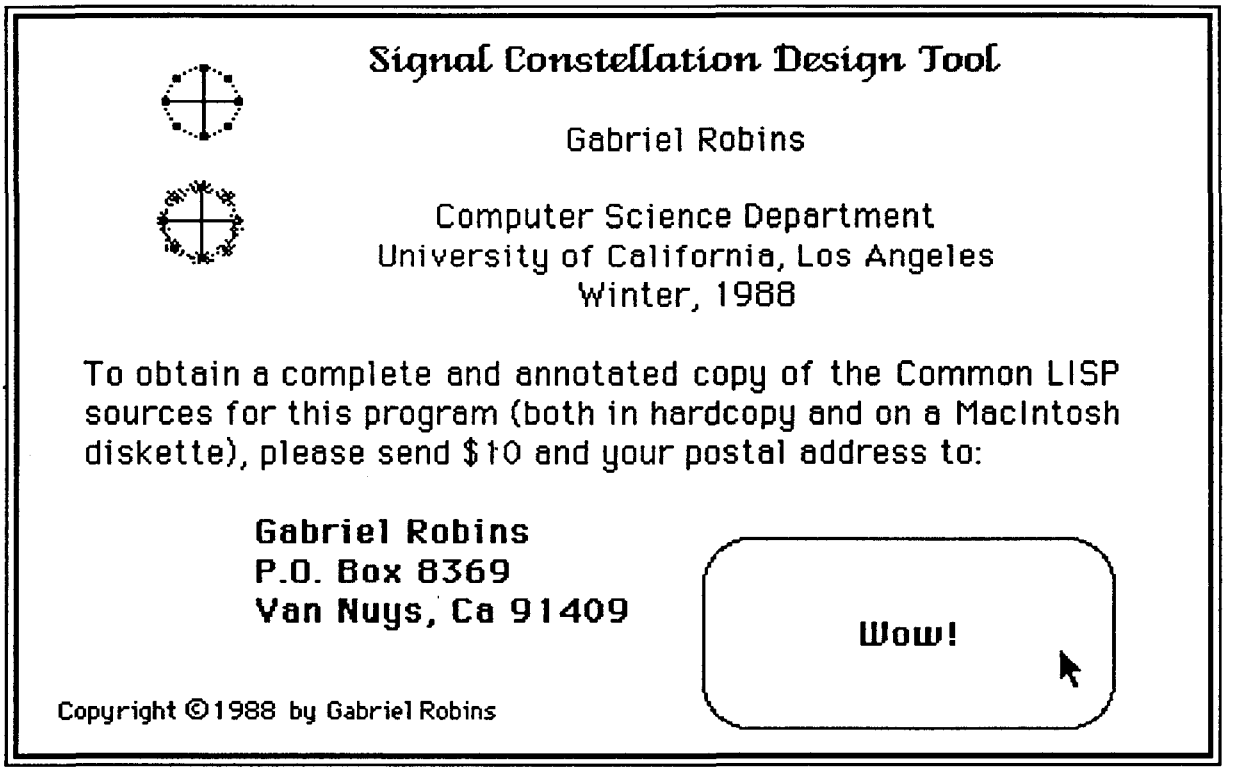
Schwartz, M., Information Transmission, Modulation, and Noise, McGraw-Hill, Third Edition, pp. 212-235, 1980.

Sklar, B., Digital Communications: Fundamentals and Applications, Prentice Hall, New Jersey, pp. 412-424, 1988.

OK

5.8. On-Line Help Screen for the "About..." Item

Here is the on-line help screen for the "About..." item; this screen contains some general information regarding the Signal Constellation Design Tool and is displayed when the user selects the "About..." menu item from the main menu anytime during execution:



6. The Software/Hardware Used

The Signal Constellation Design Tool is implemented in ExperCommon LISP® (version 2.3B), marketed by ExperTelligence, Inc. [Bollay, McConnell, Reali, and Ritz]. ExperCommon LISP is a production LISP system that runs on the MacIntosh family of machines. The user interface was constructed with ExperInterface Builder®, an interactive package that allows a user to quickly and easily design a graphical (menu and icon-based) user interface from scratch on an object-oriented paradigm [Hullot]. The hardware used was the MacIntosh Plus with 2 megabytes of memory and a 20-megabyte hard disk. ExperCommon LISP® and ExperInterface Builder® may be purchased directly from ExperTelligence Inc., 5638 Hollister Avenue, 3rd Floor, Goleta, California 93117, U.S.A., (805) 967-1797..

7. Obtaining the sources

The annotated Common LISP sources for the Signal Constellation Design Tool are available upon request. Although this tool was developed on the MacIntosh, it should be portable to any system which supports Common LISP and reasonable window and graphics conventions. To obtain the sources, both in hardcopy and on a MacIntosh diskette, please send $10 to Gabriel Robins, UCLA Computer Science Department, Los Angeles, California, 90024.

8. Summary

Signal constellation design essentially entails trading off error frequency against information throughput, a chief occupation of modem designers. We proposed and implemented an interactive tool for designing and simulating arbitrary signal constellations. While the actual code that simulates signal constellations is rather trivial in itself, the user interface to this code is not.

To design and construct the user interface we have used Interface Builder, a new interactive tool that greatly facilitates the synthesis of user interfaces through an object-oriented methodology. Using the Interface Builder package and the Signal Constellation Design Tool as the target prototype,

we showed how an order-of-magnitude improvement can be achieved in the effort required to produce a complex user interface.

We hope that we have helped to dispel some of the mystique surrounding user interface synthesis on state-of-the-art workstations by showing that given the proper tools and methodology, the synthesis of complex user interfaces could be rather trivial. In particular, designing and implementing the user interface specified here took only a few days, and that includes the overhead to read the user manuals and learn (from scratch) how to use the software.

9. Acknowledgements

I thank Professor Jack Carlyle for igniting my interest in this subject, and for his helpful advice on numerous occasions. I thank ExperTelligence Inc. for providing the software which made possible the implementation of the Signal Constellation Design Tool.

10. Bibliography

Bollay, D., McConnell, J., Reali, R., Ritz, D., ExperCommon LISP Documentation: Volume I, II, and III, The ExperTelligence Press, Santa Barbara, California, 1987.

Carlyle, J., Analog Transmission/Reception for Digital Communication, CS214 Class Notes, Computer Science Department, University of California, Los Angeles, Winter, 1988,

Forney, D., Gallager, R., Lang, G., Longstaff, F., Qureshi, S., Efficient Modulation for Band-Limited Channels, IEEE Journal on Selected Areas in Communication, Vol. SAC-2, No. 5, September, 1985.

Goodman, D., The Complete HyperCard Handbook, Bantham Books, New York, 1987.

Hullot, J., ExperInterface Builder Documentation, The ExperTelligence Press, Santa Barbara, California, 1987.

Kaczmarek, T., Mark, W., & Wilczynski, D., The CUE Project, Proceedings of SoftFair, July, 1983.

Robins, G. The ISI Grapher: A Portable Tool for Displaying Graphs Pictorially, Invited Talk in Symboliikka '87, Helsinki, Finland, August, 17-18, 1987. Reprinted in Multicomputer Vision, Levialdi, S., Chapter 12, Academic Press, London, 1988.

Schwartz, M., Information Transmission, Modulation, and Noise, McGraw-Hill, Third Edition, pp. 212-235, 1980.

Sklar, B., Digital Communications: Fundamentals and Applications, Prentice Hall, New Jersey, pp. 412-424, 1988.