Computer Programs for the Acquisition and Analysis of Eddy-Current Array Probe Data

RECEIVED
AUG 19 1996
OSTI

Prepared by
J. R. Pate, C. V. Dodd

Oak Ridge National Laboratory

Prepared for
U.S. Nuclear Regulatory Commission

MASTER

AVAILABILITY NOTICE

Availability of Reference Materials Cited in NRC Publications

Most documents cited in NRC publications will be available from one of the following sources:

1. The NRC Public Document Room, 2120 L Street, NW., Lower Level, Washington, DC 20555-0001
2. The Superintendent of Documents, U.S. Government Printing Office, P. O. Box 37082, Washington, DC 20402-9328
3. The National Technical Information Service, Springfield, VA 22161-0002

Although the listing that follows represents the majority of documents cited in NRC publications, it is not intended to be exhaustive.

Referenced documents available for inspection and copying for a fee from the NRC Public Document Room include NRC correspondence and internal NRC memoranda; NRC bulletins, circulars, information notices, inspection and investigation notices; licensee event reports; vendor reports and correspondence; Commission papers; and applicant and licensee documents and correspondence.

The following documents in the NUREG series are available for purchase from the Government Printing Office: formal NRC staff and contractor reports, NRC-sponsored conference proceedings, international agreement reports, grantee reports, and NRC booklets and brochures. Also available are regulatory guides, NRC regulations in the *Code of Federal Regulations*, and *Nuclear Regulatory Commission Issuances*.

Documents available from the National Technical Information Service include NUREG-series reports and technical reports prepared by other Federal agencies and reports prepared by the Atomic Energy Commission, forerunner agency to the Nuclear Regulatory Commission.

Documents available from public and special technical libraries include all open literature items, such as books, journal articles, and transactions. *Federal Register* notices, Federal and State legislation, and congressional reports can usually be obtained from these libraries.

Documents such as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings are available for purchase from the organization sponsoring the publication cited.

Single copies of NRC draft reports are available free, to the extent of supply, upon written request to the Office of Administration, Distribution and Mail Services Section, U.S. Nuclear Regulatory Commission, Washington, DC 20555-0001.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at the NRC Library, Two White Flint North, 11545 Rockville Pike, Rockville, MD 20852-2738, for use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from the American National Standards Institute, 1430 Broadway, New York, NY 10018-3308.

DISCLAIMER NOTICE

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for any third party's use, or the results of such use, of any information, apparatus, product, or process disclosed in this report, or represents that its use by such third party would not infringe privately owned rights.

DISCLAIMER

**Portions of this document may be illegible
in electronic image products. Images are
produced from the best available original
document.**

Computer Programs for the Acquisition and Analysis of Eddy-Current Array Probe Data

Manuscript Completed: August 1994
Date Published: July 1996

Prepared by
J. R. Pate, C. V. Dodd

Oak Ridge National Laboratory
Managed by Lockheed Martin Energy Research Corporation

Oak Ridge National Laboratory
Oak Ridge, TN 37831

J. Muscara, NRC Project Manager

Prepared for
Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001
NRC Job Code B0417

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED ^{HH}

Abstract

The objective of the Improved Eddy-Current ISI for Steam Generators Tubing program is to upgrade and validate eddy-current inspections, including probes, instrumentation, and data processing techniques for inservice inspection of new, used, and repaired steam generator tubes; to improve defect detection, classification and characterization as affected by diameter and thickness variations, denting, probe wobble, tube sheet, tube supports, copper and sludge deposits, even when defect types and other variables occur in combination; to transfer this advanced technology to NRC's mobile NDE laboratory and staff. This report documents computer programs that were developed for acquisition of eddy-current data from specially-designed 16-coil array probes. Complete code as well as instructions for use are provided.

Contents

	<u>Page</u>
Abstract.....	iii
Introduction.....	1
Secort - A program to analyze eddy-current array coil data	2
Opening a file	3
Viewing the data	4
Setting the probe speed	6
Setting the span	8
Setting the rotation	9
Performing a software null.....	9
Setting the voltage conversion	10
Performing a mix	11
Subrouting for SECORT.....	16
Important variables for the program SECORT	60
Listing for the program SECORT	61
SCAN18.C - Program to acquire data from a MIZ-18.....	151
SCZANA.C - Program to acquire data from an Impedance Analyzer.....	158
SCAN30.C and MOVE30.C - Programs to position and acquire data from a MIZ-30	165

**Computer Programs for The Acquisition and Analysis of
Eddy-Current Array Probe Data
J. R. Pate and C. V. Dodd**

Introduction

ORNL has been developing advanced array probes for the inspection of steam generators. These probes include both reflection and pancake coil arrays. There is no commercial software at the present time to perform the types of analysis and display of the data that need to be done in order to best analyze and display the array coil data. This report describes the software we have written to analyze and present array coil data.

The analysis program, SECORT, is written in ANSI C to run under X-Windows on a Hewlett Packard series 700 workstation, using the HP-UX 9.01 operating system. The analysis program will read the data files acquired with Zetec MIZ-30 acquisition program. The analysis program is designed to analyze files using a 16-coil configuration, with the coils arranged according to their location around the circumference of the tubing. Both the MIZ-30 and the MIZ-18 use a multiplexing scheme to drive the different coils and frequencies at separate time intervals. The MIZ-30 will drive a bank of 8 coils at the same time, with the different frequencies. The acquisition program for the MIZ-30 is configured such that the coils are energized in alternate sequence, so adjacent coils are not on at the same time. This minimizes the cross talk between the coils, but the coils are not turned on in numerical order. The analysis program arranges the coils such that the data is displayed in a continuous manner around the circumference of the tubing, so that the shape and circumferential extent of a defect can be determined directly from the display. The analysis program will work with data from either a pancake coil array or a reflection coil array, with no difference between the two. The program will allow linear mixes to be performed between the data at the different frequencies and neural network mixes that have been previously run can be recalled from files. In addition, the program will plot the computed defect depth in different colors or densities, so that problem areas of the tube can be quickly identified.

In addition, software that positions the probe on a standard and then acquires the instrument readings is included. This software is written for both the MIZ-18 and the MIZ-30, and requires acquisition programs from Zetec to work. The MIZ-30 auxiliary programs were written by Zetec and modified by ORNL, and work with the normal MIZ-30 acquire program. The MIZ-18 programs were written by ORNL and require a special MIZ-18 acquisition program from Zetec.

There is another program to acquire multiple frequency data from the HP Impedance Analyzer. All of these programs will position a probe on a sample at a given location using a scanner, controlled by a PC over an IEEE-488 line. The programs will write the acquired data to a file, along with the property values of interest for that particular sample, and the position information. The position controllers allow up to three axis motion, and the standards can be of special design that allows a very large number of property combinations to be measured. These data are used as input for the fitting programs that determine least squares coefficients and neural network weights that can be used in the analysis program.

Secort - A program to analyze eddy-current array coil data

Program SECORT was written at ORNL to aid in the analysis of eddy-current data from the ORNL 16-coil array probes. It provides flexible options for displaying the data and allows the user to perform mixes for the suppression of unwanted signals. It also will apply previously calculated neural network weights to the readings, giving an additional means of characterizing the eddy current data.

The program is compiled by using the following command:

```
OPTIONS="-Aa O2"  
LIBDIRS="-L /usr/lib/Motif1.2 -L /usr/lib/X11R5"  
INCDIRS="-I /usr/include/X11R5 -I /usr/include/Motif1.2"  
LIBS="-lXm -lX11 -lm"  
cc $OPTIONS $LIBDIRS $INCDIRS -o secort secort.c $LIBS
```

The program is then run by typing:

secort

The program then comes up with the five main screen functions displayed. The main screen functions have pull down windows that allow a number of sub-functions of the main function to be performed. An overview of the main screen functions is given, followed by more detailed instructions for running the program.

File

- Open** - Used to select the data file to be analyzed
- Print Screen** - Prints the screen to the laser printer
- Write Data** - Writes the selected data in ASCII format, for use with other programs
- Exit** - Exits the program

Edit

- Probe Speed** - Used to set the probe speed for the offset between the coil banks
- Align** - Not used at this time
- Set volts** - Sets the system "gain" so that the coils read a given voltage

Analysis

- Null** - Sets the "zero point" of a scan for use with mixing
- Normalize** - Not used at this time
- Mix** - Used to suppress the signals from unwanted artifacts in the tubing
- Properties** - Select previously calculated properties to compute

View

- 16 Charts** - Displays 16 data plots side-by-side
- 8 Lissajous** - Displays 8 Lissajous plots
- Density Plots** - Displays 16 data plots with property increases represented by color
- Set Rotation** - The phase of the Lissajous can be rotated to any desired value
- Set Span** - The span or gain of a plot can be set
- Zoom In** - A section of the tube can be magnified
- Zoom Out** - Displays the full data file

File Edit Analysis View Options

Help

SG: Row: Col:

oml

Coll 1	Coll 9	Coll 2	Coll 10	Coll 3	Coll 11	Coll 4	Coll 12	Coll 5	Coll 13	Coll 6	Coll 14	Coll 7	Coll 15	Coll 8	Coll 16
Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V	Frq2 V

filesb_popup

Filter

/rod001a/raw/SG21HCAL00030/

Directories

/rod001a/raw/SG21HCAL00030/
/rod001a/raw/SG21HCAL00030/..

Files

DHR039C0301044
DHR042C0301045
DHR043C0461028
DHR899C8881001
DHR899C8991002
DHR899C8881003
DHR899C8991004
DHR899C8881005

Selection

/rod001a/raw/SG21HCAL00030/DHR899C8991004

OK

Filter

Cancel

Help

SG: 21 Row: 999 Col: 999

oml

Coil 1	Coil 9	Coil 2	Coil 10
300 V	300 V	300 V	300 V

Coil 1	300 kHz	Coil 2	300 kHz
Vpp: 1.03		Vpp: 1.02	
Phase: 107		Phase: 105	
Span: 5.00	▲▼	Span: 5.00	▲▼
Coil 3	300 kHz	Coil 4	300 kHz
Vpp: 1.01		Vpp: 1.01	
Phase: 97		Phase: 92	
Span: 5.00	▲▼	Span: 5.00	▲▼
Coil 5	300 kHz	Coil 6	300 kHz
Vpp: 1.02		Vpp: 0.98	
Phase: 95		Phase: 102	
Span: 5.00	▲▼	Span: 5.00	▲▼
Coil 7	300 kHz	Coil 8	300 kHz
Vpp: 1.00		Vpp: 1.01	
Phase: 103		Phase: 114	
Span: 5.00	▲▼	Span: 5.00	▲▼

SG: 21 Row: 999 Col: 999

oml

SG: 21 Row: 999 Col: 999

oml

SG: 21 Row: 888 Col: 888

oml

SG: 21 Row: 888 Col: 888

oral

Cell 1	Cell 9	Cell 2	Cell 10	Cell 3	Cell 11	Cell 4	Cell 12	Cell 5	Cell 13	Cell 6	Cell 14	Cell 7	Cell 15	Cell 8	Cell 16
100 V	300 V	300 V	300 V	300 V	300 V	300 V	300 V	300 V	300 V	300 V	300 V	300 V	300 V	300 V	300 V

mbdialog_popup

Mix 1 Polynomial	Mix 2 Polynomial	Mix 3 Polynomial
☐ Constant	☐ Constant	☒ Constant
☐ 800 kHz H	☐ 800 kHz H	☐ 800 kHz H
☐ 600 kHz V	☐ 600 kHz V	☐ 600 kHz V
☐ 300 kHz H	☐ 300 kHz H	☐ 300 kHz H
☐ 300 kHz V	☐ 300 kHz V	☐ 300 kHz V
☐ 100 kHz H	☐ 100 kHz H	☐ 100 kHz H
☐ 100 kHz V	☐ 100 kHz V	☐ 100 kHz V
☐ 50 kHz H	☐ 50 kHz H	☐ 50 kHz H
☐ 50 kHz V	☐ 50 kHz V	☐ 50 kHz V

OK Cancel

over the region with the left mouse button held down, and then click the Preserve button. Repeat the process of dragging over the regions to preserve and clicking on the Preserve button until all regions to be preserved have been entered. Then select the first region to suppress by dragging over the scan with the left button pressed down and then click the Suppress button. When all regions to be suppressed have been entered, click the Calculate button to compute the mix coefficients. After this has been done, the mixed data can be displayed in the charts, lissajous windows, and density plot window. To abort the calculation of the mix at any time, click the Cancel button.

Calculating properties

The program will apply previously calculated and stored neural net weights to the data to calculate some property of the sample being tested such as defect depth, deposit type, or wall thickness. To apply the weights to the data, select the appropriate item from the Properties submenu of the Analysis menu. After this has been done, the calculated property can be displayed in the charts like any other type of data.

Subroutines for SECORT

The major subroutines of program SECORT are described briefly below.

allocate_memory

Synopsis

```
float *allocate_memory(nalloc)
      size_t nalloc;
```

Description

Function allocate_memory is called by read_data to allocate space for the data being read from a file and by functions MixCalculateCB and CalDefectDepthCB to allocate space to hold mixed and calculated data. It allocates a block of memory large enough to hold nalloc floating point numbers. If an error occurs in allocating the memory, the function prints an error message and causes program execution to terminate. Otherwise the function returns a pointer to the allocated memory.

External variables used

None

alsqs

Synopsis

```
void alsqs(a, y, b, r2, drows, dcols, ncols)
      float *a;
      float *y;
      float *b;
      float *r2;
      int drows;
      int dcols;
      int ncols;
```

Description

Function alsqs performs the least squares fits used in the mixes for the suppression of unwanted signals. The eddy current data are contained in the matrix a. The vector y contains the data for the desired output of the mix; that is, it contains eddy current data in the regions which the user indicated he wanted to preserve, and zeros for the regions he wanted to suppress. The coefficients for the mix are returned in the vector b. The variables drows and dcols give the number of rows and columns used in the data matrix a, and the variable ncols is the physical number of columns allocated for the matrix a.

External variables used

None

CalDefectDepthCB

Synopsis

```
void CalDefectDepthCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function CalDefectDepthCB is the callback routine associated with the Calculate Defect Depth item in the Properties submenu of the Analysis menu. When called the function will open a file containing neural network weights written by another program and apply these weights to the data to calculate the defect depth at every point in the tube. The function also allocates memory to hold the defect depths and sets program variables to allow the display of the defect depth in the charts.

External variables modified

DataType
lastactivetype
defect_depth_calculated

External variables read

chart_canvas
gc
color
ncharts
chart
npts

ChangeRotationCB

Synopsis

```
void ChangeRotationCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ChangeRotationCB is the callback function associated with the arrow buttons created by function SetRotationCB. If the calling widget is the up arrow, the rotation for that lissajous figure is incremented, and if it is the down arrow, the rotation is decremented.

External variables modified
DataType

External variables read
rotation_up
rotation_down
lissajous
rotation_label

ChangeSpanCB

Synopsis

```
void ChangeSpanCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ChangeSpanCB is the callback function associated with the arrow buttons created by function SetSpanCB. If the function is called in response to the up arrow's being pressed, the span for that lissajous figure is increased. If the down arrow was pressed, the span is decreased.

External variables modified
DataType

External variables read
span_up
span_down
lissajous
span_label
liss_canvas
chart_canvas
gc
color

chartCB

Synopsis

```
void chartCB(widget, event, args, num_args)
    Widget widget;
    XEvent *event;
    String *args;
    int *num_args;
```

Arguments

widget	Specifies the widget where the event occurred.
event	Gives information about the event.
args	Not used.
num_args	Not used.

Description

Function chartCB is the callback function executed when a mouse pointer motion or mouse button event occurs with the pointer in the chart drawing area. If the event is a motion event, the function erases the cursor at the old pointer location and draws it at the new location. If the event is a button press and a region or point is already selected, the lines indicating the selection are erased and the variables storing the selection are reset. Then the location of the pointer at the time of the button press is recorded and a line indicating the beginning of the selected region is drawn. If the event is a button release and the mouse pointer location has changed since the corresponding button press event, the location of the button release is stored and a line indicating the end of the selected region is drawn. If the pointer location has not changed since the corresponding button press event, the location of the selected point is stored and the data labels at the bottom of the charts are updated.

External variables modified

```
spt1,spt2
sspt
sy1,sy2
cursor_y
```

External variables read

```
chart_canvas
gc
color
chart_area_width
data_label
DataType
chart
```

chart_exposeCB

Synopsis

```
void chart_exposeCB(w, client_data, cbs)
    Widget w;
    XtPointer client_data;
    XmDrawingAreaCallbackStruct *cbs;
```

Description

Function `chart_exposeCB` is the callback function executed when the chart drawing area receives an expose event, that is, when a part of the chart drawing area that has been obscured by another window is made visible. The function redraws the data displayed in the charts.

External variables modified

`cursor_y`

External variables read

`file_open`
`chart_canvas`
`color`
`gc`
`chart_area_width`

chart_resizeCB

Synopsis

```
void chart_resizeCB(w, client_data, cbs)
    Widget w;
    XtPointer client_data;
    XmDrawingAreaCallbackStruct *cbs;
```

Description

Function `chart_resizeCB` is the callback function executed when the chart drawing area is resized. The function assigns new values to the variables which keep track of the geometry of the chart drawing area.

External variables modified

`cursor_y`
`chart_area_width`
`chart_area_height`
`chart.llim`
`chart.ulim`

chart.center

External variables read

chart_canvas
ncharts

coil_buttonCB

Synopsis

```
void coil_buttonCB(w, client_data, cbs)
    Widget w;
    XtPointer client_data;
    XmPushButtonCallbackStruct *cbs;
```

Description

Function coil_buttonCB is the callback function executed when one of the coil buttons above the charts is selected. If the button was selected by holding the shift key and pressing the left mouse button with the mouse pointer over the button, the data displayed in that chart is replaced by data from the next lower coil; otherwise the data displayed in that chart is replaced by data from the next higher coil.

External variables modified

chart

External variables read

file_open
ncharts
coil_button
chart_canvas
gc
color

create_gc

Synopsis

```
void create_gc(w)
    Widget w;
```

Description

Function create_gc calls a system routine to create a graphics context for the program to use.

External variables modified

gc

External variables read

None

data_buttonCB

Synopsis

```
void data_buttonCB(w, client_data, cbs)
    Widget w;
    XtPointer client_data;
    XmPushButtonCallbackStruct *cbs;
```

Description

Function `data_buttonCB` is the callback function executed when one of the data type buttons above the charts is selected. If the button was selected with the shift key depressed, the data displayed in that chart is replaced by data of the next lower type; otherwise the data displayed in that chart is replaced by data of the next higher type.

External variables modified

External variables read

do_null

Synopsis

```
void do_null(command)
    int command;
```

Description

Function `do_null` is called by `ToolsNullCB` and `read_data` to subtract from the data of each channel the bare tube values for that channel. If the value of the argument is equal to one, the function calculates the average value of the data for each channel over the region that has been selected and subtracts this average from each channel. If the value of the argument is zero, the function subtracts the previously calculated average value from each channel.

External variables modified

cursor_y
DataType
nulled

spt1
spt2

External variables read

chart_canvas
color
gc
nlag
normalized
npts

do_set_rotation

Synopsis

void do_set_rotation(void);

Description

Function do_set_rotation is called by SetRotationCB, by ViewDensityPlotCB, and by ViewChartCB. The function allows the user to adjust the rotation of the data displayed in the lissajous windows and in the charts and density plot. The first time the function is called it creates a label widget to display the current value of the rotation in each lissajous window. It also creates two arrow buttons in each lissajous window to allow the user to increase or decrease the value of the rotation for each lissajous figure. After the rotation has been set to the desired value, the function may be called again to remove these widgets.

External variables modified

setting_rotation
rotation_down
rotation_up
rotation_label

External variables read

mode
setting_span
liss_data_label
liss_coil_button
liss_data_button
liss_canvas
liss_sep
span_down
span_up
span_label

do_set_span

Synopsis

```
void do_set_span(void);
```

Description

Function `do_set_span` is called by `SetSpanCB`, by `ViewChartCB`, and by `ViewDensityPlotCB` to allow the user to change the scale of the data displayed in the charts and in the lissajous windows. The first time the function is called the routine creates three widgets for each lissajous window: a label to show the current value of the span, an up arrow button to increase the value of the span, and a down arrow button to decrease the value of the span. Callback routines that perform the work of changing the value of the span are associated with the arrow buttons, and the attachments of the neighboring widgets are adjusted to make room for the new ones. If the function is called a second time, it removes the widgets it previously created and resets the attachments of the other widgets.

External variables modified

```
setting_span  
span_down  
span_up  
span_label
```

External variables read

```
setting_rotation  
mode  
liss_data_label  
liss_coil_button  
liss_canvas  
rotation_label  
rotation_up  
rotation_down  
liss_sep
```

dp_colorCB

Synopsis

```
void dp_colorCB(w, client_data, call_data)  
    Widget w;  
    XtPointer client_data;  
    XtPointer call_data;
```

Description

Function `dp_colorCB` is the callback function executed when the Color/Gray button above the density plot window is selected. It toggles the density plot between a color and a gray scale display.

External variables modified

`dp_color_flag`

External variables read

None

dp_dataCB

Synopsis

```
void dp_dataCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function `dp_dataCB` is the callback function executed when the data button above the density plot window is selected. If the shift key was depressed when the button was selected, the data in the density plot window is replaced by data of the next lower type; otherwise the data in the density plot window is replaced by data of the next higher type.

External variables modified

`dp_type`

External variables read

`lastactivetype`
`DataType`
`dp_vertical_flag`

dp_refreshCB

Synopsis

```
void dp_refreshCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function `dp_refreshCB` is the callback routine executed when the Refresh button

above the density plot window is selected. It calls a function to redraw the data in the density plot window.

External variables used

None

draw_chart_data

Synopsis

void draw_chart_data(void);

Descriptions

Function draw_chart_data draws the data in either the sixteen or the four vertical chart displays.

External variables modified

chart

point

External variables read

ncharts

nulled

DataType

nlag

cpt1

cpt2

chart_canvas

gc

draw_density_plot

Synopsis

void draw_density_plot(void);

Description

Function draw_density_plot displays a color or grayscale density plot of the data.

External variables modified

dp_color

External variables read

dp_color_flag

file_open
dp_canvas
gc
DataType
cpt1
cpt2
draw_liss_data

Synopsis

void draw_liss_data(void);

Description

Function draw_liss_data is responsible for drawing the lissajous figures in the lissajous windows.

External variables modified

lissajous.xoffset
lissajous.yoffset
point

External variables read

lissajous
nlag
liss_area_width
liss_area_height
nulled
lpt1
lpt2
npts
DataType
liss_canvas
gc

draw_one_chart_data

Synopsis

void draw_one_chart_data(ichart)
int ichart;

Description

Function draw_one_chart_data draws the data in the chart of index ichart.

External variables modified

point

External variables read

DataType
chart
nlag
cpt1
cpt2
chart_canvas
gc

draw_one_liss_data

Synopsis

```
void draw_one_liss_data(iliss)
    int iliss;
```

Description

Function draw_one_liss_data draws a single lissajous figure in a lissajous window. The index of the lissajous figure to be drawn is given in the argument iliss.

External variables modified

lissajous.xoffset
lissajous.yoffset
point

External variables read

liss_area_width
liss_area_height
nulled
nlag
lissajous
npts
lpt1
lpt2
DataType
liss_canvas
gc

erase_chart_data

Synopsis

```
void erase_chart_data(chart)
    int chart;
```

Description

Function `erase_chart_data` erases the data in the chart specified in its argument.

External variables modified

```
cursor_y
spt1
spt2
sspt
```

External variables read

```
chart_canvas
gc
color
```

FileOpenCancelCB

Synopsis

```
void FileOpenCancelCB(widget, client_data, call_data)
    Widget widget;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function `FileOpenCancelCB` is the callback function executed when the cancel button in the file open dialog box is selected. The function removes the file open dialog box from the screen.

External variables modified

```
file_open_dialog_exists
```

External variables read

```
None
```

FileOpenCB

Synopsis

```
void FileOpenCB(widget, client_data, call_data)
    Widget widget;
    XtPointer client_data;
    XtPointer call_data;
```


Description

Function FileOpenCB is the callback function associated with the Open button in the File menu. It creates a file selection dialog box that allows the user to choose a data file to open, and it associates callback routines with the push buttons in the dialog box.

External variables modified

file_open_dialog_exists

External variables read

toplevel

FilePrint_screenCB

Synopsis

```
void FilePrint_screenCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function FilePrint_screenCB is the callback function executed when the Print Screen button is selected from the File menu. It will print the contents of the output window of the program if the system default printer is a LaserJet.

External variables modified

None

External variables read

main_w

FileWrite_dataCB

Synopsis

```
void FileWrite_dataCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function FileWrite_dataCB is the callback function executed when the Write Data item is selected from the File menu. The function writes out the selected

data in ASCII format so that it can be imported by external programs for analysis or display.

External variables modified

None

External variables read

DataType
spt1
spt2
lag
lastactivetype
npts

get_colors

Synopsis

```
void get_colors(w)
    Widget w;
```

Description

Function `get_colors` calls system routines to allocate the colors the program will use. The pixel values of the colors allocated are written into the elements of the color structure and of the `dp_colors` array.

External variables modified

color
dp_colors

External variables read

None

initialize_chart_display

Synopsis

```
void initialize_chart_display(void);
```

Description

Function `initialize_chart_display` is called by `read_data` when a file has been opened. The function sets variables controlling which data points will be displayed in the charts and also sets variables which record the sizes and positions of the charts within the main window. Finally the function calls subroutines to label the chart buttons and to draw the data in the charts.

External variables modified

cpt1
cpt2
chart.llim
chart.ulim
chart.center
chart_area_width
chart_area_height

External variables read

ncharts
chart_canvas
npts

initialize_liss_display

Synopsis

void initialize_liss_display(void);

Description

Function initialize_liss_display is called by ViewLissajousCB when eight lissajous mode is to be entered. The function sets the variables describing the geometry of the lissajous windows and those controlling which data points will be shown in the lissajous windows. The function also calls subroutines to place the correct text in the buttons and labels associated with the lissajous windows and to draw the lissajous figures.

External variables modified

liss_area_width
liss_area_height
lpt1
lpt2
lissajous.xlim
lissajous.xulim
lissajous.ylim
lissajous.xcenter
lissajous.ycenter

External variables read

cpt1
cpt2
liss_canvas
file_open

label_chart_buttons

Synopsis

void label_chart_buttons(void);

Description

Function **label_chart_buttons** puts the correct labels on the coil and data buttons above the chart displays.

External variables modified

None

External variables read

chart
coil_button
data_button
DataType
file_open

label_liss_buttons

Synopsis

void label_liss_buttons(void);

Description

Function **label_liss_buttons** puts the correct labels on the coil and data buttons in the lissajous windows.

External variables modified

None

External variables read

lissajous
liss_coil_button
liss_data_button
DataType
file_open

liss_coil_buttonCB

Synopsis

void liss_coil_buttonCB(w, client_data, cbs)
Widget w;

```
XtPointer client_data;  
XmPushButtonCallbackStruct *cbs;
```

Description

Function `liss_coil_buttonCB` is the callback routine executed when the coil button in one of the lissajous windows is activated. If the shift key is depressed when the button is selected, the data displayed in the lissajous window is replaced by data from the next lower coil; otherwise the data is replaced by data from the next higher coil.

External variables modified

`lissajous`

External variables read

`liss_coil_button`
`file_open`
`liss_canvas`
`gc`
`color`

`liss_data_buttonCB`

Synopsis

```
void liss_data_buttonCB(w, client_data, cbs)  
    Widget w;  
    XtPointer client_data;  
    XmPushButtonCallbackStruct *cbs;
```

Description

Function `liss_data_buttonCB` is the callback routine executed when the data type button in one of the lissajous windows is selected. If the shift key is pressed when the button is selected, the data displayed in the lissajous window is replaced by data of the next lower data type; otherwise the data is replaced by data of the next higher data type.

External variables modified

`lissajous`

External variables read

`liss_data_button`
`file_open`
`chart_canvas`
`gc`

color
liss_canvas

liss_exposeCB

Synopsis

```
void liss_exposeCB(w, client_data, cbs)
    Widget w;
    XtPointer client_data;
    XmDrawingAreaCallbackStruct *cbs;
```

Description

Function `liss_exposeCB` is the callback function executed when a lissajous window receives an expose event, that is, when a portion of a lissajous window which has been obscured by another window is uncovered. The function redraws the data in the lissajous window.

External variables modified

None

External variables read

file_open
liss_canvas
gc
color

liss_resizeCB

Synopsis

```
void liss_resizeCB(w, client_data, cbs)
    Widget w;
    XtPointer client_data;
    XmDrawingAreaCallbackStruct *cbs;
```

Description

Function `liss_resizeCB` is the callback function executed when a lissajous drawing area receives a resize event. The function assigns new values to the variables which keep track of the geometry of the lissajous drawing area.

External variables modified

lissajous
liss_area_width
liss_area_height

External variables read
liss_canvas

make_info_label

Synopsis
void make_info_label(void);

Description
Function make_info_label writes information about the current tube at the top of the window.

External variables modified
None

External variables read
id
row
col

MixActivatedCB

Synopsis
void MixActivatedCB(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;

Description
Function MixActivatedCB is the callback function executed when one of the toggle buttons in the Set Active dialog box is selected. The function determines which button was selected and toggles the activity of the corresponding mix.

External variables modified
mix_active

External variables read
None

MixApplyCB

Synopsis
void MixApplyCB(w, client_data, call_data)

```
Widget w;  
XtPointer client_data;  
XtPointer call_data;
```

Description

Function MixApplyCB is the callback function executed when the Apply item is selected from the Mix submenu of the Analysis menu. The function takes any mix coefficients which have been previously calculated and applies them to the data to produce mixed data.

External variables modified

```
DataType  
mix_calculated  
lastactivetype
```

External variables read

```
coeffs_calculated  
lasttype  
npts  
mix_polynomial  
coeffr  
coeffi
```

MixCalculateCB

Synopsis

```
void MixCalculateCB(w, client_data, call_data)  
    Widget w;  
    XtPointer client_data;  
    XtPointer call_data;
```

Description

Function MixCalculateCB is the callback function executed when the Calculate button in the Compute Mix dialog box is selected. The function sets up arrays containing the data used as input to the least squares routines and then calls these routines to calculate the mix coefficients. The function then uses the coefficients returned from the least squares routines to calculate the mixed data.

External variables modified

```
mix_nreg_pres  
mix_nreg_sup  
DataType  
coeffs_calculated  
mix_calculated
```


External variables read

coeffr
coeffi
mix_active
mix_polynomial
lag

MixCancelCB

Synopsis

```
void MixCancelCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixCancelCB is the callback function executed when the Cancel button in the Compute Mix dialog box is selected. The function will undo all changes made in the mix parameters made since the opening of the dialog box and abort the calculation of the mix.

External variables modified

mix_nreg_pres
mix_nreg_sup

MixComputeCoefficientsCB

Synopsis

```
void MixComputeCoefficientsCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixComputeCoefficientsCB is the callback function executed when the Compute item is chosen from the Mix submenu of the Analysis menu. The function creates and displays a dialog box with push buttons allowing the user to specify regions of the tube to preserve and suppress in the mix and to calculate the coefficients of the mix.

External variables modified

args

External variables read
None

MixOutputChangedCB

Synopsis

```
void MixOutputChangedCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixOutputChangedCB is the callback function executed when one of the buttons in the Set Output dialog box is selected. The function sets the value of the variable which keeps track of the desired output of the mixes to the newly selected value.

External variables modified
mix_desired_output

External variables read
None

MixPolyChangedCB

Synopsis

```
void MixPolyChangedCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixPolyChangedCB is the callback function executed when one of the toggle buttons in the dialog box created by MixSetPolyCB is selected. The function determines which frequency component corresponds to the selected button and then either adds that frequency component to the mix polynomial or removes it from the mix polynomial depending on whether the button was set or reset.

External variables modified
mix_polynomial

MixPreserveCB

Synopsis

```
void MixPreserveCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixPreserveCB is the callback function that is executed when the Preserve button in the Compute Coefficients dialog box is activated. The function first checks to see if a region has been selected. If so the counter for the number of regions to be preserved is incremented and the first point and the last point in the region are stored. If no region is selected, an error message is printed.

External variables modified

```
mix_pres_pt1
mix_pres_pt2
mix_nreg_pres
```

External variables read

```
spt1
spt2
```

MixRestoreCB

Synopsis

```
void MixRestoreCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixRestoreCB is the callback function executed when the Restore button in the Mix submenu of the Analysis menu is selected. The function opens a data file and reads mix coefficients which were previously calculated and stored.

External variables modified

```
mix_active
coeffs_calculated
coeffr
coeffi
mix_polynomial
```

External variables read
None

MixSaveCB

Synopsis

```
void MixSaveCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixSaveCB is the callback function executed when the Save button is selected from the Mix submenu of the Analysis menu. The function writes any mix coefficients which have been calculated to a data file so they can be used later.

External variables modified
None

External variables read
mix_active
coeffs_calculated
mix_polynomial
coeffr
coeffi

MixSetActiveCancelCB

Synopsis

```
void MixSetActiveCancelCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixSetActiveCancelCB is the callback function executed when the Cancel button in the Set Active dialog box is selected. The function restores the array that keeps track of which mixes are active to the state it was in before any changes were made.

External variables modified
mix_active

External variables read
mix_active_old

MixSetActiveCB

Synopsis

```
void MixSetActiveCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixSetActiveCB is the callback function executed when the Set Active option is selected from the Mix submenu of the Analysis menu. The function creates and displays a dialog box. It also creates toggle buttons and labels to go in the box and associates callback functions with the buttons so that by selecting the buttons, the user can specify which of the mixes are to be calculated.

External variables modified
mix_active_old
args

External variables read
mix_active
toplevel

MixSetActiveOKCB

Synopsis

```
void MixSetActiveOKCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixSetActiveOKCB is the callback function executed when the OK button in the Set Active dialog box is selected. The function destroys the dialog box and then frees any memory allocated to mixes that have just been made inactive.

External variables modified
mix_calculated
DataType

External variables read
lastactivetype

MixSetOutputCancelCB

Synopsis

```
void MixSetOutputCancelCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixSetOutputCancelCB is the callback function executed when the Cancel button in the Set Output dialog box is selected. The function restores the desired output of the mixes to the value it had before any changes were made.

External variables modified
mix_desired_output

External variables read
mix_output_old

MixSetOutputCB

Synopsis

```
void MixSetOutputCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixSetOutputCB is the callback function executed when the Set Output button is selected from the Mix submenu of the Analysis menu. The function display a dialog box allowing the user to choose which frequency will be preserved when mixes are calculated.

External variables modified
mix_output_old

External variables read
mix_desired_output
toplevel

MixSetOutputOKCB

Synopsis

```
void MixSetOutputOKCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixSetOutputOKCB is the callback function executed when the OK button in the dialog box created by MixSetOutputCB is selected. The function destroys the dialog box and preserves any changes which have been made to the desired output of the mixes.

External variables used

None

MixSetPolyCancelCB

Synopsis

```
void MixSetPolyCancelCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixSetPolyCancelCB is the callback function executed when the Cancel button in the dialog box created by MixSetPolyCB is selected. The function restores the mix polynomials to the values they had before MixSetPolyCB was called, and it removes the dialog box from the screen.

External variables modified

mix_polynomial

External variables read

mix_polynomial_old

MixSetPolyCB

Synopsis

```
void MixSetPolyCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
```

XtPointer call_data;

Description

Function MixSetPolyCB is the callback function executed when the Set Polynomial button is selected from the Mix submenu of the Analysis menu. The function creates and displays a dialog box allowing the user to choose which frequency components of the data will be used in calculating the polynomials for each of the three mixes.

External variables modified

mix_polynomial_old

External variables read

mix_polynomial

MixSetPolyOKCB

Synopsis

```
void MixSetPolyOKCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixSetPolyOKCB is the callback function executed when the OK button in the dialog box created by MixSetPolyCB is selected. The function removes the dialog box and preserves any changes which have been made to the mix polynomials.

External variables used

None

MixSuppressCB

Synopsis

```
void MixSuppressCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function MixSuppressCB is the callback function that is executed when the Suppress button in the Compute Coefficients dialog box is activated. The

function first checks to see if a region has been selected. If it has been, the counter for the number of regions to be suppressed is incremented and the location of the region is stored. If no region is selected, an error message is printed.

External variables modified

mix_sup_pt1
mix_sup_pt2
mix_nreg_sup

External variables read

spt1
spt2

ProbeSpeedChanged

Synopsis

```
void ProbeSpeedChanged(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ProbeSpeedChanged is the callback function executed when the value of the scale widget in the Probe Speed dialog box is changed. The function records the new value of the probe speed.

External variables used

None

quitCB

Synopsis

```
void quitCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function quitCB is the callback function executed when the Exit button is selected from the File menu. The function causes the program to stop execution.

External variables used

None

rdgexp

Synopsis

```
void rdgexp(readng, dat, offset_flag, npoly, maxdegree, m)
    float *readng;
    float *dat;
    int offset_flag;
    int npoly;
    int maxdegree;
    int m;
```

Description

Function `rdgexp` expands the data into the format needed for input to the function `alsqs`. The pointer `readng` points to the expanded data. The pointer `dat` points to the input data. Variable `offset_flag` is 1 if a constant term is to be included in the fit and 0 otherwise. Variable `npoly` gives the length of the vector `dat`, `maxdegree` give the maximum degree of the output polynomial, and `m` gives the index of the mix currently being performed.

External variables read

`mix_polynomial`

read_data

Synopsis

```
void read_data(w, client_data, cbs)
    Widget w;
    XtPointer client_data;
    XmFileSelectionCallbackStruct *cbs;
```

Description

Function `read_data` is the callback function executed when the OK button in the file open dialog box is selected. After freeing resources occupied by data from any previously opened file, the function opens the selected file and reads the data from it, allocating memory as necessary. When the data has been read in, the function corrects for the velocity of the probe, and it will do a software null on the data and convert the raw readings to volts if the information needed to do this exists.

External variables modified

file_open
file_open_dialog_exists
DataType
nlag
npts
freq

External variables read

lasttype
nulled
voltage_set

reallocate_memory

Synopsis

```
float *reallocate_memory(nalloc, ptr1)
    size_t nalloc;
    float *ptr1;
```

Description

Function `reallocate_memory` is called by `read_data` to allocate space for data being read from a file. The arguments to the function are `nalloc`, which specifies the number of floating point variables for which to allocate memory, and `ptr1`, a pointer to the memory allocated on a previous call to `allocate_memory` or `reallocate_memory`. The function is called to allocate additional memory when the memory allocated by previous calls to `allocate_memory` or `reallocate_memory` has been filled. It is also called to free excess memory when the end of the input data file is reached. It accepts as input the pointer returned by `allocate_memory` or `reallocate_memory` and returns a pointer to a segment of memory large enough to hold `nalloc` floating point numbers. If an error occurs during memory allocation, the function prints an error message and causes the program to abort.

External variables used

None

RefreshCB

Synopsis

```
void RefreshCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function RefreshCB is the callback function executed when the Refresh item is selected from the View menu. The function causes the screen to be redrawn.

External variables modified

None

External variables read

file_open
chart_canvas
gc
color
liss_canvas
mode

SetRotationCB

Synopsis

```
void SetRotationCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function SetRotationCB is the callback routine executed when the Set Rotation button in the View menu is selected. It calls the function do_set_rotation.

External variables used

None

SetSpanCB

Synopsis

```
void SetSpanCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function SetSpanCB is the callback routine executed when the Set Span button in the View menu is selected. It calls the function do_set_span.

External variables used

None

setup_chart_display

Synopsis

```
void setup_chart_display(void);
```

Description

Function `setup_chart_display` is called by main to specify which data will initially be displayed in the charts. The function also labels the buttons associated with the charts.

External variables modified

chart

External variables read

None

setup_liss_display

Synopsis

```
void setup_liss_display(void);
```

Description

Function `setup_liss_display` is called by main to specify which data will initially be displayed in the lissajous windows. The function also labels the buttons in the lissajous windows.

External variables modified

lissajous

External variables read

None

setup_types

Synopsis

```
void setup_types(void);
```

Description

Function `setup_types` is called by main to initialize the types of data that the program may display. A type can be raw data at a particular frequency, data which has been mixed to suppress unwanted signals but otherwise to reproduce

data of a given frequency, or a calculated quantity such as defect depth.

External variables modified

DataType
lasttype
lastactivetype

External variables read

None

SetVoltsCB

Synopsis

```
void SetVoltsCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function SetVoltsCB is the callback function executed when the SetVolts button is selected from the Edit menu. The function calculates the maxima over the selected region of all of the channels and stores these values in the variables used to hold the conversion factors between readings and volts. The function then calls set_volts to do the actual conversion.

External variables modified

cursor_y
lpt1
lpt2
spt1
spt2
voltfac

External variables read

cpt1
cpt2
DataType
mode
nlag

set_volts

Synopsis

```
void set_volts(void);
```

Description

Function set_volts is called by SetVoltsCB and by read_data to convert raw readings into probe voltage readings.

External variables modified

DataType
voltage_set

External variables read

voltfac

ToolsNullCB

Synopsis

```
void ToolsNullCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ToolsNullCB is the callback function executed when the Null button is selected from the Analysis menu. The function calls do_null with an argument instructing it to calculate new null values for each channel.

External variables used

None

ToolsProbe_speedApplyCB

Synopsis

```
void ToolsProbe_speedApplyCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ToolsProbe_speedApplyCB is the callback function executed when the Apply button in the Probe Speed dialog box is selected. The function redisplay the data in the charts with the newly chosen value for the probe speed without removing the dialog box.

External variables modified

probe_speed
nlag

External variables read

file_open
ncharts
chart
slices_per_sec

ToolsProbe_speedCancelCB

Synopsis

```
void ToolsProbe_speedCancelCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ToolsProbe_speedCancelCB is the callback function executed when the Cancel button in the Probe Speed dialog box is selected. The function restores the variables which keep track of the probe speed to the values they had before any changes were made and redisplay the data in the charts with the old probe speed values.

External variables modified

probe_speed
nlag

External variables read

file_open
chart_canvas
gc
color
ncharts
chart
slices_per_sec
distance_between_coil_groups

ToolsProbe_speedCB

Synopsis

```
void ToolsProbe_speedCB(w, client_data, call_data)
```



```
Widget w;  
XtPointer client_data;  
XtPointer call_data;
```

Description

Function ToolsProbe_speedCB is the callback function executed when the Probe Speed item is selected from the Edit menu. The function creates and displays a dialog box with a scale widget which allows the user to tell the program the speed at which the probe was moving when the data in the currently open file were taken. It is necessary that the program have this information so that it can correctly align the data taken with different coils.

External variables modified

None

External variables read

toplevel

ToolsProbe_speedOKCB

Synopsis

```
void ToolsProbe_speedOKCB(w, client_data, call_data)  
    Widget w;  
    XtPointer client_data;  
    XtPointer call_data;
```

Description

Function ToolsProbe_speedOKCB is the callback function executed when the OK button in the Probe Speed dialog box is selected. The function preserves the changes made in the value of probe speed, destroys the dialog box, and redisplay the chart data.

External variables modified

probe_speed
nlag

External variables read

file_open
chart_canvas
gc
color
ncharts
slices_per_sec

distance_between_coil_groups

update_liss_labels

Synopsis

```
void update_liss_labels(liss)
    int liss;
```

Description

Function update_liss_labels calculates the peak-to-peak voltage and phase for the lissajous figure specified in its argument, or, if the argument is less than zero, it calculates the peak-to-peak voltage and phase for all of the lissajous figures on the screen. The function then writes these values into the label widgets attached to the lissajous windows.

External variables modified

None

External variables read

lissajous
nlag
npts

ViewChartCB

Synopsis

```
void ViewChartCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ViewChartCB is the callback function executed when the 16 Charts item is selected from the View menu. If the program is in eight lissajous mode, this function will return to the mode in which 16 charts are visible.

External variables modified

mode
ncharts

External variables read

liss_canvas

liss_coil_button
liss_data_button
liss_data_label
liss_sep
coil_button
data_button
data_label
chart_canvas
gc
color

ViewDensityPlotCB

Synopsis

```
void ViewDensityPlotCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ViewDensityPlotCB is the callback function executed when the Density Plot item is selected from the View menu. The function switches the display to the mode in which four charts and a density plot are visible. The chart drawing area is resized and the density plot drawing area and its associated widgets are pasted into the available space.

External variables modified

mode
ncharts

External variables read

data_button
coil_button
data_label
chart_canvas
setting_rotation
setting_span
liss_canvas
liss_coil_button
liss_data_button
liss_data_label
liss_sep
dp_canvas

dp_refresh
dp_color_button
dp_data_button

ViewLissajousCB

Synopsis

```
void ViewLissajousCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ViewLissajousCB is the callback function executed when the 8 Lissajous button is selected from the View menu. It switches to the mode in which eight lissajous figures and four charts are shown on the screen. The widgets associated with the rightmost twelve charts are removed and the chart drawing area is reduced in size. Then the lissajous drawing areas and their associated widgets are placed on the screen.

External variables modified

ncharts
mode

External variables read

mode
data_button
coil_button
data_label
chart_canvas
dp_canvas
dp_refresh
dp_color_button
dp_data_button
liss_sep
liss_data_label
liss_coil_button
liss_data_button
liss_canvas
file_open
gc
color

ViewZoom_inCB

Synopsis

```
void ViewZoom_inCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ViewZoom_inCB is the callback function executed when the Zoom In item from the View menu is selected. If a region of data has been selected in the charts, the function expands this region so that it occupies the entire chart area.

External variables modified

```
cpt1
cpt2
spt1
spt2
sspt
cursor_y
```

External variables read

```
chart_canvas
gc
color
```

ViewZoom_outCB

Synopsis

```
void ViewZoom_outCB(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;
```

Description

Function ViewZoom_outCB is the callback function executed when the Zoom Out button in the View menu is activated. The function displays the full tube in the charts.

External variables modified

```
cpt1
cpt2
```

spt1
spt2
sspt
cursor_y

External variables read

chart_canvas
gc
color

Important variables for the program SECORT

Some of the most important variables of program SECORT are listed below with a brief description of their purpose.

cpt1	The first data point shown in the chart or density plot windows
cpt2	The last data point shown in the chart or density plot windows
file_open	Flag indicating whether a data file has been opened
lpt1	The first data point shown in the lissajous windows
lpt2	The last data point shown in the lissajous windows
ncharts	The number of charts displayed
npts	The total number of data points in the currently open file
nulled	Flag indicating whether a software null has been performed
probe_speed	Numerical value of the probe speed in inches per second
slices_per_sec	Rate of data acquisition in data points per second
spt1	The first data point in the selected region
spt2	The last data point in the selected region
sspt	The selected data point

Listing for the program SECORT

A complete listing of program SECORT.C is given below.

```
/*
 * Program secort.c   Version 6/15/94
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include <Xm/MainW.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/CascadeBG.h>
#include <Xm/RowColumn.h>
#include <Xm/FileSB.h>
#include <Xm/LabelG.h>
#include <Xm/Separator.h>
#include <Xm/DialogS.h>
#include <Xm/PanedW.h>
#include <Xm/Scale.h>
#include <Xm/ToggleBG.h>
#include <Xm/Frame.h>
#include <Xm/ArrowBG.h>

typedef struct {
    char *label;
    void (*callback)();
    XtPointer data;
} ActionAreaItem;

typedef struct {
    char *label;
    WidgetClass *class;
    char mnemonic;
    char *accelerator;
    char *accel_text;
    void (*callback)();
    XtPointer callback_data;
    void *subitems;
} MenuItemType;

typedef struct {
    Widget shell;
    float new_probe_speed;
    float old_probe_speed;
} ProbeSpeedDataType;

Widget GetTopShell(Widget w);
void setup_types(void);

void RefreshCB(Widget,XtPointer,XtPointer);
void chartCB(Widget widget, XEvent *event, String *args, int *num_args);
void chart_exposeCB(Widget w, XtPointer client_data,
    XmDrawingAreaCallbackStruct *cbs);
void chart_resizeCB(Widget w, XtPointer client_data,
    XmDrawingAreaCallbackStruct *cbs);
void handle_chart_resize(void);

void coil_buttonCB(Widget,XtPointer,XmPushButtonCallbackStruct *);
void data_buttonCB(Widget,XtPointer,XmPushButtonCallbackStruct *);
```



```

void FileOpenCB(Widget,XtPointer,XtPointer);
void FileOpenCancelCB(Widget,XtPointer,XtPointer);
void read_data(Widget w,XtPointer client_data,
    XmFileSelectionBoxCallbackStruct *cbs);
void FilePrint_screenCB(Widget,XtPointer,XtPointer);
void FileWrite_dataCB(Widget,XtPointer,XtPointer);

void ToolsProbe_speedCB(Widget,XtPointer,XtPointer);
void ToolsProbe_speedOKCB(Widget,XtPointer,XtPointer);
void ToolsProbe_speedCancelCB(Widget,XtPointer,XtPointer);
void ToolsProbe_speedApplyCB(Widget,XtPointer,XtPointer);
void ToolsNullCB(Widget,XtPointer,XtPointer);
void do_null(int command);
void SetVoltsCB(Widget w,XtPointer client_data,XtPointer call_data);
void set_volts(void);

void ToolsApplyNormCB(Widget,XtPointer,XtPointer);
void ToolsComputeNormCB(Widget,XtPointer,XtPointer);
void ToolsSaveNormCB(Widget,XtPointer,XtPointer);
void ToolsRestoreNormCB(Widget,XtPointer,XtPointer);
void unnormalize(void);

void MixSetActiveCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixSetActiveOKCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixSetActiveCancelCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixActivatedCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixSetOutputCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixSetOutputOKCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixSetOutputCancelCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixOutputChangedCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixSetPolyCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixSetPolyOKCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixSetPolyCancelCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixApplyCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixSaveCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixRestoreCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixPolyChangedCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixComputeCoefficientsCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixPreserveCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixSuppressCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixCancelCB(Widget w,XtPointer client_data,XtPointer call_data);
void MixCalculateCB(Widget w,XtPointer client_data,XtPointer call_data);
void rdgexp(float *readng, float *dat,int offset_flag,int npoly,int maxdegree,int m);
void alsqs(float *a, float *y, float *b, float *r2, int drows, int dcols,
    int ncols);

void CalDefectDepthCB(Widget,XtPointer,XtPointer);

void ViewLissajousCB(Widget,XtPointer,XtPointer);
void ViewChartCB(Widget,XtPointer,XtPointer);
void ViewDensityPlotCB(Widget,XtPointer,XtPointer);
void dp_refreshCB(Widget,XtPointer,XtPointer);
void dp_colorCB(Widget,XtPointer,XtPointer);
void dp_dataCB(Widget,XtPointer,XmPushButtonCallbackStruct *);
void ViewZoom_inCB(Widget,XtPointer,XtPointer);
void ViewZoom_outCB(Widget,XtPointer,XtPointer);
void SetSpanCB(Widget w,XtPointer client_data,XtPointer call_data);
void do_set_span(void);
void ChangeSpanCB(Widget w,XtPointer client_data,XtPointer call_data);
void SetRotationCB(Widget w,XtPointer client_data,XtPointer call_data);
void do_set_rotation(void);
void ChangeRotationCB(Widget w,XtPointer client_data,XtPointer call_data);

void ProbeSpeedChanged(Widget w, XtPointer client_data,
    XmScaleCallbackStruct *cbs);
void quitCB(Widget, XtPointer, XtPointer);

void setup_chart_display(void);

```

```

void initialize_chart_display(void);
void draw_chart_dividers(void);
void draw_chart_data(void);
void draw_one_chart_data(int);
void erase_chart_data(int);

void setup_liss_display(void);
void initialize_liss_display(void);
void label_liss_buttons(void);
void draw_liss_data(void);
void draw_one_liss_data(int);
void liss_exposeCB(Widget w, XtPointer client_data,
    XmDrawingAreaCallbackStruct *cbs);
void liss_resizeCB(Widget w, XtPointer client_data,
    XmDrawingAreaCallbackStruct *cbs);
void liss_coil_buttonCB(Widget, XtPointer, XmPushButtonCallbackStruct *);
void liss_data_buttonCB(Widget, XtPointer, XmPushButtonCallbackStruct *);
void update_liss_labels(int);

void draw_density_plot(void);

float *allocate_memory(size_t nalloc);
float *reallocate_memory(size_t nalloc, float *ptr1);

void get_colors(Widget);
void create_gc(Widget);

Widget BuildPulldownMenu(Widget parent, char *menu_title,
    char menu_mnemonic, MenuItemType *items);
void draw_logo(void);
void make_info_label(void);
void label_chart_buttons(void);
Widget create_action_area(Widget parent, ActionAreaItem *actions,
    int num_actions);

Arg args[20];
GC gc;

struct {
    unsigned long background;
    unsigned long foreground;
    unsigned long red;
    unsigned long medium;
    unsigned long cursor;
    unsigned long light;
    unsigned long highlight;
    unsigned long black;
} color;

struct {
    int active;
    float *hdata;
    float *vdata;
    float span;
    float rotation;
    int freqind;
    int frequency;
    int un;
    int mix;
    int mixind;
    int property;
} DataType[16][10];

struct {
    int coil;
    int type;
    int vertical_flag;
    float data_offset;

```

```

    int ulim;
    int llim;
    int center;
} chart[16];

struct {
    int coil;
    int type;
    float xoffset;
    float yoffset;
    int xulim;
    int xllim;
    int yulim;
    int yllim;
    int xcenter;
    int ycenter;
} lissajous[8];

Widget toplevel;
Widget main_w;
Widget form;
Widget chart_canvas, liss_canvas[8];
Widget dp_canvas;
Widget dp_data_button, dp_refresh;
Widget dp_color_button;
Widget coil_button[16], data_button[16];
Widget data_label[16];
Widget infolabel;
Widget liss_sep[8];
Widget liss_coil_button[8], liss_data_button[8];
Widget liss_data_label[8][2];
Widget span_label[8];
Widget span_up[8], span_down[8];
Widget rotation_label[8];
Widget rotation_up[8], rotation_down[8];

int file_open = 0;
int file_open_dialog_exists = 0; /*move*/
int nulled = 0;
int voltage_set = 0;
int normalized = 0;
int defect_depth_calculated = 0;
int setting_span = 0;
int setting_rotation = 0;

int npts;
int cpt1, cpt2;
int lpt1, lpt2;
int spt1 = -1, spt2 = -1;
int sspt = -1;
int sy1, sy2;
char id[5];
char row[5];
char col[5];
int freq[4];
int ncharts = 16;
int lasttype;
int lastactivetype;
int mode = 1; /* 1 for chart mode, 2 for lissajous mode */

int dp_nlevels = 8;
int dp_type = 1;
int dp_vertical_flag = 1;
unsigned long dp_color[8];
unsigned long dp_colors[2][8];
int dp_color_flag = 1;

Dimension logoheight, logowidth;

```

```

Pixmap logopixmap;

Dimension chart_area_width, chart_area_height;
Dimension liss_area_width, liss_area_height;
XPoint point[2001];
int cursor_y = -1;

int nlag = 0;
float probe_speed = 15.;
int slices_per_sec;
/*float distance_between_coil_groups = 0.;*/
float distance_between_coil_groups = 4.5;

float normfac[16][4];
float phase[16][4];
float voltfac[16][4];

/* MIX VARIABLES */

int mix_active[4] = {1,1,1,1};
int mix_calculated[4] = {0,0,0,0};
int coeffs_calculated[4] = {0,0,0,0};
int mix_polynomial[3][9] = {
    {1,1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1,1},
    {0,0,1,1,1,1,1,0,1}};
int mix_desired_output = 1;
int mix_nreg_pres = 0;
int mix_pres_pt1[25];
int mix_pres_pt2[25];
int mix_nreg_sup = 0;
int mix_sup_pt1[25];
int mix_sup_pt2[25];
int mix_active_old[4];
int mix_polynomial_old[3][9];
int mix_output_old;
float *coeffr[4][16];
float *coeffi[4][16];

void main(int argc, char *argv[])
{
    Widget menubar, menu_pane, button;
    Widget cascade; /*temporary */
    Widget logolabel;
    XtAppContext app;
    XGCValues gcv;
    XSetWindowAttributes attrs;
    int i;
    int n;
    XtActionsRec actions;
    String translations =
        "<Btn1Down>: chartCB()\n\
         <Btn1Up>: chartCB()\n\
         <Motion>: chartCB()";
    XmString cstring;
    char tmp[80];
    ProbeSpeedDataType ProbeSpeedData;
    String fallbacks[] = {
        "Demos*background: Gray50",
        NULL};

    MenuItemType file_menu[] = {
        {"Open...", &xmPushButtonGadgetClass,
         NULL, NULL, NULL, FileOpenCB, NULL, NULL},
        {"Print screen", &xmPushButtonGadgetClass,
         NULL, NULL, NULL, FilePrint_screenCB, NULL, NULL},
        {"Write data", &xmPushButtonGadgetClass,
         NULL, NULL, NULL, FileWrite_dataCB, NULL, NULL},

```

```

{"Exit",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,quitCB,NULL,NULL},
NULL
};

MenuItemType edit_menu[] = {
{"Probe speed",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ToolsProbe_speedCB,NULL,NULL},
{"Align",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,NULL,NULL,NULL},
{"Set Volts",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,SetVoltsCB,NULL,NULL},
NULL
};

MenuItemType mix_menu[] = {
{"Apply coefficients",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,MixApplyCB,NULL,NULL},
{"Compute coefficients",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,MixComputeCoefficientsCB,NULL,NULL},
{"Save coefficients",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,MixSaveCB,NULL,NULL},
{"Restore coefficients",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,MixRestoreCB,NULL,NULL},
{"Set active/inactive",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,MixSetActiveCB,NULL,NULL},
{"Set polynomial",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,MixSetPolyCB,NULL,NULL},
{"Set desired output",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,MixSetOutputCB,NULL,NULL},
{"Adjust mix 0",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,NULL,NULL,NULL},
NULL
};

MenuItemType normalize_menu[] = {
{"Apply normalization",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ToolsApplyNormCB,NULL,NULL},
{"Compute normalization factors",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ToolsComputeNormCB,NULL,NULL},
{"Save normalization factors",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ToolsSaveNormCB,NULL,NULL},
{"Restore normalization factors",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ToolsRestoreNormCB,NULL,NULL},
NULL
};

MenuItemType property_menu[] = {
{"Calculate defect depth",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,CalDefectDepthCB,NULL,NULL},
NULL
};

MenuItemType analysis_menu[] = {
{"Null",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ToolsNullCB,NULL,NULL},
{"Normalize",&xmCascadeButtonGadgetClass,
 NULL,NULL,NULL,NULL,NULL,NULL},
{"Mix",&xmCascadeButtonGadgetClass,
 NULL,NULL,NULL,NULL,NULL,NULL},
{"Properties",&xmCascadeButtonGadgetClass,
 NULL,NULL,NULL,NULL,NULL,NULL},
NULL
};

MenuItemType view_menu[] = {
{"16 Charts",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ViewChartCB,NULL,NULL},

```

```

{"8 Lissajous",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ViewLissajousCB,NULL,NULL},
{"Density Plot",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ViewDensityPlotCB,NULL,NULL},
{"Set Rotation",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,SetRotationCB,NULL,NULL},
{"Set Span",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,SetSpanCB,NULL,NULL},
{"Zoom In",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ViewZoom_inCB,NULL,NULL},
{"Zoom Out",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,ViewZoom_outCB,NULL,NULL},
{"Refresh",&xmPushButtonGadgetClass,
 NULL,NULL,NULL,RefreshCB,NULL,NULL},
NULL
};

edit_menu[0].callback_data = &ProbeSpeedData;
analysis_menu[1].subitems = normalize_menu;
analysis_menu[2].subitems = mix_menu;
analysis_menu[3].subitems = property_menu;

id[0] = '\0';
row[0] = '\0';
col[0] = '\0';

for(i=0;i<16;i++) {
    chart[i].coil = i+1;
}

toplevel = XtVaAppInitialize(&app,"Demos",NULL,0,
                             &argc,argv,fallbacks,
                             NULL);

/* Create main window */

n = 0;
main_w = XmCreateMainWindow(toplevel,"main_w",args,n);
XtManageChild(main_w);

get_colors(main_w);

/* Create menubar */

n = 0;
menubar = XmCreateMenuBar(main_w,"menubar",args,n);
XmChangeColor(menubar,color.medium);
XtVaSetValues(menubar,
              XmNforeground,color.foreground,
              NULL);

BuildPulldownMenu(menubar,"File",NULL,file_menu);
BuildPulldownMenu(menubar,"Edit",NULL,edit_menu);
BuildPulldownMenu(menubar,"Analysis",NULL,analysis_menu);
BuildPulldownMenu(menubar,"View",NULL,view_menu);

/* TEMPORARY options menu */

n = 0;
cascade = XmCreateCascadeButtonGadget(menubar,"Options",args,n);
XtManageChild(cascade);

/* TEMPORARY help menu */

n = 0;
cascade = XmCreateCascadeButtonGadget(menubar,"Help",args,n);
XtManageChild(cascade);
XtVaSetValues(menubar,

```

```

        XmNmenuHelpWidget,cascade,
        NULL);

XtManageChild(menubar);

/* Create form */

n = 0;
XtSetArg(args[n],XmNwidth,1000);n++;
XtSetArg(args[n],XmNheight,800);n++;
XtSetArg(args[n],XmNfractionBase,170);n++;
form = XmCreateForm(main_w,"form",args,n);
XtManageChild(form);

/* Create information label */

n = 0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNalignment,XmALIGNMENT_END);n++;
infoLabel = XmCreateLabelGadget(form,"infoLabel",args,n);
make_info_label();
XtManageChild(infoLabel);

/* Create the two rows of buttons at the top */

for(i=0;i<16;i++) {
    n = 0;
    XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNtopWidget,infoLabel);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
    XtSetArg(args[n],XmNleftPosition,(i+1)*10);n++;
    XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION);n++;
    XtSetArg(args[n],XmNrightPosition,(i+2)*10);n++;
    sprintf(tmp,"coilbutton%d",i+1);
    coil_button[i] = XmCreatePushButtonGadget(form,tmp,args,n);
    XtManageChild(coil_button[i]);
    XtAddCallback(coil_button[i],
        XmNactivateCallback,(XtCallbackProc)coil_buttonCB,NULL);
}

for(i=0;i<16;i++) {
    n = 0;
    XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNtopWidget,coil_button[i]);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
    XtSetArg(args[n],XmNleftPosition,(i+1)*10);n++;
    XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION);n++;
    XtSetArg(args[n],XmNrightPosition,(i+2)*10);n++;
    sprintf(tmp,"databutton%d",i+1);
    data_button[i] = XmCreatePushButtonGadget(form,tmp,args,n);
    XtManageChild(data_button[i]);
    XtAddCallback(data_button[i],
        XmNactivateCallback,(XtCallbackProc)data_buttonCB,NULL);
}

/* Create data labels */

for(i=0;i<16;i++) {
    sprintf(tmp," ");
    cstring = XmStringCreateSimple(tmp);
    n = 0;
    XtSetArg(args[n],XmNlabelString,cstring);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
    XtSetArg(args[n],XmNleftPosition,(i+1)*10);n++;
    XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION);n++;
    XtSetArg(args[n],XmNrightPosition,(i+2)*10);n++;
    XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM);n++;
    sprintf(tmp,"dataLabel%d",i);

```

```

    data_label[i] = XmCreateLabelGadget(form,tmp,args,n);
    XtManageChild(data_label[i]);
}

/* Create chart drawing area */

actions.string = "chartCB";
actions.proc = (XtActionProc)chartCB;
XtAppAddActions(app,&actions,1);

n = 0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNtopWidget,data_button[0]);n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNbottomWidget,data_label[0]);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
XtSetArg(args[n],XmNleftPosition,10);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNtranslations,XtParseTranslationTable(translations));n++;
chart_canvas = XmCreateDrawingArea(form,"chart_canvas",args,n);

XtAddCallback(chart_canvas,XmNcxposeCallback,
               (XtCallbackProc)chart_exposeCB,NULL);
XtAddCallback(chart_canvas,XmNresizeCallback,
               (XtCallbackProc)chart_resizeCB,NULL);

create_gc(chart_canvas);

/* Create density plot widgets */

n = 0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNtopWidget,data_button[0]);n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNbottomWidget,data_label[0]);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNleftWidget,chart_canvas);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM);n++;
dp_canvas = XmCreateDrawingArea(form,"dp_canvas",args,n);

n = 0;
cstring = XmStringCreateLocalized("Refresh");
XtSetArg(args[n],XmNlabelString,cstring);n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNbottomWidget,dp_canvas);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION);n++;
XtSetArg(args[n],XmNrightPosition,165);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
XtSetArg(args[n],XmNleftPosition,145);n++;
dp_refresh = XmCreatePushButtonGadget(form,"dp_refresh",args,n);
XmStringFree(cstring);
XtAddCallback(dp_refresh,XmNactivateCallback,
               (XtCallbackProc)dp_refreshCB,NULL);

n = 0;
cstring = XmStringCreateLocalized("Color/Gray");
XtSetArg(args[n],XmNlabelString,cstring);n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNbottomWidget,dp_canvas);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
XtSetArg(args[n],XmNleftPosition,115);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION);n++;
XtSetArg(args[n],XmNrightPosition,135);n++;
dp_color_button = XmCreatePushButtonGadget(form,"dp_color",args,n);
XmStringFree(cstring);
XtAddCallback(dp_color_button,XmNactivateCallback,
               (XtCallbackProc)dp_colorCB,NULL);

```



```

n = 0;
sprintf(tmp,"%d %d",dp_type,dp_vertical_flag);
cstring = XmStringCreateLocalized(tmp);
XtSetArg(args[n],XmNlabelString,cstring);n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNbottomWidget,dp_canvas);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
XtSetArg(args[n],XmNleftPosition,85);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION);n++;
XtSetArg(args[n],XmNrightPosition,105);n++;
dp_data_button = XmCreatePushButtonGadget(form,"dp_data",args,n);
XmStringFree(cstring);
XtAddCallback(dp_data_button,XmNactivateCallback,
              (XtCallbackProc)dp_dataCB,NULL);

/* Create lissajous separators */

for(i=0;i<4;i++) {
    n = 0;
    XtSetArg(args[n],XmNorientation,XmVERTICAL);n++;
    XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNtopWidget,info_label);n++;
    XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
    XtSetArg(args[n],XmNleftPosition,i*30+50);n++;
    liss_sep[i] = XmCreateSeparatorGadget(form,"liss_sep",args,n);
}

n = 0;
XtSetArg(args[n],XmNorientation,XmVERTICAL);n++;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNtopWidget,info_label);n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM);n++;
liss_sep[4] = XmCreateSeparatorGadget(form,"liss_sep",args,n);

n = 0;
XtSetArg(args[n],XmNorientation,XmHORIZONTAL);n++;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNtopWidget,info_label);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNleftWidget,chart_canvas);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM);n++;
liss_sep[5] = XmCreateSeparatorGadget(form,"liss_sep",args,n);

n = 0;
XtSetArg(args[n],XmNorientation,XmHORIZONTAL);n++;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_POSITION);n++;
XtSetArg(args[n],XmNtopPosition,85);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNleftWidget,chart_canvas);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM);n++;
liss_sep[6] = XmCreateSeparatorGadget(form,"liss_sep",args,n);

n = 0;
XtSetArg(args[n],XmNorientation,XmHORIZONTAL);n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNleftWidget,chart_canvas);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM);n++;
liss_sep[7] = XmCreateSeparatorGadget(form,"liss_sep",args,n);

/* Create lissajous data labels */

for(i=0;i<8;i++) {
    sprintf(tmp,"%d,1",i);
    cstring = XmStringCreateLocalized(tmp);
    n = 0;

```

```

XtSetArg(args[n], XmNLabelString, cstring); n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
if(i < 4) {
    XtSetArg(args[n], XmNbottomWidget, liss_sep[6]); n++;
}
else {
    XtSetArg(args[n], XmNbottomWidget, liss_sep[7]); n++;
}
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNleftWidget, liss_sep[i%4]); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNrightWidget, liss_sep[i%4+1]); n++;
sprintf(tmp, "lisslabel%d1", i);
liss_data_label[i][1] = XmCreateLabelGadget(form, tmp, args, n);
XmStringFree(cstring);
}

for(i=0; i<8; i++) {
    sprintf(tmp, "Phase %d", i);
    cstring = XmStringCreateLocalized(tmp);
    n = 0;
    XtSetArg(args[n], XmNLabelString, cstring); n++;
    XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
    XtSetArg(args[n], XmNbottomWidget, liss_data_label[i][1]); n++;
    XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
    XtSetArg(args[n], XmNleftWidget, liss_sep[i%4]); n++;
    XtSetArg(args[n], XmNrightAttachment, XmATTACH_WIDGET); n++;
    XtSetArg(args[n], XmNrightWidget, liss_sep[i%4+1]); n++;
    sprintf(tmp, "lisslabel%d0", i);
    liss_data_label[i][0] = XmCreateLabelGadget(form, tmp, args, n);
    XmStringFree(cstring);
}

/* Create lissajous buttons */

for(i=0; i<8; i++) {
    sprintf(tmp, "Coil %d", i+1);
    cstring = XmStringCreateLocalized(tmp);
    n = 0;
    XtSetArg(args[n], XmNLabelString, cstring); n++;
    XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
    XtSetArg(args[n], XmNbottomWidget, liss_data_label[i][0]); n++;
    XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
    XtSetArg(args[n], XmNleftWidget, liss_sep[i%4]); n++;
    XtSetArg(args[n], XmNrightAttachment, XmATTACH_POSITION); n++;
    XtSetArg(args[n], XmNrightPosition, (i%4)*30+65); n++;
    sprintf(tmp, "lisscoil%d", i);
    liss_coil_button[i] = XmCreatePushButtonGadget(form, tmp, args, n);
    XmStringFree(cstring);
    XtAddCallback(liss_coil_button[i], XmNactivateCallback,
        (XtCallbackProc)liss_coil_buttonCB, NULL);
}

for(i=0; i<8; i++) {
    sprintf(tmp, " %d ", i+1);
    cstring = XmStringCreateLocalized(tmp);
    n = 0;
    XtSetArg(args[n], XmNLabelString, cstring); n++;
    XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
    XtSetArg(args[n], XmNbottomWidget, liss_data_label[i][0]); n++;
    XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
    XtSetArg(args[n], XmNleftWidget, liss_coil_button[i]); n++;
    XtSetArg(args[n], XmNrightAttachment, XmATTACH_WIDGET); n++;
    XtSetArg(args[n], XmNrightWidget, liss_sep[(i%4)+1]); n++;
    sprintf(tmp, "lissdatab%d", i);
    liss_data_button[i] = XmCreatePushButtonGadget(form, tmp, args, n);
    XmStringFree(cstring);
    XtAddCallback(liss_data_button[i], XmNactivateCallback,

```

```

        (XtCallbackProc)liss_data_buttonCB,NULL);
    }

/* Create lissajous drawing areas */

    n = 0;
    XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNtopWidget,liss_sep[5]);n++;
    XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNbottomWidget,liss_coil_button[0]);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNleftWidget,liss_sep[0]);n++;
    XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNrightWidget,liss_sep[1]);n++;
    XtSetArg(args[n],XmNrightPosition,80);n++;
    XtSetArg(args[n],XmNbackground,color.background);n++;
    liss_canvas[0] = XmCreateDrawingArea(form,"liss_canvas0",args,n);

    n = 0;
    XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNtopWidget,liss_sep[5]);n++;
    XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNbottomWidget,liss_coil_button[1]);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNleftWidget,liss_sep[1]);n++;
    XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNrightWidget,liss_sep[2]);n++;
    XtSetArg(args[n],XmNrightPosition,110);n++;
    XtSetArg(args[n],XmNbackground,color.background);n++;
    liss_canvas[1] = XmCreateDrawingArea(form,"liss_canvas1",args,n);

    n = 0;
    XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNtopWidget,liss_sep[5]);n++;
    XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNbottomWidget,liss_coil_button[2]);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNleftWidget,liss_sep[2]);n++;
    XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNrightWidget,liss_sep[3]);n++;
    XtSetArg(args[n],XmNrightPosition,140);n++;
    XtSetArg(args[n],XmNbackground,color.background);n++;
    liss_canvas[2] = XmCreateDrawingArea(form,"liss_canvas2",args,n);

    n = 0;
    XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNtopWidget,liss_sep[5]);n++;
    XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNbottomWidget,liss_coil_button[3]);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNleftWidget,liss_sep[3]);n++;
    XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNrightWidget,liss_sep[4]);n++;
    XtSetArg(args[n],XmNbackground,color.background);n++;
    liss_canvas[3] = XmCreateDrawingArea(form,"liss_canvas3",args,n);

    n = 0;
    XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNtopWidget,liss_sep[6]);n++;
    XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNbottomWidget,liss_coil_button[4]);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNleftWidget,liss_sep[0]);n++;
    XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNrightWidget,liss_sep[1]);n++;
    XtSetArg(args[n],XmNbackground,color.background);n++;
    liss_canvas[4] = XmCreateDrawingArea(form,"liss_canvas4",args,n);

```

```

n = 0;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNtopWidget, liss_sep[6]);n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNbottomWidget, liss_coil_button[5]);n++;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNleftWidget, liss_sep[1]);n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNrightWidget, liss_sep[2]);n++;
XtSetArg(args[n], XmNbackground, color.background);n++;
liss_canvas[5] = XmCreateDrawingArea(form, "liss_canvas5", args, n);

n = 0;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNtopWidget, liss_sep[6]);n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNbottomWidget, liss_coil_button[6]);n++;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNleftWidget, liss_sep[2]);n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNrightWidget, liss_sep[3]);n++;
XtSetArg(args[n], XmNbackground, color.background);n++;
liss_canvas[6] = XmCreateDrawingArea(form, "liss_canvas6", args, n);

n = 0;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNtopWidget, liss_sep[6]);n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNbottomWidget, liss_coil_button[7]);n++;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNleftWidget, liss_sep[3]);n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNrightWidget, liss_sep[4]);n++;
XtSetArg(args[n], XmNbackground, color.background);n++;
liss_canvas[7] = XmCreateDrawingArea(form, "liss_canvas7", args, n);

for(i=0; i<8; i++) {
    XtAddCallback(liss_canvas[i], XmNexposeCallback,
        (XtCallbackProc)liss_exposeCB, NULL);
}

XtAddCallback(liss_canvas[0], XmNresizeCallback,
    (XtCallbackProc)liss_resizeCB, NULL);

/* Create logo */

XtVaGetValues(infolabel,
    XmNheight, &logoheight,
    NULL);

logoheight *= 1.5;
logowidth = 54.*logoheight/21.;

logopixmap = XCreatePixmap(XtDisplay(form),
    RootWindowOfScreen(XtScreen(form)),
    logowidth, logoheight,
    DefaultDepthOfScreen(XtScreen(form)));

draw_logo();

n = 0;
XtSetArg(args[n], XmNlabelType, XmPIXMAP);n++;
XtSetArg(args[n], XmNlabelPixmap, logopixmap);n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM);n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM);n++;
/* XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET);n++;
    XtSetArg(args[n], XmNleftWidget, infolabel);n++;*/
logolabel = XmCreateLabelGadget(form, "logolabel", args, n);

```

```

XtManageChild(logolabel);

/* Set colors and attachments */

XtVaSetValues(infolabel,
               XmNrightAttachment,XmATTACH_WIDGET,
               XmNrightWidget,logolabel,
               XmNtopAttachment,XmATTACH_NONE,
               XmNbottomAttachment,XmATTACH_WIDGET,
               XmNbottomWidget,coil_button[0],
               XmNleftAttachment,XmATTACH_FORM,
               NULL);

XtVaSetValues(form,
               XmNbackground,color.background,
               XmNbottomShadowColor,color.black,
               XmNtopShadowColor,color.light,
               XmNhighlightColor,color.background,
               NULL);

for(i=0;i<16;i++) {
    XtVaSetValues(coil_button[i],
                  XmNarmColor,color.background,
                  XmNtopAttachment,XmATTACH_WIDGET,
                  XmNtopWidget,logolabel,
                  NULL);
    XtVaSetValues(data_button[i],
                  XmNarmColor,color.background,
                  NULL);
}

XtVaSetValues(chart_canvas,
               XmNbackground,color.background,
               NULL);

XtManageChild(chart_canvas);

XtRealizeWidget(toplevel);

attrs.bit_gravity = ForgetGravity;
XChangeWindowAttributes(XtDisplay(chart_canvas),
                        XtWindow(chart_canvas),CWBitGravity,&attrs);
for(i=0;i<8;i++) {
    XChangeWindowAttributes(XtDisplay(liss_canvas[i]),
                            XtWindow(liss_canvas[i]),
                            CWBitGravity,&attrs);
}

setup_types();
setup_chart_display();
setup_liss_display();

XtAppMainLoop(app);
}

void SetSpanCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    do_set_span();

    return;
}

void do_set_span(void)
{
    int i;
    int n;
    char tmp[80];
    XmString cstring;

```

```

if(mode != 2) {
    printf("Must be in 8 Lissajous mode.\n");
    return;
}

if(setting_span) {
    setting_span = 0;
}

/* Remove widgets for setting the span and reset the attachments. */

for(i=0;i<8;i++) {
    XtUnmanageChild(liss_data_label[i][1]);
    XtUnmanageChild(liss_data_label[i][0]);
    XtUnmanageChild(liss_coil_button[i]);
    XtUnmanageChild(liss_canvas[i]);
    if(setting_rotation) {
        XtUnmanageChild(rotation_label[i]);
        XtUnmanageChild(rotation_up[i]);
        XtUnmanageChild(rotation_down[i]);
    }
    XtDestroyWidget(span_label[i]);
    XtDestroyWidget(span_up[i]);
    XtDestroyWidget(span_down[i]);
}

/* Reset attachments */

if(setting_rotation) {
    XtVaSetValues(liss_data_label[i][1],
                  XmNbottomWidget,rotation_label[i],
                  NULL);
}
else {
    if(i < 4) {
        XtVaSetValues(liss_data_label[i][1],
                      XmNbottomWidget,liss_sep[6],
                      NULL);
    }
    else {
        XtVaSetValues(liss_data_label[i][1],
                      XmNbottomWidget,liss_sep[7],
                      NULL);
    }
}

if(setting_rotation) {
    XtManageChild(rotation_down[i]);
    XtManageChild(rotation_up[i]);
    XtManageChild(rotation_label[i]);
}
XtManageChild(liss_data_label[i][1]);
XtManageChild(liss_data_label[i][0]);
XtManageChild(liss_coil_button[i]);
XtManageChild(liss_data_button[i]);
XtManageChild(liss_canvas[i]);
}
}
else {
    setting_span = 1;
    for(i=0;i<8;i++) {
        /* Remove lissajous widgets */

        if(setting_rotation) {
            XtUnmanageChild(rotation_down[i]);
            XtUnmanageChild(rotation_up[i]);
            XtUnmanageChild(rotation_label[i]);
        }
        XtUnmanageChild(liss_data_label[i][1]);
        XtUnmanageChild(liss_data_label[i][0]);
    }
}

```

```

XtUnmanageChild(liss_coil_button[i]);
XtUnmanageChild(liss_data_button[i]);
XtUnmanageChild(liss_canvas[i]);

/* Create span widgets */

n = 0;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
if(setting_rotation) {
    XtSetArg(args[n],XmNbottomWidget,rotation_label[i]);n++;
}
else {
    if(i < 4) {
        XtSetArg(args[n],XmNbottomWidget,liss_sep[6]);n++;
    }
    else {
        XtSetArg(args[n],XmNbottomWidget,liss_sep[7]);n++;
    }
}
XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNrightWidget,liss_sep[i%4+1]);n++;
XtSetArg(args[n],XmNarrowDirection,XmARROW_DOWN);n++;
span_down[i] = XmCreateArrowButtonGadget(form,"spandown",args,n);
XtAddCallback(span_down[i],XmNactivateCallback,
               (XtCallbackProc)ChangeSpanCB,NULL);

n = 0;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
if(setting_rotation) {
    XtSetArg(args[n],XmNbottomWidget,rotation_label[i]);n++;
}
else {
    if(i < 4) {
        XtSetArg(args[n],XmNbottomWidget,liss_sep[6]);n++;
    }
    else {
        XtSetArg(args[n],XmNbottomWidget,liss_sep[7]);n++;
    }
}
XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNrightWidget,span_down[i]);n++;
XtSetArg(args[n],XmNarrowDirection,XmARROW_UP);n++;
span_up[i] = XmCreateArrowButtonGadget(form,"spanup",args,n);
XtAddCallback(span_up[i],XmNactivateCallback,
               (XtCallbackProc)ChangeSpanCB,NULL);

sprintf(tmp,"Span: %.2f",
        DataType[lissajous[i].coil-1][lissajous[i].type].span);
cstring = XmStringCreateLocalized(tmp);
n = 0;
XtSetArg(args[n],XmNlabelString,cstring);n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_WIDGET);n++;
if(setting_rotation) {
    XtSetArg(args[n],XmNbottomWidget,rotation_label[i]);n++;
}
else {
    if(i < 4) {
        XtSetArg(args[n],XmNbottomWidget,liss_sep[6]);n++;
    }
    else {
        XtSetArg(args[n],XmNbottomWidget,liss_sep[7]);n++;
    }
}
XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNleftWidget,liss_sep[i%4]);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNrightWidget,span_up[i]);n++;
span_label[i] = XmCreateLabelGadget(form,"spanlabel",args,n);

```

```

        XmStringFree(cstring);

/* Set attachments */

        XtVaSetValues(liss_data_label[i][1],
                      XmNbottomWidget,span_label[i],
                      NULL);

/* Manage lissajous widgets */

        if(setting_rotation) {
            XtManageChild(rotation_down[i]);
            XtManageChild(rotation_up[i]);
            XtManageChild(rotation_label[i]);
        }
        XtManageChild(span_down[i]);
        XtManageChild(span_up[i]);
        XtManageChild(span_label[i]);
        XtManageChild(liss_data_label[i][1]);
        XtManageChild(liss_data_label[i][0]);
        XtManageChild(liss_coil_button[i]);
        XtManageChild(liss_data_button[i]);
        XtManageChild(liss_canvas[i]);
    }
}

return;
}

void SetRotationCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    do_set_rotation();

    return;
}

void do_set_rotation(void)
{
    int i;
    int n;
    char tmp[80];
    XmString cstring;

    if(mode != 2) {
        printf("Must be in 8 Lissajous mode.\n");
        return;
    }

    if(setting_rotation) {
        setting_rotation = 0;
    }

/* Remove widgets for setting the rotation and reset the attachments. */

    for(i=0;i<8;i++) {
        XtUnmanageChild(liss_data_label[i][1]);
        XtUnmanageChild(liss_data_label[i][0]);
        XtUnmanageChild(liss_coil_button[i]);
        XtUnmanageChild(liss_data_button[i]);
        XtUnmanageChild(liss_canvas[i]);
        if(setting_span) {
            XtUnmanageChild(span_label[i]);
            XtUnmanageChild(span_up[i]);
            XtUnmanageChild(span_down[i]);
        }
        XtDestroyWidget(rotation_label[i]);
        XtDestroyWidget(rotation_up[i]);
        XtDestroyWidget(rotation_down[i]);
    }
}

```



```
/* Reset attachments */
```

```

if(setting_span) {
    if(i < 4) {
        XtVaSetValues(span_label[i],
                      XmNbottomWidget,liss_sep[6],
                      NULL);
        XtVaSetValues(span_up[i],
                      XmNbottomWidget,liss_sep[6],
                      NULL);
        XtVaSetValues(span_down[i],
                      XmNbottomWidget,liss_sep[6],
                      NULL);
    }
    else {
        XtVaSetValues(span_label[i],
                      XmNbottomWidget,liss_sep[7],
                      NULL);
        XtVaSetValues(span_up[i],
                      XmNbottomWidget,liss_sep[7],
                      NULL);
        XtVaSetValues(span_down[i],
                      XmNbottomWidget,liss_sep[7],
                      NULL);
    }
}
else {
    if(i < 4) {
        XtVaSetValues(liss_data_label[i][1],
                      XmNbottomWidget,liss_sep[6],
                      NULL);
    }
    else {
        XtVaSetValues(liss_data_label[i][1],
                      XmNbottomWidget,liss_sep[7],
                      NULL);
    }
}
if(setting_span) {
    XtManageChild(span_down[i]);
    XtManageChild(span_up[i]);
    XtManageChild(span_label[i]);
}
XtManageChild(liss_data_label[i][1]);
XtManageChild(liss_data_label[i][0]);
XtManageChild(liss_coil_button[i]);
XtManageChild(liss_data_button[i]);
XtManageChild(liss_canvas[i]);
}
else {
    setting_rotation = 1;
    for(i=0;i<8;i++) {

```

```
/* Remove lissajous widgets */
```

```

XtUnmanageChild(liss_data_label[i][1]);
XtUnmanageChild(liss_data_label[i][0]);
XtUnmanageChild(liss_coil_button[i]);
XtUnmanageChild(liss_data_button[i]);
XtUnmanageChild(liss_canvas[i]);

if(setting_span) {
    XtUnmanageChild(span_label[i]);
    XtUnmanageChild(span_up[i]);
    XtUnmanageChild(span_down[i]);
}

```

```
/* Create rotation widgets */
```

```

n = 0;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET);n++;
if(i < 4) {
    XtSetArg(args[n], XmNbottomWidget, liss_sep[6]);n++;
}
else {
    XtSetArg(args[n], XmNbottomWidget, liss_sep[7]);n++;
}
XtSetArg(args[n], XmNrightAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNrightWidget, liss_sep[i%4+1]);n++;
XtSetArg(args[n], XmNarrowDirection, XmARROW_DOWN);n++;
rotation_down[i] = XmCreateArrowButtonGadget(form, "rotationdown", args, n);
XtAddCallback(rotation_down[i], XmNactivateCallback,
               (XtCallbackProc)ChangeRotationCB, NULL);

```

```

n = 0;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET);n++;
if(i < 4) {
    XtSetArg(args[n], XmNbottomWidget, liss_sep[6]);n++;
}
else {
    XtSetArg(args[n], XmNbottomWidget, liss_sep[7]);n++;
}
XtSetArg(args[n], XmNrightAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNrightWidget, rotation_down[i]);n++;
XtSetArg(args[n], XmNarrowDirection, XmARROW_UP);n++;
rotation_up[i] = XmCreateArrowButtonGadget(form, "rotationup", args, n);
XtAddCallback(rotation_up[i], XmNactivateCallback,
               (XtCallbackProc)ChangeRotationCB, NULL);

```

```

sprintf(tmp, "Rotation: %.0f",
        DataType[lissajous[i].coil-1][lissajous[i].type].rotation);
cstring = XmStringCreateLocalized(tmp);
n = 0;
XtSetArg(args[n], XmNlabelString, cstring);n++;
XtSetArg(args[n], XmNbottomAttachment, XmATTACH_WIDGET);n++;
if(i < 4) {
    XtSetArg(args[n], XmNbottomWidget, liss_sep[6]);n++;
}
else {
    XtSetArg(args[n], XmNbottomWidget, liss_sep[7]);n++;
}
XtSetArg(args[n], XmNleftAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNleftWidget, liss_sep[i%4]);n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_WIDGET);n++;
XtSetArg(args[n], XmNrightWidget, rotation_up[i]);n++;
rotation_label[i] = XmCreateLabelGadget(form, "rotationlabel", args, n);
XmStringFree(cstring);

```

```
/* Set attachments */
```

```

if(setting_span) {
    XtVaSetValues(span_label[i],
                  XmNbottomWidget, rotation_label[i],
                  NULL);
    XtVaSetValues(span_up[i],
                  XmNbottomWidget, rotation_label[i],
                  NULL);
    XtVaSetValues(span_down[i],
                  XmNbottomWidget, rotation_label[i],
                  NULL);
}
else {
    XtVaSetValues(liss_data_label[i][1],
                  XmNbottomWidget, rotation_label[i],
                  NULL);
}

```

```

    }

/* Manage lissajous widgets */

    XtManageChild(rotation_down[i]);
    XtManageChild(rotation_up[i]);
    XtManageChild(rotation_label[i]);
    if(setting_span) {
        XtManageChild(span_down[i]);
        XtManageChild(span_up[i]);
        XtManageChild(span_label[i]);
    }
    XtManageChild(liss_data_label[i][1]);
    XtManageChild(liss_data_label[i][0]);
    XtManageChild(liss_coil_button[i]);
    XtManageChild(liss_data_button[i]);
    XtManageChild(liss_canvas[i]);
}
}

return;
}

void ChangeRotationCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int i;
    int ii = -1;
    int type;
    float delta;
    int dir;
    char tmp[25];
    XmString cstring;

    delta = 3.1415927/180.;

    for(i=0;i<8;i++) {
        if(w == rotation_up[i]) {
            dir = 1;
            ii = i;
            break;
        }
        else if(w == rotation_down[i]) {
            dir = -1;
            ii = i;
            break;
        }
    }

    if(ii < 0) {
        printf("Error in ChangeRotationCB.\n");
        return;
    }

    type = lissajous[ii].type;

    for(i=0;i<16;i++) {
        DataType[i][type].rotation += dir*delta;
        if(DataType[i][type].rotation < 0.) {
            DataType[i][type].rotation += 2.*3.1415927;
        }
        else if(DataType[i][type].rotation > 2.*3.1415927) {
            DataType[i][type].rotation -= 2.*3.1415927;
        }
    }

    for(i=0;i<8;i++) {
        #if(0)
            XClearWindow(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]));

```

```

#endif
    sprintf(tmp,"Rotation %.0f",
        DataType[lissajous[i].coil-1][lissajous[i].type].rotation*180./3.1415927);
    cstring = XmStringCreateLocalized(tmp);
    XtVaSetValues(rotation_label[i],
        XmNlabelString,cstring,
        NULL);
    XmStringFree(cstring);
}

return;
}

void ChangeSpanCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int i;
    int ii = -1;
    int dir;
    int type;
    float delta = 0.1;
    char tmp[80];
    XmString cstring;

    for(i=0;i<8;i++) {
        if(w == span_up[i]) {
            dir = 1;
            ii = i;
            break;
        }
        else if(w == span_down[i]) {
            dir = -1;
            ii = i;
            break;
        }
    }

    if(ii < 0) {
        printf("Error in ChangeSpanCB.\n");
        return;
    }

    type = lissajous[ii].type;

    for(i=0;i<16;i++) {
        DataType[i][type].span += dir*delta;
    }

    for(i=0;i<8;i++) {
        XClearWindow(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]));
        sprintf(tmp,"Span %.2f",
            DataType[lissajous[i].coil-1][lissajous[i].type].span);
        cstring = XmStringCreateLocalized(tmp);
        XtVaSetValues(span_label[i],
            XmNlabelString,cstring,
            NULL);
        XmStringFree(cstring);
    }

    XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
    draw_liss_data();

    return;
}

void setup_types(void)
{
    int i,j;

    for(i=0;i<16;i++) {
        for(j=0;j<4;j++) {

```

```

        DataType[i][j].active = 1;
        DataType[i][j].un = 1;
        DataType[i][j].freqind = j;
        DataType[i][j].mix = 0;
        DataType[i][j].property = 0;
        DataType[i][j].rotation = 0.;
    }
}

for(i=0;i<16;i++) {
    for(j=4;j<8;j++) {
        DataType[i][j].active = 0;
        DataType[i][j].un = 0;
        DataType[i][j].mix = 1;
        DataType[i][j].mixind = j - 4;
        DataType[i][j].property = 0;
        DataType[i][j].rotation = 0.;
    }
}

for(i=0;i<16;i++) {
    DataType[i][8].active = 0;
    DataType[i][8].un = 0;
    DataType[i][8].mix = 0;
    DataType[i][8].property = 1;
}

lasttype = 8;
lastactivetype = 3;

return;
}

void coil_buttonCB(Widget w, XtPointer client_data,
XmPushButtonCallbackStruct *cbs)
{
    XEvent *event = cbs->event;
    int i;
    char string[20];
    XmString cstring;

    for(i=0;i<ncharts;i++) {
        if(coil_button[i] == w) {

/* Erase chart data */

            if(file_open) {
#iif(0)
                XSetForeground(XtDisplay(chart_canvas),gc,color.background);
                draw_one_chart_data(i);
#endif
                erase_chart_data(i);
            }
            if(event->xbutton.state & ShiftMask) {
                chart[i].coil--;
                if(chart[i].coil < 1) {
                    chart[i].coil = 16;
                }
            }
            else {
                chart[i].coil++;
                if(chart[i].coil > 16) {
                    chart[i].coil = 1;
                }
            }
            sprintf(string,"Coil %d",chart[i].coil);
            cstring = XmStringCreateSimple(string);
            XtVaSetValues(coil_button[i],

```

```

        XmNlabelString,cstring,
        NULL);
        XmStringFree(cstring);

/* Redraw data */

        if(file_open) {
            XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
            draw_one_chart_data(i);
        }

        return;
    }
}

printf("Error in coil_buttonCB.\n");

return;
}

void erase_chart_data(int i)
{
    if(cursor_y != -1) {
        XSetForeground(XtDisplay(chart_canvas),gc,color.cursor);
        XSetFunction(XtDisplay(chart_canvas),gc,GXxor);
        XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,
            0,cursor_y,chart_area_width,cursor_y);
        cursor_y = -1;
        XSetFunction(XtDisplay(chart_canvas),gc,GXcopy);
    }

    if((spt1 != -1) || (sspt != -1)) {
        XSetForeground(XtDisplay(chart_canvas),gc,color.background^color.light);
        XSetFunction(XtDisplay(chart_canvas),gc,GXxor);
        XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,
            0,sy1,chart_area_width,sy1);
        XSetFunction(XtDisplay(chart_canvas),gc,GXcopy);
        spt1 = -1;
        sspt = -1;
    }

    if(spt2 != -1) {
        XSetForeground(XtDisplay(chart_canvas),gc,color.background^color.light);
        XSetFunction(XtDisplay(chart_canvas),gc,GXxor);
        XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,
            0,sy2,chart_area_width,sy2);
        XSetFunction(XtDisplay(chart_canvas),gc,GXcopy);
        spt2 = -1;
    }

    XClearArea(XtDisplay(chart_canvas),XtWindow(chart_canvas),
        chart[i].llim,0,
        chart[i].ulim-chart[i].llim+1,chart_area_height,
        False);

    XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);

    return;
}

void data_buttonCB(Widget w, XtPointer client_data,
    XmPushButtonCallbackStruct *cbs)
{
    XEvent *event = cbs->event;
    int i;
    char string[20];
    XmString cstring;

```

```

for(i=0;i<ncharts;i++) {
    if(data_button[i] == w) {

/* Erase chart data */

        if(file_open) {
#ifdef 0
            XSetForeground(XtDisplay(chart_canvas),gc,color.background);
            draw_one_chart_data(i);
#endif
            erase_chart_data(i);
        }
        if(event->xbutton.state & ShiftMask) {
            if(chart[i].vertical_flag == 1) {
                chart[i].vertical_flag = 0;
            }
            else {
                chart[i].vertical_flag = 1;
                do {
                    chart[i].type--;
                    if(chart[i].type < 0) {
                        chart[i].type = lastactivetype;
                    }
                } while(DataType[0][chart[i].type].active != 1);
                if(DataType[chart[i].coil-1][chart[i].type].property == 1) {
                    chart[i].vertical_flag = 0;
                }
            }
        }
        else {
            if((chart[i].vertical_flag == 0) &&
                (DataType[chart[i].coil-1][chart[i].type].property == 0)) {
                chart[i].vertical_flag = 1;
            }
            else {
                chart[i].vertical_flag = 0;
                do {
                    chart[i].type++;
                    if(chart[i].type > lastactivetype) {
                        chart[i].type = 0;
                    }
                } while(DataType[0][chart[i].type].active != 1);
            }
        }
        if(DataType[chart[i].coil-1][chart[i].type].property == 0) {
            if(file_open) {
                if(DataType[chart[i].coil-1][chart[i].type].un) {
                    sprintf(string,"%d ",
                        DataType[chart[i].coil-1][chart[i].type].frequency);
                }
                else if(DataType[chart[i].coil-1][chart[i].type].mix) {
                    sprintf(string,"Mix %d",
                        DataType[chart[i].coil-1][chart[i].type].mixind);
                }
            }
            else {
                sprintf(string,"Frq%d ",chart[i].type + 1);
            }
            if(chart[i].vertical_flag) {
                strcat(string,"V");
            }
            else {
                strcat(string,"H");
            }
        }
        else if(DataType[chart[i].coil-1][chart[i].type].property == 1) {
            strcpy(string,"DfDep");
        }
    }
}

```

```

        else {
            printf("Error in data_buttonCB.\n");
        }
        cstring = XmStringCreateSimple(string);
        XtVaSetValues(data_button[i],
                      XmNlabelString,cstring,
                      NULL);
        XmStringFree(cstring);
    /* Redraw data */

    if(file_open) {
        XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
        draw_one_chart_data(i);
    }

    return;
}

printf("Error in data_buttonCB.\n");
return;
}

void liss_coil_buttonCB(Widget w, XtPointer client_data,
                      XmPushButtonCallbackStruct *cbs)
{
    XEvent *event = cbs->event;
    int i;
    char string[20];
    XmString cstring;

    for(i=0;i<8;i++) {
        if(liss_coil_button[i] == w) {
    /* Erase lissajous data */

        if(file_open) {
            XClearWindow(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]));
        }
        #if(0)
            XSetForeground(XtDisplay(liss_canvas[i]),gc,color.background);
            draw_one_liss_data(i);
        #endif
    }
        if(event->xbutton.state & ShiftMask) {
            lissajous[i].coil--;
            if(lissajous[i].coil < 1) {
                lissajous[i].coil = 16;
            }
        }
        else {
            lissajous[i].coil++;
            if(lissajous[i].coil > 16) {
                lissajous[i].coil = 1;
            }
        }
        sprintf(string,"Coil %d",lissajous[i].coil);
        cstring = XmStringCreateSimple(string);
        XtVaSetValues(liss_coil_button[i],
                      XmNlabelString,cstring,
                      NULL);
        XmStringFree(cstring);
    /* Redraw data */

    if(file_open) {
        XSetForeground(XtDisplay(liss_canvas[i]),gc,color.foreground);

```



```

        draw_one_liss_data(i);
        update_liss_labels(i);
    }

    return;
}

printf("Error in liss_coil_buttonCB.\n");

return;
}

void liss_data_buttonCB(Widget w, XtPointer client_data,
                        XmPushButtonCallbackStruct *cbs)
{
    XEvent *event = cbs->event;
    int i;
    char string[20];
    XmString cstring;

    for(i=0;i<8;i++) {
        if(liss_data_button[i] == w) {
/* Erase lissajous data */

            if(file_open) {
                XSetForeground(XtDisplay(chart_canvas),gc,color.background);
                draw_one_liss_data(i);
            }
            if(event->xbutton.state & ShiftMask) {
                do {
                    lissajous[i].type--;
                    if(lissajous[i].type < 0) {
                        lissajous[i].type = lastactivetype;
                    }
                } while((DataType[0][lissajous[i].type].active != 1)
                        || (DataType[0][lissajous[i].type].property != 0));
            }
            else {
                do {
                    lissajous[i].type++;
                    if(lissajous[i].type > lastactivetype) {
                        lissajous[i].type = 0;
                    }
                } while((DataType[0][lissajous[i].type].active != 1)
                        || (DataType[0][lissajous[i].type].property != 0));
            }
            if(file_open) {
                if(DataType[lissajous[i].coil-1][lissajous[i].type].un) {
                    sprintf(string,"%d kHz",
                        DataType[lissajous[i].coil-1][lissajous[i].type].frequency);
                }
                else if(DataType[lissajous[i].coil-1][lissajous[i].type].mix) {
                    sprintf(string,"Mix %d",
                        DataType[lissajous[i].coil-1][lissajous[i].type].mixind);
                }
                else {
                    printf("Error in label_liss_buttons.\n");
                }
            }
            else {
                sprintf(string,"Freq %d",lissajous[i].type + 1);
            }
            cstring = XmStringCreateSimple(string);
            XtVaSetValues(liss_data_button[i],
                        XmNlabelString,cstring,
                        NULL);

```

```

        XmStringFree(cstring);

/* Redraw data */

        if(file_open) {
            XSetForeground(XtDisplay(liss_canvas[i]),gc,color.foreground);
            draw_one_liss_data(i);
            update_liss_labels(i);
        }

        return;
    }
}

printf("Error in liss_data_buttonCB.\n");

return;
}

void update_liss_labels(int ii)
{
    char tmp[20];
    XmString cstring;
    float delx,dely;
    float d2,d2max;
    float x1,y1,x2,y2;
    float xtmp;
    float mag1,mag2;
    float angle;
    int bp1,bp2;
    int i,j,k;
    int istart,istop;
    int lag;

    if(ii < 0) {
        istart = 0;
        istop = 7;
    }
    else {
        istart = istop = ii;
    }

    for(i=istart;i<=istop;i++) {
        if(lissajous[i].coil < 9) {
            lag = nlag;
        }
        else {
            lag = 0;
        }
        delx = DataType[lissajous[i].coil-1][lissajous[i].type].hdata[(lpt1+lag)%npts] -
            DataType[lissajous[i].coil-1][lissajous[i].type].hdata[(lpt1+1+lag)%npts];
        dely = DataType[lissajous[i].coil-1][lissajous[i].type].vdata[(lpt1+lag)%npts] -
            DataType[lissajous[i].coil-1][lissajous[i].type].vdata[(lpt1+1+lag)%npts];
        d2max = delx*delx + dely*dely;
        bp1 = lpt1;
        bp2 = lpt1 + 1;

        for(j=lpt1+1;j<lpt2;j++) {
            for(k=j+1;k<lpt2;k++) {
                delx = DataType[lissajous[i].coil-1][lissajous[i].type].hdata[(j+lag)%npts] -
                    DataType[lissajous[i].coil-1][lissajous[i].type].hdata[(k+lag)%npts];
                dely = DataType[lissajous[i].coil-1][lissajous[i].type].vdata[(j+lag)%npts] -
                    DataType[lissajous[i].coil-1][lissajous[i].type].vdata[(k+lag)%npts];
                d2 = delx*delx + dely*dely;
                if(d2 > d2max) {
                    d2max = d2;
                    bp1 = j;
                    bp2 = k;
                }
            }
        }
    }
}

```

```

    }
}
}
sprintf(tmp, "Vpp: %5.2f", sqrt(d2max));
cstring = XmStringCreateLocalized(tmp);
XtVaSetValues(liss_data_label[i][0],
               XmNlabelString, cstring,
               NULL);
XmStringFree(cstring);

x1 = DataType[lissajous[i].coil-1][lissajous[i].type].hdata[(bp1+lag)%npts];
y1 = DataType[lissajous[i].coil-1][lissajous[i].type].vdata[(bp1+lag)%npts];
x2 = DataType[lissajous[i].coil-1][lissajous[i].type].hdata[(bp2+lag)%npts];
y2 = DataType[lissajous[i].coil-1][lissajous[i].type].vdata[(bp2+lag)%npts];
mag1 = x1*x1 + y1*y1;
mag2 = x2*x2 + y2*y2;
if(mag1 > mag2) {
    xtmp = x1;
    x1 = x2;
    x2 = xtmp;
    ytmp = y1;
    y1 = y2;
    y2 = ytmp;
}
delx = x2 - x1;
dely = y2 - y1;

angle = 180.*
    (atan2(dely, delx) +
     DataType[lissajous[i].coil-1][lissajous[i].type].rotation)
    /3.1415927;

while(angle < 0.) {
    angle += 360.;
}
while(angle > 360.) {
    angle -= 360.;
}

sprintf(tmp, "Phase: %5.0f", angle);
cstring = XmStringCreateLocalized(tmp);
XtVaSetValues(liss_data_label[i][1],
               XmNlabelString, cstring,
               NULL);
XmStringFree(cstring);
}

return;
}

void get_colors(Widget w)
{
    int default_depth;
    Visual *default_visual;
    static char *name[] = {"Red", "Gray50", "Gray40", "Gray70",
                           "MidnightBlue", "Blue", "MediumTurquoise", "green2",
                           "GreenYellow",
                           "Yellow", "Orange",
                           "Gray12", "Gray25", "Gray37", "Gray62",
                           "Gray75", "Gray88"};

    XColor exact_def;
    Colormap default_cmap;
    int ncolors = 0;
    unsigned long colors[17];
    int i = 5;
    XVisualInfo visual_info;

    default_depth = DefaultDepthOfScreen(XtScreen(w));

```

```

default_visual = DefaultVisualOfScreen(XtScreen(w));
default_cmap = DefaultColormapOfScreen(XtScreen(w));
if(default_depth == 1) {
    color.background = BlackPixelOfScreen(XtScreen(w));
    color.foreground = WhitePixelOfScreen(XtScreen(w));
    color.red = color.foreground;
    color.medium = color.foreground;
    color.black = color.foreground;
    color.highlight = color.background;
    color.cursor = color.foreground;
    return;
}

while(!XMatchVisualInfo(XtDisplay(w),
    XScreenNumberOfScreen(XtScreen(w)),
    default_depth, i--,
    &visual_info));

/* printf("Found a class %d visual at default depth.\n", ++i);*/

if(i < StaticColor) {
    color.background = BlackPixelOfScreen(XtScreen(w));
    color.foreground = WhitePixelOfScreen(XtScreen(w));
    color.red = color.foreground;
    color.medium = color.foreground;
    color.black = color.foreground;
    color.highlight = color.background;
    color.cursor = color.foreground;
    return;
}

if(visual_info.visual != default_visual) {
    printf("Class visual at default depth is not default visual.\n");
    printf("Continuing anyway\n");
}

for(i=0; i<17; i++) {
/* printf("Allocating %s\n", name[i]);*/
    if(!XParseColor(XtDisplay(w), default_cmap, name[i], &exact_def)) {
        printf("Color name %s not in database.", name[i]);
        exit(-1);
    }
/* printf("The rgb values from the database are %d %d %d\n",
    exact_def.red, exact_def.green, exact_def.medium);*/
    if(!XAllocColor(XtDisplay(w), default_cmap, &exact_def)) {
        printf("Cannot allocate color:\n");
        printf("All colorcells allocated and no matching cells found\n");
        exit(-1);
    }
/* printf("The rgb values actually allocated are %d, %d, %d\n",
    exact_def.red, exact_def.green, exact_def.medium);*/
    colors[i] = exact_def.pixel;
    ncolors++;
}
/* printf("Allocated %d read-only color cells\n", ncolors);*/
color.black = BlackPixelOfScreen(XtScreen(w));
color.foreground = WhitePixelOfScreen(XtScreen(w));
color.red = colors[0];
color.medium = colors[1];
color.background = colors[2];
color.light = colors[3];
color.highlight = color.background ^ colors[3];
color.cursor = color.background ^ color.red;
dp_colors[0][0] = colors[11];
dp_colors[0][1] = colors[12];
dp_colors[0][2] = colors[13];
dp_colors[0][3] = colors[11];
dp_colors[0][4] = colors[14];

```

```

    dp_colors[0][5] = colors[15];
    dp_colors[0][6] = colors[16];
    dp_colors[0][7] = color.foreground;
    dp_colors[1][0] = colors[4];
    dp_colors[1][1] = colors[5];
    dp_colors[1][2] = colors[6];
    dp_colors[1][3] = colors[7];
    dp_colors[1][4] = colors[8];
    dp_colors[1][5] = colors[9];
    dp_colors[1][6] = colors[10];
    dp_colors[1][7] = colors[0];

    return;
}

void create_gc(Widget w)
{
    XGCValues gcv;

    gcv.foreground = color.foreground;

    gc = XCreateGC(XtDisplay(w),
                   RootWindowOfScreen(XtScreen(w)),
                   GCForeground,&gcv);

    return;
}

void chartCB(Widget widget, XEvent *event, String *args, int *num_args)
{
    XmString cstring;
    char tmp[80];
    float val;
    float angle;
    int i;
    int lag;

    if(event->type == MotionNotify) {
        XSetForeground(XtDisplay(chart_canvas),gc,color.cursor);
        XSetFunction(XtDisplay(chart_canvas),gc,GXxor);
        /* Erase cursor at previous location */
        if(cursor_y != -1) {
            XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,
                      0,cursor_y,chart_area_width,cursor_y);
        }
        cursor_y = event->xmotion.y;
        XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,
                  0,cursor_y,chart_area_width,cursor_y);
        XSetFunction(XtDisplay(chart_canvas),gc,GXcopy);
    }
    else if(event->type == ButtonPress) {
        printf("Button press\n");
        /* Erase selection lines */
        if(spt1 != -1) {
            XSetForeground(XtDisplay(chart_canvas),gc,
                          color.background^color.light);
            XSetFunction(XtDisplay(chart_canvas),gc,GXxor);
            XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,
                      0,sy1,chart_area_width,sy1);
            XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,
                      0,sy2,chart_area_width,sy2);
            XSetFunction(XtDisplay(chart_canvas),gc,GXcopy);
        }
        if(sspt != -1) {
            XSetForeground(XtDisplay(chart_canvas),gc,
                          color.background^color.light);
            XSetFunction(XtDisplay(chart_canvas),gc,GXxor);
            XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,

```

```

        0,sy1,chart_area_width,sy1);
XSetFunction(XtDisplay(chart_canvas),gc,GXcopy);
/* Erase data labels */
for(i=0;i<16;i++) {
    sprintf(tmp," ");
    cstring = XmStringCreateSimple(tmp);
    XtVaSetValues(data_label[i],
        XmNlabelString,cstring,
        NULL);
    XmStringFree(cstring);
}
}
spt2 = -1;
sspt = -1;
spt1 = cpt1
    + (cpt2-cpt1) * ((float)event->xbutton.y)/chart_area_height;
sy1 = ((float)(spt1 - cpt1))/(cpt2 - cpt1) * chart_area_height;
XSetForeground(XtDisplay(chart_canvas),gc,
    color.background^color.light);
XSetFunction(XtDisplay(chart_canvas),gc,GXxor);
XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,
    0,sy1,chart_area_width,sy1);
XSetFunction(XtDisplay(chart_canvas),gc,GXcopy);
}
else if(event->type == ButtonRelease) {
    printf("Button release\n");
    spt2 = cpt1
        + (cpt2 - cpt1) * ((float)event->xbutton.y)/chart_area_height;
    if(spt2 > cpt2) {
        spt2 = cpt2;
    }
    else if(spt2 < 0) {
        spt2 = 0;
    }
    if(spt2 == spt1) {
        sspt = spt2;
        spt1 = spt2 = -1;
        for(i=0;i<16;i++) {
            if(chart[i].coil < 9) {
                lag = nlag;
            }
            else {
                lag = 0;
            }
            if(DataType[chart[i].coil-1][chart[i].type].property == 1) {
                val = DataType[chart[i].coil-1][chart[i].type].hdata[(sspt+lag)%npts];
                sprintf(tmp,"%d%%",(int)(val+0.5));
                cstring = XmStringCreateSimple(tmp);
                XtVaSetValues(data_label[i],
                    XmNlabelString,cstring,
                    NULL);
                XmStringFree(cstring);
            }
            else if(DataType[chart[i].coil-1][chart[i].type].property == 0) {
                if(chart[i].vertical_flag) {
                    angle = 90. * 3.1415927/180.;
                }
                else {
                    angle = 0.;
                }
                val = DataType[chart[i].coil-1][chart[i].type].hdata[(sspt+lag)%npts]
                    * cos(angle)
                    + DataType[chart[i].coil-1][chart[i].type].vdata[(sspt+lag)%npts]
                    * sin(angle);
                sprintf(tmp,"%4.1fv",val);
                cstring = XmStringCreateSimple(tmp);
                XtVaSetValues(data_label[i],
                    XmNlabelString,cstring,

```

```

        NULL);
        XmStringFree(cstring);
    }
}
}
else {
    sy2 = ((float)(spt2 - cpt1))/(cpt2 - cpt1) * chart_area_height;
    XSetForeground(XtDisplay(chart_canvas),gc,
        color.background^color.light);
    XSetFunction(XtDisplay(chart_canvas),gc,GXxor);
    XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,
        0,sy2,chart_area_width,sy2);
    XSetFunction(XtDisplay(chart_canvas),gc,GXcopy);
    printf("%d %d\n",spt1,spt2);
    printf("%d %d\n",sy1,sy2);
    if(mode == 2) {
/* Clear lissajous drawing areas */
        for(i=0;i<8;i++) {
            XClearWindow(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]));
        }
        lpt1 = spt1;
        lpt2 = spt2;
        XSetForeground(XtDisplay(liss_canvas[0]),gc,color.foreground);
        draw_liss_data();
        update_liss_labels(-1);
    }
}
}
return;
}

void FileOpenCB(Widget w, XtPointer client_data, XtPointer call_data)
{
    static Widget dialog;

    if(file_open_dialog_exists) {
        XRaiseWindow(XtDisplay(dialog),XtWindow(dialog));
    }
    else {
        dialog = XmCreateFileSelectionDialog(toplevel,"filesb",NULL,0);
        file_open_dialog_exists = 1;
        XtAddCallback(dialog,XmNokCallback,
            (XtCallbackProc)read_data,NULL);
        XtAddCallback(dialog,XmNcancelCallback,FileOpenCancelCB,NULL);

        XtManageChild(dialog);
    }

    return;
}

void FileOpenCancelCB(Widget w,XtPointer client_data, XtPointer call_data)
{
    XtDestroyWidget(w);
    file_open_dialog_exists = 0;

    return;
}

void read_data(Widget w,XtPointer client_data,
    XmFileSelectionBoxCallbackStruct *cbs)
{
    struct {
        int coil;
        int type;
    } channel[64];

```

```

FILE *strin;
char *filename;
int i,j,k,l;
short datr, dati;
int eoftst;
int char1;
size_t nalloc = 10000;
int file_type;
long time_date;
char free_space0[5];
char id_name[5];
char row_name[5];
char col_name[5];
char free_space1[21];
int delay;
char free_space2[17];
int num_chans;
int bytesinslice;
char free_space3[64];
char row1[5];
char col1[5];
double xxx;
int num_of_probes;
char row2[5];
char col2[5];
char free_space4[202];
char comment[32];
char pr_name[14];
char pr_sn[16];
short test_plan_leg;
short test_plan_number;
short leg;
char free_space5[554];
int data_type;
int entry_num;
char free_space6[16];
char brd_dr[32];
int chans_on_freq[4];
char tslot[8][32];
int num_of_chans;
char config_name[17];
char notuse1[3];
int bytes_per_slice;
char drive_ampl[8];
int mode;
char gain[8][8][4];
char null[8][8][4];
char select[8][8][4];
short te_chan[2];
short threshold[2];
short dp_gain;
short db_gain;
int frequency;
char type[10];
char notuse1b[2];
double xscale;
double yscale;
int voltscale;
int rotation;
int nullx;
int nully;
double cos;
double sin;
int group;
int coil;
int *more;
char notuse2b[4];
int left_chan;

```



```

int left_dim;
int right_chan;
int right_dim;
int liz_chan;
char up_down[2];
char notuse2[2];
int chart_chan[32];
int chart_dim[32];
int liss_chan[8];
char lastspace[592];
int tmp;
int datatype;
float dt;
float dat0r,dat0i;
float datm1r,datm1i;

XtDestroyWidget(w);
file_open_dialog_exists = 0;

if(!XmStringGetLtoR(cbs->value,XmSTRING_DEFAULT_CHARSET,&filename)) {
    return;
}

if(!*filename) {
    printf("No file selected.\n");
    XtFree(filename);
    return;
}

printf("filename: %s\n",filename);

if(file_open) {

/* Free memory occupied by data from previous file */

    for(j=0;j<=lasttype;j++) {
        for(i=0;i<16;i++) {
            if(DataType[i][j].active) {
                free(DataType[i][j].hdata);
                free(DataType[i][j].vdata);
                DataType[i][j].active = 0;
            }
        }
    }

    for(j=0;j<4;j++) {
        mix_calculated[j] = 0;
    }

}

if((strin = fopen(filename,"rb")) == NULL) {
    printf("Error: failure to open input data file.\n");
    return;
}

fread(&file_type,sizeof(int),1,strin);
fread(&time_date,sizeof(long),1,strin);
fread(&free_space0,sizeof(char),4,strin);
fread(&id,sizeof(char),4,strin);
fread(&row,sizeof(char),4,strin);
fread(&col,sizeof(char),4,strin);

printf("row = %s\n",row);
printf("col = %s\n",col);
printf("id = %s\n",id);

fread(&id_name,sizeof(char),4,strin);
fread(&row_name,sizeof(char),4,strin);

```

```

fread(col_name,sizeof(char),4,stdin);

printf("row_name = %s\n",row_name);
printf("col_name = %s\n",col_name);
printf("id_name = %s\n",id_name);

fread(free_space1,sizeof(char),20,stdin);
fread(&delay,sizeof(int),1,stdin);
printf("delay = %d\n",delay);
fread(free_space2,sizeof(char),16,stdin);

fread(&num_chans,sizeof(int),1,stdin);
printf("num_chans = %d\n",num_chans);

fread(&bytesinslice,sizeof(int),1,stdin);
fread(free_space3,sizeof(char),64,stdin);

fread(row1,sizeof(char),4,stdin);
fread(col1,sizeof(char),4,stdin);

fread(&xxx,sizeof(double),1,stdin);

fread(&num_of_probes,sizeof(int),1,stdin);

fread(row2,sizeof(char),4,stdin);
fread(col2,sizeof(char),4,stdin);
fread(free_space4,sizeof(char),202,stdin);
fread(comment,sizeof(char),32,stdin);
fread(pr_name,sizeof(char),14,stdin);
fread(pr_sn,sizeof(char),16,stdin);

fread(&test_plan_leg,sizeof(short),1,stdin);
fread(&test_plan_number,sizeof(short),1,stdin);
fread(&leg,sizeof(short),1,stdin);
fread(free_space5,sizeof(char),554,stdin);

/* Position 1000 */

fread(&data_type,sizeof(int),1,stdin);

fread(&entry_num,sizeof(int),1,stdin);

fread(free_space6,sizeof(char),16,stdin);

fread(brd_dr,sizeof(char),32,stdin);

fread(chans_on_freq,sizeof(int),4,stdin);

for(i=0;i<32;i++) {
    fread(&tslot[0][i],sizeof(char),8,stdin);
}

fread(&num_of_chans,sizeof(int),1,stdin);

fread(config_name,sizeof(char),17,stdin);
fread(notuse1,sizeof(char),3,stdin);

fread(&bytes_per_slice,sizeof(int),1,stdin);

fread(freq,sizeof(int),4,stdin);

for(i=0;i<4;i++) {
    printf("freq[%d] = %d\n",i,freq[i]);
}

/* Put the frequencies in descending order */

for(i=0;i<4;i++) {

```

```

        for(j=i+1;j<4;j++) {
            if(freq[i] < freq[j]) {
                tmp = freq[i];
                freq[i] = freq[j];
                freq[j] = tmp;
            }
        }
    }

    for(i=0;i<4;i++) {
        printf("freq[%d] = %d\n",i,freq[i]);
    }

    fread(drive_ampl,sizeof(char),8,strin);
    fread(&mode,sizeof(int),1,strin);
    fread(&slices_per_sec,sizeof(int),1,strin);
    fread(gain,sizeof(char),256,strin);
    fread(null,sizeof(char),256,strin);
    fread(select,sizeof(char),256,strin);
    fread(te_chan,sizeof(short),2,strin);
    fread(threshold,sizeof(short),2,strin);
    fread(&dp_gain,sizeof(short),1,strin);
    fread(&db_gain,sizeof(short),1,strin);

/* TEMPORARY */

/*
    freq[0] = 520;
    freq[1] = 260;
    freq[2] = 130;
    freq[3] = 65;
*/

/* End Temporary */

    for(i=0;i<64;i++) {
        fread(&frequency,sizeof(int),1,strin);
        printf("chan=%d,frequency=%d\n",i,frequency);
        fread(type,sizeof(char),10,strin);
        fread(notuse1b,sizeof(char),2,strin);
        fread(&xscale,sizeof(double),1,strin);
        fread(&yyscale,sizeof(double),1,strin);
        fread(&voltscale,sizeof(int),1,strin);
        fread(&rotation,sizeof(int),1,strin);
        fread(&nullx,sizeof(int),1,strin);
        fread(&nully,sizeof(int),1,strin);
        fread(&cos,sizeof(double),1,strin);
        fread(&sin,sizeof(double),1,strin);
        fread(&group,sizeof(int),1,strin);
        fread(&coil,sizeof(int),1,strin);
        fread(&more,sizeof(int *),1,strin);
        fread(notuse2b,sizeof(char),4,strin);

        for(j=0;j<4;j++) {
            if(frequency == freq[j]) {
                for(l=0;l<=lasttype;l++) {
                    if((DataType[0][l].un) && (DataType[0][l].freqind == j)) {
                        datatype = l;
                        printf("%d %d\n",j,l);
                    }
                }
            }
        }
    }

    #if(0)
        printf("%d\n",datatype);
    #endif
    channel[i].coil = coil;
    channel[i].type = datatype;

```

```

    DataType[coil-1][datatype].hdata = allocate_memory(nalloc);
    DataType[coil-1][datatype].vdata = allocate_memory(nalloc);
    DataType[coil-1][datatype].span = 2000.;
    DataType[coil-1][datatype].frequency = frequency;
    DataType[coil-1][datatype].mix = 0;
    DataType[coil-1][datatype].property = 0;
    DataType[coil-1][datatype].active = 1;
}

fread(&left_chan,sizeof(int),1,strin);
fread(&left_dim,sizeof(int),1,strin);
fread(&right_chan,sizeof(int),1,strin);
fread(&right_dim,sizeof(int),1,strin);
fread(&liz_chan,sizeof(int),1,strin);
fread(up_down,sizeof(char),2,strin);
fread(notuse2,sizeof(char),2,strin);
fread(chart_chan,sizeof(int),32,strin);
fread(chart_dim,sizeof(int),32,strin);
fread(liss_chan,sizeof(int),8,strin);
fread(lastspace,sizeof(char),592,strin);

/* Read the data */

k = 0;
do {
    for(i=0;i<64;i++) {
        fread(&datr,sizeof(short),1,strin);
        eoftst = fread(&dati,sizeof(short),1,strin);
        DataType[channel[i].coil-1][channel[i].type].hdata[k] = datr;
        DataType[channel[i].coil-1][channel[i].type].vdata[k] = dati;
    }
    k++;
    if(k>nalloc-1) {
        nalloc += 10000;
        for(coil=1;coil<=16;coil++) {
            for(datatype=0;datatype<4;datatype++) {
                DataType[coil-1][datatype].hdata =
                    reallocate_memory(nalloc,DataType[coil-1][datatype].hdata);
                DataType[coil-1][datatype].vdata =
                    reallocate_memory(nalloc,DataType[coil-1][datatype].vdata);
            }
        }
    }
} while(eoftst == sizeof(char));

npts = k-1;

printf("npts = %d\n",npts);

for(coil=1;coil<=16;coil++) {
    for(datatype=0;datatype<4;datatype++) {
        DataType[coil-1][datatype].hdata =
            reallocate_memory((size_t)npts,DataType[coil-1][datatype].hdata);
        DataType[coil-1][datatype].vdata =
            reallocate_memory((size_t)npts,DataType[coil-1][datatype].vdata);
    }
}

fclose(strin);

file_open = 1;

/* Calculate the amount by which to shift the data in one group of coils */

nlag = slices_per_sec/probe_speed
    * distance_between_coil_groups;

```

```

    if(nulled) {
        do_null(0);
    }

    if(voltage_set) {
        set_volts();
    }

    make_info_label();
    initialize_chart_display();

    label_liss_buttons();

    return;
}

float *allocate_memory(size_t nalloc)
{
    float *ptr;

    ptr = (float *)calloc(nalloc,sizeof(float));

    if(ptr == NULL) {
        printf("Error allocating memory\n");
        exit(-1);
    }

    return ptr;
}

float *reallocate_memory(size_t nalloc, float *ptr1)
{
    float *ptr2;

    ptr2 = (float *)realloc(ptr1,nalloc*sizeof(float));

    if(ptr2 == NULL) {
        printf("Error allocating memory\n");
        exit(-1);
    }

    return ptr2;
}

void quitCB(Widget w, XtPointer client_data, XtPointer call_data)
{
    exit(0);
}

void FilePrint_screenCB(Widget w, XtPointer client_data, XtPointer call_data)
{
    char command[80];

    sprintf(command,"xwd -id %d| xpr -device ljet | lp -oraw",XtWindow(main_w));
    system(command);

    return;
}

void setup_chart_display(void)
{
    int i;

    for(i=0;i<16;i++) {
        chart[i].coil = (i/2+1) + 8*(i%2);
        chart[i].type = 1;
        chart[i].vertical_flag = 1;
    }
}

```

```

    label_chart_buttons();

    return;
}

void setup_liss_display(void)
{
    int i;

    for(i=0;i<8;i++) {
        lissajous[i].coil = i+1;
        lissajous[i].type = 1;
    }

    label_liss_buttons();

    return;
}

void initialize_chart_display(void)
{
    int i;

    cpt1 = 0;
    cpt2 = npts;

    XClearWindow(XtDisplay(chart_canvas),XtWindow(chart_canvas));

/* Get dimensions of window */

    XtVaGetValues(chart_canvas,
                  XmNwidth,&chart_area_width,
                  XmNheight,&chart_area_height,
                  NULL);

    for(i=0;i<ncharts;i++) {
        chart[i].llim = i * chart_area_width/ncharts + 1;
        chart[i].ulim = (i+1) * chart_area_width/ncharts - 1;
        chart[i].center = 0.5 * (chart[i].ulim + chart[i].llim);
    }
    label_chart_buttons();

    XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
    draw_chart_data();

    return;
}

void initialize_liss_display(void)
{
    int i;

    lpt1 = cpt1;
    lpt2 = cpt2;

    for(i=0;i<8;i++) {
        XClearWindow(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]));
    }

/* Get dimensions of window */

    XtVaGetValues(liss_canvas[0],
                  XmNwidth,&liss_area_width,
                  XmNheight,&liss_area_height,
                  NULL);

    for(i=0;i<8;i++) {
        lissajous[i].xllim = 1;

```

```

    lissajous[i].xulim = liss_area_width - 1;
    lissajous[i].yllim = 1;
    lissajous[i].yulim = liss_area_height - 1;
    lissajous[i].xcenter = 0.5 * (lissajous[i].xulim + lissajous[i].xllim);
    lissajous[i].ycenter = 0.5 * (lissajous[i].yulim + lissajous[i].yllim);
}

label_liss_buttons();

if(file_open) {
    XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
    draw_liss_data();
    update_liss_labels(-1);
}

return;
}

void draw_liss_data(void)
{
    float x,y;
    float xmin,xmax;
    float ymin,ymax;
    float *hdat,*vdat;
    int dim;
    int ix,iy;
    int i,j;
    int n;
    int lag;
    float angle;
    float sa,ca;

    if(liss_area_width < liss_area_height) {
        dim = liss_area_width;
    }
    else {
        dim = liss_area_height;
    }

    if(nulled) {
        for(i=0;i<8;i++) {
            lissajous[i].xoffset = 0.;
            lissajous[i].yoffset = 0.;
        }
    }
    else {
        for(i=0;i<8;i++) {
            if(!DataType[lissajous[i].coil-1][lissajous[i].type].active) {
                continue;
            }
            if(lissajous[i].coil < 9) {
                lag = nlag;
            }
            else {
                lag = 0;
            }
            hdat = DataType[lissajous[i].coil-1][lissajous[i].type].hdata;
            vdat = DataType[lissajous[i].coil-1][lissajous[i].type].vdata;
            xmax = xmin = *(hdat + lpt1);
            ymax = ymin = *(vdat + lpt1);
            for(j=lpt1;j<lpt2;j++) {
                x = *(hdat + (j+lag)%npts);
                y = *(vdat + (j+lag)%npts);
                if(x > xmax) {
                    xmax = x;
                }
                if(x < xmin) {
                    xmin = x;
                }
            }
        }
    }
}

```

```

    }
    if(y > ymax) {
        ymax = y;
    }
    if(y < ymin) {
        ymin = y;
    }
}
lissajous[i].xoffset = -0.5 * (xmax + xmin);
lissajous[i].yoffset = -0.5 * (ymax + ymin);
}
}

/* Draw data */

for(i=0;i<8;i++) {
    if(!DataType[lissajous[i].coil-1][lissajous[i].type].active) {
        continue;
    }
    if(lissajous[i].coil < 9) {
        lag = nlag;
    }
    else {
        lag = 0;
    }
    angle = DataType[lissajous[i].coil-1][lissajous[i].type].rotation;
    sa = sin(angle);
    ca = cos(angle);
    hdat = DataType[lissajous[i].coil-1][lissajous[i].type].hdata;
    vdat = DataType[lissajous[i].coil-1][lissajous[i].type].vdata;
    n = 0;
    for(j=lpt1;j<lpt2;j++) {
        x = (*(hdat + (j+lag)%npts) + lissajous[i].xoffset) * dim
            /DataType[lissajous[i].coil-1][lissajous[i].type].span;
        y = (*(vdat + (j+lag)%npts) + lissajous[i].yoffset) * dim
            /DataType[lissajous[i].coil-1][lissajous[i].type].span;

        ix = x*ca - y*sa;
        iy = y*ca + x*sa;

        point[n].x = ix + lissajous[i].xcenter;
        point[n].y = -iy + lissajous[i].ycenter;
        n++;
        if(n > 2000) {
            XDrawLines(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]),
                gc,point,n,CoordModeOrigin);
            point[0].x = point[n-1].x;
            point[0].y = point[n-1].y;
            n = 1;
        }
    }
    if(n > 1) {
        XDrawLines(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]),
            gc,point,n,CoordModeOrigin);
    }
}

return;
}

void draw_one_liss_data(int i)
{
    float x,y;
    float xmin,xmax;
    float ymin,ymax;
    float *hdat,*vdat;
    float angle;
    float sa,ca;

```



```

int dim;
int ix, iy;
int j;
int n;
int lag;

if(liss_area_width < liss_area_height) {
    dim = liss_area_width;
}
else {
    dim = liss_area_height;
}

if(lissajous[i].coil < 9) {
    lag = nlag;
}
else {
    lag = 0;
}

if(nulled) {
    lissajous[i].xoffset = 0.;
    lissajous[i].yoffset = 0.;
}
else {
    hdat = DataType[lissajous[i].coil-1][lissajous[i].type].hdata;
    vdat = DataType[lissajous[i].coil-1][lissajous[i].type].vdata;
    xmax = xmin = *(hdat + lpt1);
    ymax = ymin = *(vdat + lpt1);
    for(j=lpt1; j<lpt2; j++) {
        x = *(hdat + (j+lag)%npts);
        y = *(vdat + (j+lag)%npts);
        if(x > xmax) {
            xmax = x;
        }
        if(x < xmin) {
            xmin = x;
        }
        if(y > ymax) {
            ymax = y;
        }
        if(y < ymin) {
            ymin = y;
        }
    }
    lissajous[i].xoffset = -0.5 * (xmax + xmin);
    lissajous[i].yoffset = -0.5 * (ymax + ymin);
}

/* Draw data */

angle = DataType[lissajous[i].coil-1][lissajous[i].type].rotation;
sa = sin(angle);
ca = cos(angle);
hdat = DataType[lissajous[i].coil-1][lissajous[i].type].hdata;
vdat = DataType[lissajous[i].coil-1][lissajous[i].type].vdata;
n = 0;
for(j=lpt1; j<lpt2; j++) {
    x = (*(hdat + (j+lag)%npts) + lissajous[i].xoffset) * dim
        /DataType[lissajous[i].coil-1][lissajous[i].type].span;
    y = (*(vdat + (j+lag)%npts) + lissajous[i].yoffset) * dim
        /DataType[lissajous[i].coil-1][lissajous[i].type].span;

    ix = x*ca - y*sa;
    iy = y*ca + x*sa;

    point[n].x = ix + lissajous[i].xcenter;
    point[n].y = -iy + lissajous[i].ycenter;
}

```

```

    n++;
    if(n > 2000) {
        XDrawLines(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]),
            gc,point,n,CoordModeOrigin);
        point[0].x = point[n-1].x;
        point[0].y = point[n-1].y;
        n = 1;
    }
}
if(n > 1) {
    XDrawLines(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]),
        gc,point,n,CoordModeOrigin);
}
return;
}

void draw_chart_dividers(void)
{
    int i;

    XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);

    for(i=0;i<ncharts;i++) {
        XDrawLine(XtDisplay(chart_canvas),XtWindow(chart_canvas),gc,
            (int)(i*chart_area_width/16),0,
            (int)(i*chart_area_width/16),chart_area_height);
    }

    return;
}

void RefreshCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int i;

    if(!file_open) {
        return;
    }

    XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);

    XClearWindow(XtDisplay(chart_canvas),XtWindow(chart_canvas));
    draw_chart_data();

    if(mode == 2) {
        for(i=0;i<8;i++) {
            XClearWindow(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]));
            draw_one_liss_data(i);
        }
    }

    if(mode == 3) {
        draw_density_plot();
    }

    return;
}

void chart_exposeCB(Widget w, XtPointer client_data,
    XmDrawingAreaCallbackStruct *cbs)
{
    printf("Chart area exposed\n");

    if(cursor_y != -1) {
        XClearArea(XtDisplay(chart_canvas),XtWindow(chart_canvas),0,cursor_y-5,
            chart_area_width,10,False);
    }
}

```

```

    cursor_y = -1;

    if(file_open) {
        XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
        draw_chart_data();
    }

    return;
}

void liss_exposeCB(Widget w, XtPointer client_data,
    XmDrawingAreaCallbackStruct *cbs)
{
    int i;

    if(!file_open) {
        return;
    }

    for(i=0;i<8;i++) {
        if(w == liss_canvas[i]) {
            printf("Lissajous %d exposed\n",i);
            XSetForeground(XtDisplay(liss_canvas[i]),gc,color.foreground);
            draw_one_liss_data(i);
            return;
        }
    }

    return;
}

void draw_logo(void)
{
    XPoint logo_xpoint[82];
    float logo_point[82][2] = {
        {0.,13.},{0.,16.},{1.,18.},{2.,19.},{3.,20.},{6.,21.},
        {9.,21.},{11.,20.},{13,18},{13.5,17},{14,14},{16,14},
        {16,19},
        {15,20},{15,20.5},{15.5,21},{22.5,21},{23,20.5},{23,20},
        {22,19},{22,11},{23,11},{24,12},{24.5,13},{25.5,13},
        {26,12},{27,12.5},{27,19},{26,20},{26,20.5},{26.5,21},
        {33.5,21},{34,20.5},{34,20},{33,19},{33,13},{35,12},
        {35.5,12},{36,13},{37,15},{38,17},{38,19},{37,20},
        {37,20.5},{37.5,21},{53.5,21},{54,20.5},{54,20},{53,19},
        {53,0.5},{52.5,0},{50,0},{45,3},{45,4},{47,5},
        {47,18},{46,19},{45,19},{44,18},{42,10},{41.5,9},
        {40,8},{34,8},{33,9},{32.5,8},{31,8},{28,9},
        {27,9},{25,8},{24,8},{22,9},{21,8.5},{20,8.5},
        {14,11},{12,9},{10,8},{8,7.5},{7,7.5},{5,8},
        {3,9},{1,11},{0,13}
    };
    float logo_point2[11][2] = {
        {9,19},{10,17},{10,13},{9,11},{8,10},{7,9.5},
        {6,10},{5,13},{5,17},{7,19},{9,19}
    };
    int i;
    int x1,y1,x2,y2;

    /* Clear pixmap */

    XSetForeground(XtDisplay(main_w),gc,color.background);
    XFillRectangle(XtDisplay(main_w),logopixmap,gc,0,0,logowidth,logoheight);

    /* Draw logo */

    XSetForeground(XtDisplay(chart_canvas),gc,color.light);

    logo_xpoint[0].x = logo_point[0][0] * 0.95*logoheight/21. + 0.5;

```

```

logo_xpoint[0].y = logo_point[0][1] * 0.95*logoheight/21. + 0.5;
for(i=1;i<82;i++) {
    x1 = logo_point[i-1][0] * 0.95*logoheight/21. + 0.5;
    y1 = logo_point[i-1][1] * 0.95*logoheight/21. + 0.5;
    x2 = logo_point[i][0] * 0.95*logoheight/21. + 0.5;
    y2 = logo_point[i][1] * 0.95*logoheight/21. + 0.5;
    logo_xpoint[i].x = x2;
    logo_xpoint[i].y = y2;
    XDrawLine(XtDisplay(chart_canvas),logopixmap,gc,
        x1,y1,x2,y2);
}

XSetForeground(XtDisplay(chart_canvas),gc,color.light);
XFillPolygon(XtDisplay(chart_canvas),logopixmap,gc,logo_xpoint,
    82,Nonconvex,CoordModeOrigin);

logo_xpoint[0].x = logo_point2[0][0] * 0.95*logoheight/21. + 0.5;
logo_xpoint[0].y = logo_point2[0][1] * 0.95*logoheight/21. + 0.5;
for(i=1;i<11;i++) {
    x1 = logo_point2[i-1][0] * 0.95*logoheight/21. + 0.5;
    y1 = logo_point2[i-1][1] * 0.95*logoheight/21. + 0.5;
    x2 = logo_point2[i][0] * 0.95*logoheight/21. + 0.5;
    y2 = logo_point2[i][1] * 0.95*logoheight/21. + 0.5;
    logo_xpoint[i].x = x2;
    logo_xpoint[i].y = y2;
    XDrawLine(XtDisplay(chart_canvas),logopixmap,gc,
        x1,y1,x2,y2);
}

XSetForeground(XtDisplay(chart_canvas),gc,color.background);
XFillPolygon(XtDisplay(chart_canvas),logopixmap,gc,logo_xpoint,
    11,Nonconvex,CoordModeOrigin);

XSetForeground(XtDisplay(chart_canvas),gc,color.light);
for(i=1;i<11;i++) {
    x1 = logo_point2[i-1][0] * 0.95*logoheight/21. + 0.5;
    y1 = logo_point2[i-1][1] * 0.95*logoheight/21. + 0.5;
    x2 = logo_point2[i][0] * 0.95*logoheight/21. + 0.5;
    y2 = logo_point2[i][1] * 0.95*logoheight/21. + 0.5;
    logo_xpoint[i].x = x2;
    logo_xpoint[i].y = y2;
    XDrawLine(XtDisplay(chart_canvas),logopixmap,gc,
        x1,y1,x2,y2);
}

#if(0)
    draw_chart_dividers();
#endif
return;
}

void chart_resizeCB(Widget w, XtPointer client_data,
    XmDrawingAreaCallbackStruct *cbs)
{
    int i;

    cursor_y = -1;

    XtVaGetValues(chart_canvas,
        XmNwidth,&chart_area_width,
        XmNheight,&chart_area_height,
        NULL);

    /* Recalculate sy1 and sy2 */
    /* ... */

    for(i=0;i<ncharts;i++) {
        chart[i].l1im = i * chart_area_width/ncharts + 1;
    }
}

```

```

    chart[i].ulim = (i+1) * chart_area_width/ncharts - 1;
    chart[i].center = 0.5 * (chart[i].ulim + chart[i].llim);
}

return;
}

void liss_resizeCB(Widget w,XtPointer client_data,
                  XmDrawingAreaCallbackStruct *cbs)
{
    int i;

    XtVaGetValues(liss_canvas[0],
                  XmNwidth,&liss_area_width,
                  XmNheight,&liss_area_height,
                  NULL);

    for(i=0;i<8;i++) {
        lissajous[i].xllim = 1;
        lissajous[i].xulim = liss_area_width - 1;
        lissajous[i].yllim = 1;
        lissajous[i].yulim = liss_area_height - 1;
        lissajous[i].xcenter = 0.5 * (lissajous[i].xulim + lissajous[i].xllim);
        lissajous[i].ycenter = 0.5 * (lissajous[i].yulim + lissajous[i].yllim);
    }

    return;
}

void make_info_label(void)
{
    XmString cstring;
    char string[80];

    sprintf(string,"SG: %s      Row: %s      Col: %s      ",id,row,col);

    cstring = XmStringCreateSimple(string);

    XtVaSetValues(infolabel,
                  XmNlabelString,cstring,
                  NULL);

    XmStringFree(cstring);

    return;
}

void label_chart_buttons(void)
{
    XmString cstring;
    char string[80];
    int i;

    /* The following loop should be performed for all 16 buttons
       regardless of how many are managed */

    for(i=0;i<16;i++) {
        sprintf(string,"Coil %d",chart[i].coil);
        cstring = XmStringCreateSimple(string);
        XtVaSetValues(coil_button[i],
                      XmNlabelString,cstring,
                      NULL);
        XmStringFree(cstring);

        if(DataType[chart[i].coil-1][chart[i].type].property == 0) {
            if(file_open) {
                if(DataType[chart[i].coil-1][chart[i].type].un) {
                    sprintf(string,"%d ",

```

```

        DataType[chart[i].coil-1][chart[i].type].frequency);
    }
    else if(DataType[chart[i].coil-1][chart[i].type].mix) {
        sprintf(string,"Mix %d",
            DataType[chart[i].coil-1][chart[i].type].mixind);
    }
}
else {
    sprintf(string,"Frq%d ",chart[i].type + 1);
}
if(chart[i].vertical_flag) {
    strcat(string,"V");
}
else {
    strcat(string,"H");
}
}
else if(DataType[chart[i].coil-1][chart[i].type].property == 1) {
    strcpy(string,"DfDep");
}
else {
    printf("Error in label_chart_buttons().\n");
}
cstring = XmStringCreateSimple(string);
XtVaSetValues(data_button[i],
    XmNlabelString,cstring,
    NULL);
XmStringFree(cstring);
}

return;
}

void label_liss_buttons(void)
{
    XmString cstring;
    char string[80];
    int i;

    for(i=0;i<8;i++) {
        sprintf(string,"Coil %d",lissajous[i].coil);
        cstring = XmStringCreateSimple(string);
        XtVaSetValues(liss_coil_button[i],
            XmNlabelString,cstring,
            NULL);
        XmStringFree(cstring);

        if(file_open) {
            if(DataType[lissajous[i].coil-1][lissajous[i].type].un) {
                sprintf(string,"%d kHz",
                    DataType[lissajous[i].coil-1][lissajous[i].type].frequency);
            }
            else if(DataType[lissajous[i].coil-1][lissajous[i].type].mix) {
                sprintf(string,"Mix %d",
                    DataType[lissajous[i].coil-1][lissajous[i].type].mixind);
            }
            else {
                printf("Error in label_liss_buttons.\n");
            }
        }
        else {
            sprintf(string,"Freq %d",lissajous[i].type + 1);
        }

        cstring = XmStringCreateSimple(string);
        XtVaSetValues(liss_data_button[i],
            XmNlabelString,cstring,
            NULL);
    }
}

```

```

    XmStringFree(cstring);
}

return;
}

void draw_chart_data(void)
{
/* Remove trig functions from the loops over j */
float dat;
float datmax, datmin;
float *hdat, *vdat;
float angle;
float sa, ca;
float chart_width;
int i, j;
int n = 0;
int ix, iy;
int lag;

/* Find data offset */

if(nulled) {
    for(i=0; i<ncharts; i++) {
        chart[i].data_offset = 0.;
    }
}
else {
    for(i=0; i<ncharts; i++) {
        if(!DataType[chart[i].coil-1][chart[i].type].active) {
            continue;
        }
        angle = DataType[chart[i].coil-1][chart[i].type].rotation;
        if(chart[i].vertical_flag) {
            angle -= 90. * 3.1415927/180.;
        }
        sa = sin(angle);
        ca = cos(angle);

        if(chart[i].coil < 9) {
            lag = nlag;
        }
        else {
            lag = 0;
        }
        hdat = DataType[chart[i].coil-1][chart[i].type].hdata;
        vdat = DataType[chart[i].coil-1][chart[i].type].vdata;
        datmax = datmin = *(hdat + (cpt1 + lag)%npts) * ca
            - *(vdat + (cpt1 + lag)%npts) * sa;
        for(j=cpt1; j<cpt2; j++) {
            dat = *(hdat + (j + lag)%npts) * ca
                - *(vdat + (j + lag)%npts) * sa;
            if(dat > datmax) {
                datmax = dat;
            }
            else if(dat < datmin) {
                datmin = dat;
            }
        }
        chart[i].data_offset = -0.5*(datmax + datmin);
    }
}

/* Draw data */

for(i=0; i<ncharts; i++) {
    if(!DataType[chart[i].coil-1][chart[i].type].active) {
        continue;
    }

```

```

    }
    angle = DataType[chart[i].coil-1][chart[i].type].rotation;
    if(chart[i].vertical_flag) {
        angle -= 90. * 3.1415927/180.;
    }
    sa = sin(angle);
    ca = cos(angle);

    if(chart[i].coil < 9) {
        lag = nlag;
    }
    else {
        lag = 0;
    }
    chart_width = chart[i].ulim - chart[i].llim;
    hdat = DataType[chart[i].coil-1][chart[i].type].hdata;
    vdat = DataType[chart[i].coil-1][chart[i].type].vdata;
    for(j=cpt1; j<cpt2; j++) {
        ix = (
            *(hdat + (j+lag)%npts) * ca
            - *(vdat + (j+lag)%npts) * sa
            + chart[i].data_offset) * chart_width
            /DataType[chart[i].coil-1][chart[i].type].span
            + chart[i].center;
        if(ix > chart[i].ulim) {
            ix = chart[i].ulim;
        }
        else if(ix < chart[i].llim) {
            ix = chart[i].llim;
        }
        iy = (chart_area_height * (j - cpt1))/(cpt2 - cpt1);
        point[n].x = ix;
        point[n].y = iy;
        n++;
        if(n > 2000) {
            XDrawLines(XtDisplay(chart_canvas), XtWindow(chart_canvas),
                gc, point, n, CoordModeOrigin);
            point[0].x = point[n-1].x;
            point[0].y = point[n-1].y;
            n = 1;
        }
    }
    if(n > 1) {
        XDrawLines(XtDisplay(chart_canvas), XtWindow(chart_canvas),
            gc, point, n, CoordModeOrigin);
    }
    n = 0;
}

return;
}

void draw_one_chart_data(int ichart)
{
    /* Remove trig functions from the loops over j */
    float dat;
    float datmax, datmin;
    float *hdat, *vdat;
    float angle;
    float sa, ca;
    float chart_width;
    int i, j;
    int n = 0;
    int ix, iy;
    int lag;

    i = ichart;

```



```

if(!DataType[chart[i].coil-1][chart[i].type].active) {
    return;
}

if(chart[i].coil < 9) {
    lag = nlag;
}
else {
    lag = 0;
}

/* Find data offset */

angle = DataType[chart[i].coil-1][chart[i].type].rotation;
if(chart[i].vertical_flag) {
    angle -= 90. * 3.1415927/180.;
}
sa = sin(angle);
ca = cos(angle);

hdat = DataType[chart[i].coil-1][chart[i].type].hdata;
vdat = DataType[chart[i].coil-1][chart[i].type].vdata;
datmax = datmin = *(hdat + (cpt1+lag)%npts) * ca
    - *(vdat + (cpt1+lag)%npts) * sa;
for(j=cpt1; j<cpt2; j++) {
    dat = *(hdat + (j+lag)%npts) * ca
        - *(vdat + (j+lag)%npts) * sa;
    if(dat > datmax) {
        datmax = dat;
    }
    else if(dat < datmin) {
        datmin = dat;
    }
}
if(null) {
    chart[i].data_offset = 0.;
}
else {
    chart[i].data_offset = -0.5*(datmax + datmin);
}

/* Draw data */

chart_width = chart[i].ulim - chart[i].llim;

for(j=cpt1; j<cpt2; j++) {
    ix = (
        *(hdat + (j+lag)%npts) * ca
        - *(vdat + (j+lag)%npts) * sa
        + chart[i].data_offset) * chart_width
        /DataType[chart[i].coil-1][chart[i].type].span
        + chart[i].center;
    if(ix > chart[i].ulim) {
        ix = chart[i].ulim;
    }
    else if(ix < chart[i].llim) {
        ix = chart[i].llim;
    }
    iy = (chart_area_height * (j - cpt1))/(cpt2 - cpt1);
    point[n].x = ix;
    point[n].y = iy;
    n++;
    if(n > 2000) {
        XDrawLines(XtDisplay(chart_canvas), XtWindow(chart_canvas),
            gc, point, n, CoordModeOrigin);
        point[0].x = point[n-1].x;
        point[0].y = point[n-1].y;
        n = 1;
    }
}

```

```

    }
}
if(n > 1) {
    XDrawLines(XtDisplay(chart_canvas),XtWindow(chart_canvas),
        gc,point,n,CoordModeOrigin);
}
n = 0;

return;
}

void ToolsProbe_speedCB(Widget w, XtPointer client_data,
    XtPointer call_data)
{
    Widget dialog, pane, rc, scale, action_a;
    XmString cstring;
    static ActionAreaItem action_items[] = {
        {"OK",ToolsProbe_speedOKCB,NULL},
        {"Cancel",ToolsProbe_speedCancelCB,NULL},
        {"Apply",ToolsProbe_speedApplyCB,NULL},
        {"Help",NULL,NULL}
    };
    ProbeSpeedDataType *ProbeSpeedData;
    int n;

    ProbeSpeedData = client_data;

    ProbeSpeedData->new_probe_speed = probe_speed;
    ProbeSpeedData->old_probe_speed = probe_speed;

    n = 0;
    XtSetArg(args[n],XmNdeleteResponse,XmDESTROY);n++;
    XtSetArg(args[n],XmNx,300);n++;
    XtSetArg(args[n],XmNy,200);n++;
    dialog = XmCreateDialogShell(toplevel,"dialog",args,n);

    ProbeSpeedData->shell = dialog;

    action_items[0].data = ProbeSpeedData;
    action_items[1].data = ProbeSpeedData;
    action_items[2].data = ProbeSpeedData;

    pane = XtVaCreateWidget("pane",
        xmPanedWindowWidgetClass,dialog,
        XmNsashWidth,1,
        XmNsashHeight,1,
        NULL);

    rc = XtVaCreateWidget("control_area",
        xmRowColumnWidgetClass,pane,
        NULL);

    cstring = XmStringCreateSimple("Probe Speed");
    scale = XtVaCreateManagedWidget("probespeed",
        xmScaleWidgetClass,rc,
        XmNtitleString,cstring,
        XmNorientation,XmHORIZONTAL,
        XmNmaximum,4000,
        XmNminimum,1,
        XmNvalue,(int)(100*probe_speed),
        XmNshowValue,True,
        XmNdecimalPoints,2,
        NULL);
    XmStringFree(cstring);
    XtAddCallback(scale,XmNvalueChangedCallback,
        (XtCallbackProc)ProbeSpeedChanged,ProbeSpeedData);

    XtManageChild(rc);
}

```

```

    action_a = create_action_area(pane,action_items,XtNumber(action_items));

    XtManageChild(pane);
    XtManageChild(dialog);

    return;
}

Widget create_action_area(Widget parent, ActionAreaItem *actions,
    int num_actions)
{
    Widget action_area, widget;
    int i;

    action_area = XtVaCreateWidget("action_area",
        xmFormWidgetClass,parent,
        XmNfractionBase,20*num_actions - 1,
        NULL);

    for(i=0;i<num_actions;i++) {
        widget = XtVaCreateManagedWidget(actions[i].label,
            xmPushButtonWidgetClass,action_area,
            XmNleftAttachment,i?XmATTACH_POSITION:XmATTACH_FORM,
            XmNleftPosition,20*i,
            XmNtopAttachment,XmATTACH_FORM,
            XmNbottomAttachment,XmATTACH_FORM,
            XmNrightAttachment,
                i != num_actions-1?XmATTACH_POSITION:XmATTACH_FORM,
            XmNrightPosition,20*i + 19,
            XmNshowAsDefault,i==0,
            XmNdefaultButtonShadowThickness,1,
            NULL);

        if(actions[i].callback) {
            XtAddCallback(widget,XmNactivateCallback,
                actions[i].callback,actions[i].data);
        }

        if(i == 0) {
            Dimension height,h;
            XtVaGetValues(action_area,XmNmarginHeight,&h,NULL);
            XtVaGetValues(widget,XmNheight,&height,NULL);
            height += 2*h;
            XtVaSetValues(action_area,
                XmNdefaultButton,widget,
                XmNpaneMaximum,height,
                XmNpaneMinimum,height,
                NULL);
        }
    }

    XtManageChild(action_area);

    return action_area;
}

void ToolsProbe_speedOKCB(Widget w, XtPointer client_data, XtPointer call_data)
{
    ProbeSpeedDataType *ProbeSpeedData;
    int i;

    ProbeSpeedData = client_data;

    XtDestroyWidget(ProbeSpeedData->shell);

    if(probe_speed != ProbeSpeedData->new_probe_speed) {
        /* Erase chart data */
    }
}

```

```

        if(file_open) {
            XSetForeground(XtDisplay(chart_canvas),gc,color.background);
            for(i=0;i<ncharts;i++) {
                if(chart[i].coil < 9) {
                    draw_one_chart_data(i);
                }
            }
        }

        probe_speed = ProbeSpeedData->new_probe_speed;
        nlag = slices_per_sec/probe_speed
            * distance_between_coil_groups;

/* Draw chart data */

        if(file_open) {
            XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
            for(i=0;i<ncharts;i++) {
                if(chart[i].coil < 9) {
                    draw_one_chart_data(i);
                }
            }
        }
        return;
    }

void ToolsProbe_speedCancelCB(Widget w, XtPointer client_data,
    XtPointer call_data)
{
    int i;
    ProbeSpeedDataType *ProbeSpeedData;

    ProbeSpeedData = client_data;

    XtDestroyWidget(ProbeSpeedData->shell);

    if(probe_speed != ProbeSpeedData->old_probe_speed) {
/* Erase chart data */
        if(file_open) {
            XSetForeground(XtDisplay(chart_canvas),gc,color.background);
            for(i=0;i<ncharts;i++) {
                if(chart[i].coil < 9) {
                    draw_one_chart_data(i);
                }
            }
        }
        probe_speed = ProbeSpeedData->old_probe_speed;
        nlag = slices_per_sec/probe_speed
            * distance_between_coil_groups;

/* Draw chart data */

        if(file_open) {
            XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
            for(i=0;i<ncharts;i++) {
                if(chart[i].coil < 9) {
                    draw_one_chart_data(i);
                }
            }
        }
        return;
    }

void ToolsProbe_speedApplyCB(Widget w, XtPointer client_data,
    XtPointer call_data)

```

```

{
    int i;
    ProbeSpeedDataType *ProbeSpeedData;

    ProbeSpeedData = client_data;

    if(probe_speed != ProbeSpeedData->new_probe_speed) {
/* Erase chart data */

        if(file_open) {
            XSetForeground(XtDisplay(chart_canvas),gc,color.background);
            for(i=0;i<ncharts;i++) {
                if(chart[i].coil < 9) {
                    draw_one_chart_data(i);
                }
            }
        }
        probe_speed = ProbeSpeedData->new_probe_speed;
        nlag = slices_per_sec/probe_speed
            * distance_between_coil_groups;

/* Draw chart data */

        if(file_open) {
            XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
            for(i=0;i<ncharts;i++) {
                if(chart[i].coil < 9) {
                    draw_one_chart_data(i);
                }
            }
        }
    }
    return;
}

void ProbeSpeedChanged(Widget w, XtPointer client_data,XmScaleCallbackStruct *cbs)
{
    ProbeSpeedDataType *ProbeSpeedData;

    ProbeSpeedData = client_data;

    ProbeSpeedData->new_probe_speed = cbs->value/100.;
    printf("New probe speed = %f\n",ProbeSpeedData->new_probe_speed);
}

Widget BuildPulldownMenu(Widget parent,char *menu_title,
    char menu_mnemonic,MenuItemType *items)
{
    Widget pulldown,cascade,widget;
    int i;
    XmString cstring;

    pulldown = XmCreatePulldownMenu(parent,"pulldown",NULL,0);

    cstring = XmStringCreateSimple(menu_title);
    cascade = XtVaCreateManagedWidget(menu_title,
        xmCascadeButtonGadgetClass,parent,
        XmNsubMenuId,pulldown,
        XmNlabelString,cstring,
        XmNmnemonic,menu_mnemonic,
        NULL);
    XmStringFree(cstring);

/* Add the menu items */

    for(i=0;items[i].label != NULL;i++) {

```

```

        if(items[i].subitems) {
            widget = BuildPulldownMenu(pulldown,
                items[i].label,items[i].mnemonic,
                (MenuItemType *)items[i].subitems);
        }
        else {
            widget = XtVaCreateManagedWidget(items[i].label,
                *items[i].class,pulldown,
                NULL);
        }

        if(items[i].mnemonic) {
            XtVaSetValues(widget,XmNmnemonic,items[i].mnemonic,NULL);
        }

        if(items[i].accelerator) {
            cstring = XmStringCreateSimple(items[i].accel_text);
            XtVaSetValues(widget,
                XmNaccelerator,items[i].accelerator,
                XmNacceleratorText,cstring,
                NULL);
            XmStringFree(cstring);
        }

        if(items[i].callback) {
            XtAddCallback(widget,XmNactivateCallback,
                items[i].callback,items[i].callback_data);
        }
    }
    return cascade;
}

void FileWrite_dataCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int i,j,k;
    int lag;
    int jj;
    int ctype = 1;
    char filename[80];
    float x,y;
    float rx,ry;
    float angle;
    float *hdat,*vdat;
    float sa,ca;
    FILE *str1;

    for(j=0;j<16;j++) {
        if(j < 8) {
            lag = nlag;
        }
        else {
            lag = 0;
        }
        sprintf(filename,"coil%d.dat",j);
        str1 = fopen(filename,"w");
        if(str1 == NULL) {
            printf("Unable to open output data file.\n");
            return;
        }
        for(i=spt1;i<spt2;i++) {
            for(k=0;k<=lastactivetype;k++) {
                if(DataType[j][k].active) {
                    fprintf(str1,"%f ",DataType[j][k].hdata[(i+lag)%npts]);
                    fprintf(str1,"%f ",DataType[j][k].vdata[(i+lag)%npts]);
                }
            }
            fprintf(str1,"\n");
        }
    }
}

```

```

    }
    fclose(str1);
}

str1 = fopen("mnout.dat","w");
if(str1 == NULL) {
    printf("Unable to open output data file.\n");
    return;
}

#if(0)
for(j=0;j<8;j++) {
    jj = j;
    lag = nlag;
    angle = DataType[jj][ctype].rotation;
    sa = sin(angle);
    ca = cos(angle);
    hdat = DataType[jj][ctype].hdata;
    vdat = DataType[jj][ctype].vdata;
    for(i=spt1;i<spt2;i+=3) {
        x = *(hdat + (i+lag)%npts);
        y = *(vdat + (i+lag)%npts);
        rx = x*ca - y*sa;
        ry = y*ca + x*sa;

        fprintf(str1,"%f %f %f\n",i*0.01,j*45.,ry);
    }

    jj = j + 8;
    lag = 0;
    angle = DataType[jj][ctype].rotation;
    sa = sin(angle);
    ca = cos(angle);
    hdat = DataType[jj][ctype].hdata;
    vdat = DataType[jj][ctype].vdata;
    for(i=spt1;i<spt2;i+=3) {
        x = *(hdat + (i+lag)%npts);
        y = *(vdat + (i+lag)%npts);
        rx = x*ca - y*sa;
        ry = y*ca + x*sa;

        fprintf(str1,"%f %f %f\n",i*0.01,j*45.+22.5,ry);
    }
}
#endif

for(i=spt1;i<spt2;i+=3) {
    for(j=0;j<8;j++) {
        jj = j;
        lag = nlag;
        angle = DataType[jj][ctype].rotation;
        sa = sin(angle);
        ca = cos(angle);
        hdat = DataType[jj][ctype].hdata;
        vdat = DataType[jj][ctype].vdata;
        x = *(hdat + (i+lag)%npts);
        y = *(vdat + (i+lag)%npts);
        rx = x*ca - y*sa;
        ry = y*ca + x*sa;

        fprintf(str1,"%f ",ry);

        jj = j + 8;
        lag = 0;
        angle = DataType[jj][ctype].rotation;
        sa = sin(angle);
        ca = cos(angle);
        hdat = DataType[jj][ctype].hdata;

```

```

        vdat = DataType[j][ctype].vdata;
        x = *(hdat + (i+lag)%npts);
        y = *(vdat + (i+lag)%npts);
        rx = x*ca - y*sa;
        ry = y*ca + x*sa;

        fprintf(str1,"%f ",ry);
    }
    fprintf(str1,"\n");
}

fclose(str1);

return;
}

void ToolsNullCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    do_null(1);

    return;
}

void do_null(int command)
{
    int i,j,k;
    float nullvalh[16][4];
    float nullvalv[16][4];
    static float nullvalh0[16][4];
    static float nullvalv0[16][4];
    static first = 1;
    int lag;

    if(first) {
        for(i=0;i<16;i++) {
            for(j=0;j<4;j++) {
                nullvalh0[i][j] = 0.;
                nullvalv0[i][j] = 0.;
            }
        }
        first = 0;
    }

    if(command == 0) {
        for(i=0;i<npts;i++) {
            for(j=0;j<16;j++) {
                for(k=0;k<4;k++) {
                    DataType[j][k].hdata[i] -= nullvalh0[j][k];
                    DataType[j][k].vdata[i] -= nullvalv0[j][k];
                }
            }
        }
        nulled = 1;
    }
    else if(command == 1) {
        if(spt1 == -1) {
            printf("No region selected.\n");
            return;
        }

        if(normalized) {
            unnormalize();
        }

        for(j=0;j<16;j++) {
            for(k=0;k<4;k++) {
                nullvalh[j][k] = 0.;
                nullvalv[j][k] = 0.;
            }
        }
    }
}

```



```

    }
}

/* Calculate the sum of the data values over the selected region. */

for(i=spt1;i<spt2;i++) {
    for(j=0;j<16;j++) {
        if(j < 8) {
            lag = nlag;
        }
        else {
            lag = 0;
        }
        for(k=0;k<4;k++) {
            nullvalh[j][k] += DataType[j][k].hdata[(i+lag)%npts];
            nullvalv[j][k] += DataType[j][k].vdata[(i+lag)%npts];
        }
    }
}

/* Find the average of the data values over the selected region. */

for(j=0;j<16;j++) {
    for(k=0;k<4;k++) {
        nullvalh[j][k] /= (spt2 - spt1);
        nullvalv[j][k] /= (spt2 - spt1);
    }
}

for(j=0;j<16;j++) {
    for(k=0;k<4;k++) {
        nullvalh0[j][k] += nullvalh[j][k];
        nullvalv0[j][k] += nullvalv[j][k];
    }
}

/* TEMPORARY */

#if(0)
printf("Null values:\n");
for(j=0;j<16;j++) {
    for(k=0;k<4;k++) {
        printf("%f ",nullvalh[j][k]);
    }
    printf("\n");
    for(k=0;k<4;k++) {
        printf("%f ",nullvalv[j][k]);
    }
    printf("\n");
}
#endif

/* Subtract the average from all data points. */

for(i=0;i<npts;i++) {
    for(j=0;j<16;j++) {
        for(k=0;k<4;k++) {
            DataType[j][k].hdata[i] -= nullvalh[j][k];
            DataType[j][k].vdata[i] -= nullvalv[j][k];
        }
    }
}

nulled = 1;

#if(0)
if(normalized) {
    ToolsApplyNormCB(w,client_data,call_data);
}

```

```

    }

    if(defect_depth_calculated) {
        CalDefectDepthCB(w,client_data,call_data);
    }
#endif
}

/* Erase chart data */

XClearWindow(XtDisplay(chart_canvas),XtWindow(chart_canvas));
spt1 = -1;
spt2 = -1;
cursor_y = -1;

/* Redraw chart data */

XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
draw_chart_data();

return;
}

void ToolsRestoreNormCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int j,k;
    FILE *normfile;

    normfile = fopen("norm.dat","r");

    if(normfile == NULL) {
        printf("Unable to open data file for reading normalization factors.\n");
        return;
    }

    for(j=0;j<16;j++) {
        for(k=0;k<4;k++) {
            fscanf(normfile,"%f",&normfac[j][k]);
        }
    }

    for(j=0;j<16;j++) {
        for(k=0;k<4;k++) {
            fscanf(normfile,"%f",&phase[j][k]);
        }
    }

    fclose(normfile);

    return;
}

void ToolsSaveNormCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int j,k;
    FILE *normfile;

    normfile = fopen("norm.dat","w");

    if(normfile == NULL) {
        printf("Unable to open data file for writing normalization factors.\n");
        return;
    }

    for(j=0;j<16;j++) {
        for(k=0;k<4;k++) {
            fprintf(normfile,"%f ",normfac[j][k]);
        }
    }

```

```

    fprintf(normfile, "\n");
}

for(j=0; j<16; j++) {
    for(k=0; k<4; k++) {
        fprintf(normfile, "%f ", phase[j][k]);
    }
    fprintf(normfile, "\n");
}

fclose(normfile);

return;
}

void unnormalize(void)
{
    int i, j, k;
    float xtmp, ytmp;

    for(i=0; i<npts; i++) {
        for(j=0; j<16; j++) {
            for(k=0; k<4; k++) {
                DataType[j][k].hdata[i] *= normfac[j][k];
                DataType[j][k].vdata[i] *= normfac[j][k];
                xtmp = DataType[j][k].hdata[i];
                ytmp = DataType[j][k].vdata[i];
                DataType[j][k].hdata[i] = xtmp * cos(phase[j][k]) +
                    ytmp * sin(phase[j][k]);
                DataType[j][k].vdata[i] = -xtmp * sin(phase[j][k]) +
                    ytmp * cos(phase[j][k]);
            }
        }
    }

    return;
}

void ToolsComputeNormCB(Widget w, XtPointer client_data, XtPointer call_data)
{
    /* A null should be performed before this routine is called. */

    int i, j, k;
    float magnitude;
    float refphase[4];
    float xtmp, ytmp;

    if(spt1 == -1) {
        printf("No region selected.\n");
        return;
    }

    normalized = 1;

    for(j=0; j<16; j++) {
        for(k=0; k<4; k++) {
            normfac[j][k] = 0.;
        }
    }

    /* Find maxima of all channels */

    refphase[0] = 0.;
    refphase[1] = -5.02 * 3.1415927/180.;
    refphase[2] = 124.67 * 3.1415927/180.;
    refphase[3] = -172.21 * 3.1415927/180.;

```

```

for(i=spt1;i<spt2;i++) {
    for(j=0;j<16;j++) {
        for(k=0;k<4;k++) {
            magnitude = sqrt(DataType[j][k].hdata[i] * DataType[j][k].hdata[i]
                + DataType[j][k].vdata[i] * DataType[j][k].vdata[i]);
            if(magnitude > normfac[j][k]) {
                normfac[j][k] = magnitude;
                phase[j][k] = refphase[k] -
                    atan2(DataType[j][k].vdata[i],
                        DataType[j][k].hdata[i]);
            }
        }
    }
}

/* TEMPORARY */

for(j=0;j<16;j++) {
    normfac[j][2] *= 2.;
    normfac[j][3] *= 5.;
}

printf("Normalization factors:\n");
for(j=0;j<16;j++) {
    for(k=0;k<4;k++) {
        printf("%f ",normfac[j][k]);
    }
    printf("\n");
}

/* Normalize the data */

for(i=0;i<npts;i++) {
    for(j=0;j<16;j++) {
        for(k=0;k<4;k++) {
            DataType[j][k].hdata[i] /= normfac[j][k];
            DataType[j][k].vdata[i] /= normfac[j][k];
            xtmp = DataType[j][k].hdata[i];
            ytmp = DataType[j][k].vdata[i];
            DataType[j][k].hdata[i] = xtmp * cos(phase[j][k]) -
                ytmp * sin(phase[j][k]);
            DataType[j][k].vdata[i] = xtmp * sin(phase[j][k]) +
                ytmp * cos(phase[j][k]);
        }
    }
}

/* Erase chart data */

XClearWindow(XtDisplay(chart_canvas),XtWindow(chart_canvas));
spt1 = -1;
spt2 = -1;
cursor_y = -1;

/* Set spans */

for(i=0;i<16;i++) {
    for(j=0;j<4;j++) {
        DataType[i][j].span = 5.;
    }
}

/* Redraw chart data */

XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
draw_chart_data();

return;

```

```

}

void SetVoltsCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    /* A null should be performed before this routine is called. */

    int i,j,k;
    float magnitude;
    int lag;

    if(spt1 == -1) {
        printf("No region selected.\n");
        return;
    }

    for(j=0;j<16;j++) {
        for(k=0;k<4;k++) {
            voltfac[j][k] = 0.;
        }
    }

    /* Find maxima of all channels */

    for(i=spt1;i<spt2;i++) {
        for(j=0;j<16;j++) {
            if(j < 8) {
                lag = nlag;
            }
            else {
                lag = 0;
            }
            for(k=0;k<4;k++) {
                magnitude = sqrt(DataType[j][k].hdata[(i+lag)%npts]
                                * DataType[j][k].hdata[(i+lag)%npts]
                                + DataType[j][k].vdata[(i+lag)%npts]
                                * DataType[j][k].vdata[(i+lag)%npts]);
                if(magnitude > voltfac[j][k]) {
                    voltfac[j][k] = magnitude;
                }
            }
        }
    }

    set_volts();

    /* Erase chart data */

    XClearWindow(XtDisplay(chart_canvas),XtWindow(chart_canvas));
    spt1 = -1;
    spt2 = -1;
    cursor_y = -1;

    /* Redraw chart data */

    XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
    draw_chart_data();

    /* Redraw lissajous data */

    if(mode == 2) {
        for(i=0;i<8;i++) {
            XClearWindow(XtDisplay(liss_canvas[i]),XtWindow(liss_canvas[i]));
        }
        lpt1 = cpt1;
        lpt2 = cpt2;
        draw_liss_data();
        update_liss_labels(-1);
    }
}

```

```

    }
    return;
}

void set_volts(void)
{
    int i,j,k;

    voltage_set = 1;

    /* Normalize the data */
    for(i=0;i<npts;i++) {
        for(j=0;j<16;j++) {
            for(k=0;k<4;k++) {
                DataType[j][k].hdata[i] /= voltfac[j][k];
                DataType[j][k].vdata[i] /= voltfac[j][k];
            }
        }
    }

    /* Set spans */
    for(i=0;i<16;i++) {
        for(j=0;j<4;j++) {
            DataType[i][j].span = 5.;
        }
    }

    return;
}

void ToolsApplyNormCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int i,j,k;
    float xtmp,ymtp;

    normalized = 1;

    /* Normalize the data */
    for(i=0;i<npts;i++) {
        for(j=0;j<16;j++) {
            for(k=0;k<4;k++) {
                DataType[j][k].hdata[i] /= normfac[j][k];
                DataType[j][k].vdata[i] /= normfac[j][k];
                xtmp = DataType[j][k].hdata[i];
                ymtp = DataType[j][k].vdata[i];
                DataType[j][k].hdata[i] = xtmp * cos(phase[j][k]) -
                    ymtp * sin(phase[j][k]);
                DataType[j][k].vdata[i] = xtmp * sin(phase[j][k]) +
                    ymtp * cos(phase[j][k]);
            }
        }
    }

    /* Set spans */
    for(i=0;i<16;i++) {
        for(j=0;j<4;j++) {
            DataType[i][j].span = 5.;
        }
    }

    /* Erase chart data */
    XClearWindow(XtDisplay(chart_canvas),XtWindow(chart_canvas));
    spt1 = -1;

```

```

spt2 = -1;
cursor_y = -1;

/* Redraw chart data */

XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
draw_chart_data();

return;
}

void CalDefectDepthCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int i,j,k,l;
    long nn;
    int nlayers;
    int npe[3];
    int layer;
    int type;
    FILE *weights_file;
    float pe[3][33];
    float weight[2][33][20];
    float alpha,eta;
    float sum;
    float datr,dati;
    float datm,datp;

    weights_file = fopen("../bpn30/zgbest.wts","r");

    if(weights_file == NULL) {
        printf("Unable to open weights file for defect depth.\n");
        return;
    }

    for(i=0;i<16;i++) {
        for(j=0;j<=lasttype;j++) {
            if(DataType[i][j].property == 1) {
                type = j;
                if(!DataType[i][j].active) {
                    DataType[i][j].hdata = allocate_memory(npts);
                    DataType[i][j].vdata = allocate_memory(npts);
                    DataType[i][j].span = 200.;
                    DataType[i][j].rotation = 0.;
                    DataType[i][j].frequency = 999.;
                    DataType[i][j].mix = 0;
                    DataType[i][j].property = 1;
                    DataType[i][j].active = 1;
                }
                break;
            }
        }
    }

    if(type > lastactivetype) {
        lastactivetype = type;
    }

    defect_depth_calculated = 1;

    fscanf(weights_file,"%ld",&nn);
    fscanf(weights_file,"%d",&nlayers);
    printf("nn=%ld\n",nn);
    printf("nlayers=%d\n",nlayers);
    for(i=0;i<nlayers;i++) {
        fscanf(weights_file,"%d",&npe[i]);
        printf("npe[%d]=%d\n",i,npe[i]);
    }
}

```

```

    fscanf(weights_file,"%f %f",&alpha,&eta);

/* Read weights */

for(layer=0;layer<nlayers-1;layer++) {
    for(i=0;i<npe[layer];i++) {
        for(j=1;j<npe[layer+1];j++) {
            fscanf(weights_file,"%f",&weight[layer][i][j]);
            printf("%d %d %d %f\n",layer,i,j,weight[layer][i][j]);
        }
    }

    fclose(weights_file);

/* Calculate defect depth */

for(i=0;i<npts;i++) {
    for(j=0;j<16;j++) {
        for(k=0;k<3;k++) {
            datr = DataType[j][k+1].hdata[i];
            dati = DataType[j][k+1].vdata[i];
            pe[0][2*k+1] = datr;
            pe[0][2*k+2] = dati;
        }
    }

/* TEMPORARY */

/*    pe[0][1] /= 4.;
    pe[0][2] /= 4.;*/
    pe[0][3] /= 3.;
    pe[0][4] /= 3.;
    pe[0][5] /= 10.;
    pe[0][6] /= 10.;

    pe[0][0] = 1.;
    pe[1][0] = 1.;
    pe[2][0] = 1.;

    for(layer=1;layer<nlayers;layer++) {
        for(l=1;l<npe[layer];l++) {
            sum = 0.;
            for(k=0;k<npe[layer-1];k++) {
                sum += pe[layer-1][k] * weight[layer-1][k][l];
            }
            pe[layer][l] = 1.0/(1.0 + exp(-sum));
        }
    }
    DataType[j][type].hdata[i] = (pe[2][1] - 0.2)/0.006;
    DataType[j][type].vdata[i] = i/npts;
}

XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
for(i=0;i<ncharts;i++) {
    if(DataType[chart[i].coil-1][chart[i].type].property == 1) {
        erase_chart_data(i);
        draw_one_chart_data(i);
    }
}

return;
}

void ViewZoom_inCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    if(spt1 == -1) {
        return;
    }

```



```

    }

/* Erase chart data */

XClearWindow(XtDisplay(chart_canvas),XtWindow(chart_canvas));

cpt1 = spt1;
cpt2 = spt2;
spt1 = -1;
spt2 = -1;
sspt = -1;
cursor_y = -1;

XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
draw_chart_data();

return;
}

void ViewZoom_outCB(Widget w,XtPointer client_data,XtPointer call_data)
{
/* Erase chart data */

XClearWindow(XtDisplay(chart_canvas),XtWindow(chart_canvas));

cpt1 = 0;
cpt2 = npts;
spt1 = -1;
spt2 = -1;
sspt = -1;
cursor_y = -1;

XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
draw_chart_data();

return;
}

void draw_density_plot(void)
{
    int i,j;
    int pt,tpt;
    float max,d;
    float ppp;
    Dimension dp_height,dp_width;
    unsigned int block_height,block_width;
    float dp_level[8];
    float angle;
    float sa,ca;

    for(i=0;i<8;i++) {
        dp_level[i] = 2.*(i+1)/8.;
    }

    if(!file_open) {
        return;
    }

    for(i=0;i<8;i++) {
        dp_color[i] = dp_colors[dp_color_flag][i];
    }

    #if(0)
    for(i=0;i<dp_nlevels;i++) {
        XSetForeground(XtDisplay(dp_canvas),gc,dp_color[i]);
        XFillRectangle(XtDisplay(dp_canvas),XtWindow(dp_canvas),
            gc,50*i,100,50,100);
    }
    #endif
}

```

```

#endif

XtVaGetValues(dp_canvas,
               XmNwidth,&dp_width,
               XmNheight,&dp_height,
               NULL);

ppp = (cpt2-cpt1)/((float)dp_height);
printf("ppp = %f\n",ppp);
printf("cpt1 = %d,cpt2 = %d\n",cpt1,cpt2);

pt = cpt1;
do {
    for(i=0;i<8;i++) {
        angle = DataType[i][dp_type].rotation;
        angle -= 90. * 3.1415927/180.;
        sa = sin(angle);
        ca = cos(angle);
        max = 0.;
        tpt = pt;
        for(j=0;j<ppp;j++) {
            d = ca * DataType[i][dp_type].hdata[j+tpt+nlag]
              - sa * DataType[i][dp_type].vdata[j+tpt+nlag];
            d = fabs(d);
            if(max < d) {
                max = d;
            }
            tpt++;
        }
    }

/* Color block */

    if(max > dp_level[dp_nlevels-1]) {
        XSetForeground(XtDisplay(dp_canvas),gc,dp_color[dp_nlevels-1]);
    }
    else {
        for(j=0;j<dp_nlevels;j++) {
            if(max < dp_level[j]) {
                XSetForeground(XtDisplay(dp_canvas),gc,dp_color[j]);
                break;
            }
        }
    }

    block_width = dp_width/16.;
    block_height = 1/ppp;
    if(block_height < 1) {
        block_height = 1;
    }

    XFillRectangle(XtDisplay(dp_canvas),XtWindow(dp_canvas),
                   gc,2*i*block_width,
                   dp_height*(pt-cpt1)/(cpt2-cpt1),
                   block_width+1,block_height+1);
}

for(i=8;i<16;i++) {
    angle = DataType[i][dp_type].rotation;
    angle -= 90. * 3.1415927/180.;
    sa = sin(angle);
    ca = cos(angle);
    max = 0.;
    tpt = pt;
    for(j=0;j<ppp;j++) {
        d = ca * DataType[i][dp_type].hdata[j+pt]
          - sa * DataType[i][dp_type].vdata[j+pt];
        d = fabs(d);
        if(max < d) {

```

```

        max = d;
    }
    tpt++;
}

/* Color block */

if(max > dp_level[dp_nlevels-1]) {
    XSetForeground(XtDisplay(dp_canvas),gc,dp_color[dp_nlevels-1]);
}
else {
    for(j=0;j<dp_nlevels;j++) {
        if(max < dp_level[j]) {
            XSetForeground(XtDisplay(dp_canvas),gc,dp_color[j]);
            break;
        }
    }
}

block_width = dp_width/16.;
block_height = 1/ppp;
if(block_height < 1) {
    block_height = 1;
}

XFillRectangle(XtDisplay(dp_canvas),XtWindow(dp_canvas),gc,
                (2*(i-8)+1)*block_width,
                dp_height*(pt-cpt1)/(cpt2-cpt1),
                block_width+1,block_height+1);
}

pt = tpt;
} while(pt < cpt2);

printf("pt = %d\n",pt);

return;
}

void dp_colorCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    if(dp_color_flag) {
        dp_color_flag = 0;
    }
    else {
        dp_color_flag = 1;
    }

    draw_density_plot();

    return;
}

void dp_refreshCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    draw_density_plot();

    return;
}

void dp_dataCB(Widget w,XtPointer client_data,
               XmPushButtonCallbackStruct *cbs)
{
    XEvent *event = cbs->event;
    int i;
    char string[20];
    XmString cstring;

```

```

if(event->xbutton.state & ShiftMask) {
    do {
        dp_type--;
        if(dp_type < 0) {
            dp_type = lastactivetype;
        }
    } while((DataType[0][dp_type].active != 1)
            || (DataType[0][dp_type].property != 0));
}
else {
    do {
        dp_type++;
        if(dp_type > lastactivetype) {
            dp_type = 0;
        }
    } while((DataType[0][dp_type].active != 1)
            || (DataType[0][dp_type].property != 0));
}

sprintf(string,"%d %d",dp_type,dp_vertical_flag);
cstring = XmStringCreateSimple(string);
XtVaSetValues(dp_data_button,
               XmNlabelString,cstring,
               NULL);
XmStringFree(cstring);

return;
}

void ViewDensityPlotCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int i;

    /* Remove buttons 4 through 15 from the screen */

    if(mode == 1) {
        ncharts = 4;

        for(i=4;i<16;i++) {
            XtUnmanageChild(data_button[i]);
            XtUnmanageChild(coil_button[i]);
            XtUnmanageChild(data_label[i]);
        }

    /* Resize chart drawing area */

        XtVaSetValues(chart_canvas,
                       XmNrightAttachment,XmATTACH_POSITION,
                       XmNrightPosition,50,
                       NULL);
    }
    else if(mode == 2) {
        if(setting_rotation) {
            do_set_rotation();
        }

        if(setting_span) {
            setting_span = 0;
            do_set_span();
        }
    }

    /* Unmanage lissajous widgets */

    for(i=0;i<8;i++) {
        XtUnmanageChild(liss_canvas[i]);
        XtUnmanageChild(liss_coil_button[i]);
        XtUnmanageChild(liss_data_button[i]);
    }
}

```

```

        XtUnmanageChild(liss_data_label[i][0]);
        XtUnmanageChild(liss_data_label[i][1]);
        XtUnmanageChild(liss_sep[i]);
/* Unmanage span and rotation widgets if necessary */
/* ... */
    }
}
else if(mode == 3) {
    draw_density_plot();
    return;
}
else {
    printf("Error in ViewDensityPlotCB.\n");
    printf("mode = %d\n",mode);
    exit(-1);
}

mode = 3;

/* Manage density plot drawing area */

XtManageChild(dp_canvas);
XtManageChild(dp_refresh);
XtManageChild(dp_color_button);
XtManageChild(dp_data_button);

draw_density_plot();

return;
}

void ViewLissajousCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int i;

    if(mode == 2) {
        return;
    }

    if(mode == 1) {
        ncharts = 4;

/* Remove buttons 4 through 15 from screen */

        for(i=4;i<16;i++) {
            XtUnmanageChild(data_button[i]);
            XtUnmanageChild(coil_button[i]);
            XtUnmanageChild(data_label[i]);
        }

/* Resize chart drawing area */

        XtVaSetValues(chart_canvas,
            XmNrightAttachment,XmATTACH_POSITION,
            XmNrightPosition,50,
            NULL);
    }
    else if(mode == 3) {

/* Remove density plot widgets */

        XtUnmanageChild(dp_canvas);
        XtUnmanageChild(dp_refresh);
        XtUnmanageChild(dp_color_button);
        XtUnmanageChild(dp_data_button);
    }
}

```

```

mode = 2;

/* Manage lissajous separators */
for(i=0;i<8;i++) {
    XtManageChild(liss_sep[i]);
}

/* Manage lissajous buttons */
for(i=0;i<8;i++) {
    XtManageChild(liss_data_label[i][1]);
    XtManageChild(liss_data_label[i][0]);
    XtManageChild(liss_coil_button[i]);
    XtManageChild(liss_data_button[i]);
}

/* Manage lissajous drawing areas */
for(i=0;i<8;i++) {
    XtManageChild(liss_canvas[i]);
}

initialize_liss_display();

/* Redraw chart data */
if(file_open) {
    XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
    draw_chart_data();
}

return;
}

void ViewChartCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int i;

    if(mode == 1) {
        return;
    }

    if(mode == 2) {
/* Unmanage lissajous widgets */
        if(setting_rotation) {
            do_set_rotation();
        }

        if(setting_span) {
            setting_span = 0;
            do_set_span();
        }

        for(i=0;i<8;i++) {
            XtUnmanageChild(liss_canvas[i]);
            XtUnmanageChild(liss_coil_button[i]);
            XtUnmanageChild(liss_data_button[i]);
            XtUnmanageChild(liss_data_label[i][0]);
            XtUnmanageChild(liss_data_label[i][1]);
            XtUnmanageChild(liss_sep[i]);
        }
    }
    else if(mode == 3) {
        XtUnmanageChild(dp_canvas);
        XtUnmanageChild(dp_refresh);
    }
}

```

```

        XtUnmanageChild(dp_color_button);
        XtUnmanageChild(dp_data_button);
    }

    mode = 1;
    ncharts = 16;

/* Put buttons 4 through 15 on screen */

    for(i=4;i<16;i++) {
        XtManageChild(coil_button[i]);
        XtManageChild(data_button[i]);
        XtManageChild(data_label[i]);
    }

/* Resize chart drawing area */

    XtVaSetValues(chart_canvas,
        XmNrightAttachment,XmATTACH_FORM,
        NULL);

/* Redraw chart data */

    if(file_open) {
        XSetForeground(XtDisplay(chart_canvas),gc,color.foreground);
        draw_chart_data();
    }

    return;
}

Widget GetTopShell(Widget w)
{
    while(w && !XtIsWMShell(w)) {
        w = XtParent(w);
    }

    return w;
}

/* MIX ROUTINES */

void MixSetPolyCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    static Widget form;
    Widget radio_box,rowcol;
    Widget frame;
    Widget w2;
    char tmp[80];
    XmString cstring;
    int i,j;
    int n;
    int val;

    for(i=0;i<3;i++) {
        for(j=0;j<9;j++) {
            mix_polynomial_old[i][j] = mix_polynomial[i][j];
        }
    }

    n = 0;
    XtSetArg(args[n],XmNdeleteResponse,XmDESTROY);n++;
    XtSetArg(args[n],XmNautoUnmanage,False);n++;
    XtSetArg(args[n],XmNfractionBase,30);n++;
    form = XmCreateFormDialog(toplevel,"mixdialog",args,n);

    for(i=0;i<3;i++) {
        sprintf(tmp," Mix %d Polynomial",i+1);

```

```

cstring = XmStringCreateSimple(tmp);
n = 0;
XtSetArg(args[n], XmNLabelString, cstring); n++;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_POSITION); n++;
XtSetArg(args[n], XmNleftPosition, 10*i+2); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_POSITION); n++;
XtSetArg(args[n], XmNrightPosition, 10*(i+1)-2); n++;
w2 = XmCreateLabelGadget(form, tmp, args, n);
XtManageChild(w2);
XmStringFree(cstring);

n = 0;
XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(args[n], XmNtopWidget, w2); n++;
XtSetArg(args[n], XmNleftAttachment, XmATTACH_POSITION); n++;
XtSetArg(args[n], XmNleftPosition, 10*i+2); n++;
XtSetArg(args[n], XmNrightAttachment, XmATTACH_POSITION); n++;
XtSetArg(args[n], XmNrightPosition, 10*(i+1)-2); n++;
frame = XmCreateFrame(form, "frame", args, n);

n = 0;
XtSetArg(args[n], XmNpacking, XmPACK_COLUMN); n++;
XtSetArg(args[n], XmNnumColumns, 1); n++;
rowcol = XmCreateRowColumn(frame, "rowcol", args, n);

for(j=0; j<9; j++) {
    if(file_open) {
        if(j == 0) {
            sprintf(tmp, "Constant");
        }
        else if(j%2) {
            sprintf(tmp, "%d kHz H", freq[(int)((j-1)/2)];
        }
        else {
            sprintf(tmp, "%d kHz V", freq[(int)((j-1)/2)];
        }
    }
    else {
        if(j==0) {
            sprintf(tmp, "Constant");
        }
        else {
            if(j%2) {
                sprintf(tmp, "Freq %d H", (int)((j-1)/2 + 1));
            }
            else {
                sprintf(tmp, "Freq %d V", (int)((j-1)/2 + 1));
            }
        }
    }
}
w2 = XtVaCreateManagedWidget(tmp,
                               xmToggleButtonGadgetClass, rowcol,
                               NULL);

if(mix_polynomial[i][j]) {
    XmToggleButtonSetState(w2, True, False);
}

val = i*10 + j;
XtAddCallback(w2, XmNvalueChangedCallback, MixPolyChangedCB, val);
}
XtManageChild(rowcol);
XtManageChild(frame);
}

w2 = XtVaCreateManagedWidget("OK",
                               xmPushButtonGadgetClass, form,
                               XmNtopAttachment, XmATTACH_WIDGET,

```



```

        XmNtopWidget, frame,
        XmNbottomAttachment, XmATTACH_FORM,
        XmNleftAttachment, XmATTACH_POSITION,
        XmNleftPosition, 6,
        XmNrightAttachment, XmATTACH_POSITION,
        XmNrightPosition, 12,
        NULL);
XtAddCallback(w2, XmNactivateCallback, MixSetPolyOKCB, form);

w2 = XtVaCreateManagedWidget("Cancel",
        xmPushButtonGadgetClass, form,
        XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET,
        XmNtopWidget, w2,
        XmNbottomAttachment, XmATTACH_FORM,
        XmNleftAttachment, XmATTACH_POSITION,
        XmNleftPosition, 18,
        XmNrightAttachment, XmATTACH_POSITION,
        XmNrightPosition, 24,
        NULL);
XtAddCallback(w2, XmNactivateCallback, MixSetPolyOKCB, form);

XtManageChild(form);

return;
}

void MixPolyChangedCB(Widget w, XtPointer client_data, XtPointer call_data)
{
    XmToggleButtonCallbackStruct *state;
    int i;
    int mix;
    int component;

    state = (XmToggleButtonCallbackStruct *)call_data;
    i = (int)client_data;

    mix = (int)(i/10);
    component = i%10;

    if(state->set) {
        mix_polynomial[mix][component] = 1;
    }
    else {
        mix_polynomial[mix][component] = 0;
    }

    for(mix=0; mix<3; mix++) {
        for(component=0; component<9; component++) {
            printf("%d ", mix_polynomial[mix][component]);
        }
        printf("\n");
    }
    printf("\n");
    return;
}

void MixSetPolyOKCB(Widget w, XtPointer client_data, XtPointer call_data)
{
    Widget w2;

    w2 = (Widget)client_data;
    XtDestroyWidget(w2);

    return;
}

void MixSetPolyCancelCB(Widget w, XtPointer client_data, XtPointer call_data)
{

```

```

Widget w2;
int i,j;

w2 = (Widget)client_data;
XtDestroyWidget(w2);

for(i=0;i<3;i++) {
    for(j=0;j<9;j++) {
        mix_polynomial[i][j] = mix_polynomial_old[i][j];
    }
}

return;
}

void MixSetActiveCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    static Widget form;
    Widget radio_box;
    Widget w2;
    char tmp[80];
    XmString cstring;
    int i,j;
    int n;
    int val;

    for(i=0;i<4;i++) {
        mix_active_old[i] = mix_active[i];
    }

    n = 0;
    XtSetArg(args[n],XmNdeleteResponse,XmDESTROY);n++;
    XtSetArg(args[n],XmNautoUnmanage,False);n++;
    XtSetArg(args[n],XmNfractionBase,40);n++;
    form = XmCreateFormDialog(toplevel,"mixdialog",args,n);

    for(i=0;i<4;i++) {
        sprintf(tmp," mix %d ",i);
        cstring = XmStringCreateSimple(tmp);
        n = 0;
        XtSetArg(args[n],XmNlabelString,cstring);n++;
        XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM);n++;
        XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
        XtSetArg(args[n],XmNleftPosition,10*i+2);n++;
        XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION);n++;
        XtSetArg(args[n],XmNrightPosition,10*(i+1)-2);n++;
        w2 = XmCreateLabelGadget(form,tmp,args,n);
        XtManageChild(w2);
        XmStringFree(cstring);

        n = 0;
        XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
        XtSetArg(args[n],XmNtopWidget,w2);n++;
        XtSetArg(args[n],XmNleftAttachment,XmATTACH_POSITION);n++;
        XtSetArg(args[n],XmNleftPosition,10*i+2);n++;
        XtSetArg(args[n],XmNrightAttachment,XmATTACH_POSITION);n++;
        XtSetArg(args[n],XmNrightPosition,10*(i+1)-2);n++;
        radio_box = XmCreateRadioBox(form,"radio_box",args,n);
        w2 = XtVaCreateManagedWidget("Active",
                                     xmToggleButtonGadgetClass,radio_box,
                                     NULL);

        if(mix_active[i]) {
            XmToggleButtonSetState(w2,True,False);
        }

        val = 10*i;
        XtAddCallback(w2,XmNvalueChangedCallback,MixActivatedCB,val);
    }
}

```

```

w2 = XtVaCreateManagedWidget("Inactive",
                               xmToggleButtonGadgetClass,radio_box,
                               NULL);

if(!mix_active[i]) {
    XmToggleButtonSetState(w2,True,False);
}

val = 10*i + 1;
XtAddCallback(w2,XmNvalueChangedCallback,MixActivatedCB,val);
XtManageChild(radio_box);
}

w2 = XtVaCreateManagedWidget("OK",
                               xmPushButtonGadgetClass,form,
                               XmNtopAttachment,XmATTACH_WIDGET,
                               XmNtopWidget,radio_box,
                               XmNbottomAttachment,XmATTACH_FORM,
                               XmNleftAttachment,XmATTACH_POSITION,
                               XmNleftPosition,8,
                               XmNrightAttachment,XmATTACH_POSITION,
                               XmNrightPosition,16,
                               NULL);
XtAddCallback(w2,XmNactivateCallback,MixSetActiveOKCB,form);

w2 = XtVaCreateManagedWidget("Cancel",
                               xmPushButtonGadgetClass,form,
                               XmNtopAttachment,XmATTACH_OPPOSITE_WIDGET,
                               XmNtopWidget,w2,
                               XmNbottomAttachment,XmATTACH_FORM,
                               XmNleftAttachment,XmATTACH_POSITION,
                               XmNleftPosition,24,
                               XmNrightAttachment,XmATTACH_POSITION,
                               XmNrightPosition,32,
                               NULL);

XtAddCallback(w2,XmNactivateCallback,MixSetActiveCancelCB,form);

XtManageChild(form);

return;
}

void MixSetActiveOKCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    Widget w2;
    int mix;
    int i,j;

    w2 = (Widget)client_data;
    XtDestroyWidget(w2);

    for(mix=0;mix<4;mix++) {
        if(!mix_active[mix]) {
            if(mix_calculated[mix]) {
                mix_calculated[mix] = 0;
                for(i=0;i<16;i++) {
                    for(j=0;j<=lastactivetype;j++) {
                        if(DataType[i][j].mix) {
                            if(DataType[i][j].mixind == mix) {
                                DataType[i][j].active = 0;
                                free(DataType[i][j].hdata);
                                free(DataType[i][j].vdata);
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
}

return;
}

void MixSetActiveCancelCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    Widget w2;
    int i;

    w2 = (Widget)client_data;
    XtDestroyWidget(w2);

    for(i=0;i<4;i++) {
        mix_active[i] = mix_active_old[i];
    }

    return;
}

void MixActivatedCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    XmToggleButtonCallbackStruct *state;
    int i;
    int mix;
    int activated;

    state = (XmToggleButtonCallbackStruct *)call_data;
    i = (int)client_data;

    mix = (int)(i/10);
    activated = ((i+1)%2);

    if(state->set) {
        if(activated) {
            mix_active[mix] = 1;
        }
        else {
            mix_active[mix] = 0;
        }
    }

    for(mix=0;mix<4;mix++) {
        printf("%d %d\n",mix,mix_active[mix]);
    }
    return;
}

void MixSetOutputCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    static Widget form;
    Widget radio_box;
    Widget w2;
    char tmp[80];
    XmString cstring;
    int i,j;
    int n;
    int val;

    mix_output_old = mix_desired_output;

    n = 0;
    XtSetArg(args[n],XmDeleteResponse,XmDESTROY);n++;
    XtSetArg(args[n],XmAutoUnmanage,False);n++;
    XtSetArg(args[n],XmNfractionBase,40);n++;
    form = XmCreateFormDialog(toplevel,"mixdialog",args,n);

```

```

n = 0;
sprintf(tmp,"Desired Output For Mix");
cstring = XmStringCreateLocalized(tmp);
XtSetArg(args[n],XmNlabelString,cstring);n++;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM);n++;
w2 = XmCreateLabelGadget(form,"label",args,n);
XmStringFree(cstring);
XtManageChild(w2);

n = 0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNtopWidget,w2);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM);n++;
radio_box = XmCreateRadioBox(form,"radio_box",args,n);

for(i=0;i<4;i++) {
    sprintf(tmp,"Freq %d",i);
    w2 = XtVaCreateManagedWidget(tmp,
                                   xmToggleButtonGadgetClass,radio_box,
                                   NULL);

    val = i;
    XtAddCallback(w2,XmNvalueChangedCallback,MixOutputChangedCB,val);

    if(mix_desired_output == i) {
        XmToggleButtonSetState(w2,True,False);
    }
}

XtManageChild(radio_box);

w2 = XtVaCreateManagedWidget("OK",
                               xmPushButtonGadgetClass,form,
                               XmNtopAttachment,XmATTACH_WIDGET,
                               XmNtopWidget,radio_box,
                               XmNbottomAttachment,XmATTACH_FORM,
                               XmNleftAttachment,XmATTACH_POSITION,
                               XmNleftPosition,8,
                               XmNrightAttachment,XmATTACH_POSITION,
                               XmNrightPosition,16,
                               NULL);
XtAddCallback(w2,XmNactivateCallback,MixSetOutputOKCB,form);

w2 = XtVaCreateManagedWidget("Cancel",
                               xmPushButtonGadgetClass,form,
                               XmNtopAttachment,XmATTACH_OPPOSITE_WIDGET,
                               XmNtopWidget,w2,
                               XmNbottomAttachment,XmATTACH_FORM,
                               XmNleftAttachment,XmATTACH_POSITION,
                               XmNleftPosition,24,
                               XmNrightAttachment,XmATTACH_POSITION,
                               XmNrightPosition,32,
                               NULL);

XtAddCallback(w2,XmNactivateCallback,MixSetOutputCancelCB,form);

XtManageChild(form);

return;
}

void MixSetOutputOKCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    Widget w2;
    int mix;
    int i,j;

```

```

    w2 = (Widget)client_data;
    XtDestroyWidget(w2);

    return;
}

void MixSetOutputCancelCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    Widget w2;
    int i;

    w2 = (Widget)client_data;
    XtDestroyWidget(w2);

    mix_desired_output = mix_output_old;

    return;
}

void MixOutputChangedCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    XmToggleButtonCallbackStruct *state;
    int i;

    state = (XmToggleButtonCallbackStruct *)call_data;
    i = (int)client_data;

    if(state->set) {
        mix_desired_output = i;

        printf("mix_desired_output = %d\n",mix_desired_output);
    }

    return;
}

void MixApplyCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    int i,j,k;
    int m;
    int n;
    int type;
    float mixcal;

    for(m=1;m<=3;m++) {
        printf("m=%d,coeffs_calculated=%d,mix_calculated=%d\n",
            m,coeffs_calculated[m],mix_calculated[m]);
        if(!coeffs_calculated[m]) {
            continue;
        }

        for(i=0;i<=lasttype;i++) {
            if((DataType[0][i].mix) && (DataType[0][i].mixind == m)) {
                type = i;
            }
        }

        if(type > lastactivetype) {
            lastactivetype = type;
        }

        if(!mix_calculated[m]) {
            mix_calculated[m] = 1;

            printf("m=%d,type=%d\n",m,type);

            for(i=0;i<16;i++) {
                DataType[i][type].hdata = allocate_memory(npts);
            }
        }
    }
}

```

```

        DataType[i][type].vdata = allocate_memory(npts);
        DataType[i][type].span = DataType[i][mix_desired_output].span;
        DataType[i][type].rotation = DataType[i][mix_desired_output].rotation;
        DataType[i][type].frequency = 990 + m;
        DataType[i][type].mix = 1;
        DataType[i][type].mixind = m;
        DataType[i][type].property = 0;
        DataType[i][type].active = 1;
    }
}
for(i=0;i<16;i++) {
    for(k=0;k<npts;k++) {

/* Calculate mixed real part */

        n = 0;
        mixcal = 0;
        if(mix_polynomial[m-1][0]) {
            mixcal = coeffr[m-1][i][0];
            n++;
        }
        for(j=0;j<4;j++) {
            if(mix_polynomial[m-1][2*j+1]) {
                mixcal += coeffr[m-1][i][n] * DataType[i][j].hdata[k];
                n++;
            }
            if(mix_polynomial[m-1][2*j+2]) {
                mixcal += coeffr[m-1][i][n] * DataType[i][j].vdata[k];
                n++;
            }
        }
        DataType[i][type].hdata[k] = mixcal;

/* Calculate mixed imaginary part */

        n = 0;
        mixcal = 0.;
        if(mix_polynomial[m-1][0]) {
            mixcal = coeffi[m-1][i][0];
            n++;
        }
        for(j=0;j<4;j++) {
            if(mix_polynomial[m-1][2*j+1]) {
                mixcal += coeffi[m-1][i][n] * DataType[i][j].hdata[k];
                n++;
            }
            if(mix_polynomial[m-1][2*j+2]) {
                mixcal += coeffi[m-1][i][n] * DataType[i][j].vdata[k];
                n++;
            }
        }
        DataType[i][type].vdata[k] = mixcal;
    }
}
}

return;
}

void MixSaveCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    FILE *strout;
    int i,j,k,l;

    strout = fopen("m16mix.dat","w");
    if(strout == NULL) {
        printf("Unable to open output file.\n");
        return;
    }
}

```

```

    }

    for(i=0;i<=3;i++) {
        fprintf(strout,"%d ",mix_active[i]);
    }
    fprintf(strout,"\n");

    for(i=0;i<=3;i++) {
        fprintf(strout,"%d ",coeffs_calculated[i]);
    }
    fprintf(strout,"\n");

    for(i=1;i<=3;i++) {
        if(!coeffs_calculated[i]) {
            continue;
        }
        for(j=0;j<9;j++) {
            fprintf(strout,"%d ",mix_polynomial[i-1][j]);
        }
        fprintf(strout,"\n");
        for(k=0;k<16;k++) {
            l = 0;
            for(j=0;j<9;j++) {
                if(mix_polynomial[i-1][j]) {
                    fprintf(strout,"%f\n",coeffr[i-1][k][l]);
                    l++;
                }
            }
            l = 0;
            for(j=0;j<9;j++) {
                if(mix_polynomial[i-1][j]) {
                    fprintf(strout,"%f\n",coeffi[i-1][k][l]);
                    l++;
                }
            }
        }
    }
}

fclose(strout);

return;
}

void MixRestoreCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    FILE *strin;
    int i,j,k,l;
    int nterms;

    strin = fopen("m16mix.dat","r");
    if(strin == NULL) {
        printf("Unable to open input file.\n");
        return;
    }

    for(i=0;i<=3;i++) {
        fscanf(strin,"%d",&mix_active[i]);
    }

    for(i=0;i<=3;i++) {
        fscanf(strin,"%d",&coeffs_calculated[i]);
    }

    for(i=1;i<=3;i++) {
        if(!coeffs_calculated[i]) {
            continue;
        }
        nterms = 0;

```



```

for(j=0;j<9;j++) {
    fscanf(strin,"%d",&mix_polynomial[i-1][j]);
    nterms += mix_polynomial[i-1][j];
}
for(k=0;k<16;k++) {
    coeffr[i-1][k] = calloc((size_t)nterms,sizeof(float));
    coeffi[i-1][k] = calloc((size_t)nterms,sizeof(float));
    if((coeffr[i-1][k] == NULL) || (coeffi[i-1][k] == NULL)) {
        printf("Error allocating memory.\n");
        exit(-1);
    }
    l = 0;
    for(j=0;j<9;j++) {
        if(mix_polynomial[i-1][j]) {
            fscanf(strin,"%f",&coeffr[i-1][k][l]);
            l++;
        }
    }
    l = 0;
    for(j=0;j<9;j++) {
        if(mix_polynomial[i-1][j]) {
            fscanf(strin,"%f",&coeffi[i-1][k][l]);
            l++;
        }
    }
}
}
}

fclose(strin);

return;
}

void MixComputeCoefficientsCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    Widget form;
    Widget w2;
    char tmp[80];
    XmString cstring;
    int n;

    n = 0;
    XtSetArg(args[n],XmNautoUnmanage,False);n++;
    XtSetArg(args[n],XmNdeleteResponse,XmDESTROY);n++;
    form = XmCreateFormDialog(toplevel,"mixdialog",args,n);

    sprintf(tmp,"Preserve");
    cstring = XmStringCreateLocalized(tmp);
    n = 0;
    XtSetArg(args[n],XmNlabelString,cstring);n++;
    XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM);n++;
    XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM);n++;
    w2 = XmCreatePushButtonGadget(form,tmp,args,n);
    XtManageChild(w2);
    XmStringFree(cstring);
    XtAddCallback(w2,XmNactivateCallback,MixPreserveCB,NULL);

    sprintf(tmp,"Suppress");
    cstring = XmStringCreateLocalized(tmp);
    n = 0;
    XtSetArg(args[n],XmNlabelString,cstring);n++;
    XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM);n++;
    XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM);n++;
    XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
    XtSetArg(args[n],XmNleftWidget,w2);n++;
    w2 = XmCreatePushButtonGadget(form,tmp,args,n);
    XtManageChild(w2);
}

```

```

XmStringFree(cstring);
XtAddCallback(w2,XmNactivateCallback,MixSuppressCB,NULL);

sprintf(tmp,"Calculate");
cstring = XmStringCreateLocalized(tmp);
n = 0;
XtSetArg(args[n],XmNlabelString,cstring);n++;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNleftWidget,w2);n++;
w2 = XmCreatePushButtonGadget(form,tmp,args,n);
XtManageChild(w2);
XmStringFree(cstring);
XtAddCallback(w2,XmNactivateCallback,MixCalculateCB,NULL);

sprintf(tmp,"Cancel");
cstring = XmStringCreateLocalized(tmp);
n = 0;
XtSetArg(args[n],XmNlabelString,cstring);n++;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM);n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET);n++;
XtSetArg(args[n],XmNleftWidget,w2);n++;
w2 = XmCreatePushButtonGadget(form,tmp,args,n);
XtManageChild(w2);
XmStringFree(cstring);
XtAddCallback(w2,XmNactivateCallback,MixCancelCB,NULL);

XtManageChild(form);

return;
}

void MixPreserveCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    if((spt1 > -1) && (spt2 > -1)) {
        mix_pres_pt1[mix_nreg_pres] = spt1;
        mix_pres_pt2[mix_nreg_pres] = spt2;
        mix_nreg_pres++;
        printf("%d %d\n",spt1,spt2);
    /* Unselect region */
    /* ... */
    }
    else {
        printf("No region selected.\n");
    }

    return;
}

void MixSuppressCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    if((spt1 > -1) && (spt2 > -1)) {
        mix_sup_pt1[mix_nreg_sup] = spt1;
        mix_sup_pt2[mix_nreg_sup] = spt2;
        mix_nreg_sup++;
        printf("%d %d\n",spt1,spt2);
    /* Unselect region */
    /* ... */
    }
    else {
        printf("No region selected.\n");
    }

    return;
}

```

```

void MixCancelCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    Widget w2;

    /* Set number of regions to preserve and suppress equal to zero. */

    mix_nreg_pres = 0;
    mix_nreg_sup = 0;

    /* Remove dialog box */

    w2 = XtParent(XtParent(w));
    XtDestroyWidget(w2);

    return;
}

void MixCalculateCB(Widget w,XtPointer client_data,XtPointer call_data)
{
    Widget w2;
    int mix_desired_output = 1;
    int ninclude;
    float *mixdat;
    float *mixdat1;
    float *prop;
    int maxdegree;
    int secdim;
    int nterms;
    float *expmixdat;
    float *rdg1;
    int offset_flag;
    float rsos;
    float mixcal;
    int type;
    int i,j,k,l;
    int m;
    int n;
    int lag;

    /* Remove dialog box */

    w2 = XtParent(XtParent(w));
    XtDestroyWidget(w2);

    if(mix_nreg_pres < 1) {
        mix_nreg_pres = 0;
        mix_nreg_sup = 0;
        printf("No region selected to preserve.\n");
        return;
    }

    if(mix_nreg_sup < 1) {
        mix_nreg_pres = 0;
        mix_nreg_sup = 0;
        printf("No region selected to suppress.\n");
        return;
    }

    ninclude = 0;
    for(i=0;i<mix_nreg_pres;i++) {
        ninclude += mix_pres_pt2[i] - mix_pres_pt1[i];
    }
    for(i=0;i<mix_nreg_sup;i++) {
        ninclude += mix_sup_pt2[i] - mix_sup_pt1[i];
    }

    printf("ninclude = %d\n",ninclude);
}

```

```

mixdat = calloc((size_t)(ninclud*8),sizeof(float));
if(mixdat == NULL) {
    printf("Error allocating memory.\n");
    exit(-1);
}

prop = calloc((size_t)ninclud, sizeof(float));
if(prop == NULL) {
    printf("Error allocating memory.\n");
    exit(-1);
}

for(m=0;m<3;m++) {
    if(!mix_active[m+1]) {
        break;
    }

    coeffs_calculated[m+1] = 1;
    mix_calculated[m+1] = 1;

    nterms = 0;
    for(i=0;i<9;i++) {
        nterms += mix_polynomial[m][i];
    }

    offset_flag = mix_polynomial[m][0];

    maxdegree = 0;
    for(i=1;i<9;i++) {
        if(mix_polynomial[m][i] > maxdegree) {
            maxdegree = mix_polynomial[m][i];
        }
    }

    secdim = ninclud + 1;

    expmixdat = calloc((size_t)(secdim*(nterms+1)),sizeof(float));
    if(expmixdat == NULL) {
        printf("Error allocating memory.\n");
        exit(-1);
    }

    rdg1 = calloc((size_t)nterms,sizeof(float));
    if(rdg1 == NULL) {
        printf("Error allocating memory.\n");
        exit(-1);
    }

    for(i=0;i<16;i++) {
        coeffr[m][i] = calloc((size_t)nterms,sizeof(float));
        coeffi[m][i] = calloc((size_t)nterms,sizeof(float));
        if((coeffr[m][i] == NULL) || (coeffi[m][i] == NULL)) {
            printf("Error allocating memory.\n");
            exit(-1);
        }
    }

    for(i=0;i<=lasttype;i++) {
        if((DataType[0][i].mix) && (DataType[0][i].mixind == m+1)) {
            type = i;
        }
    }

    if(type > lastactivetype) {
        lastactivetype = type;
    }

    printf("m=%d,type=%d\n",m,type);

```

```

for(i=0;i<16;i++) {
    DataType[i][type].hdata = allocate_memory(npts);
    DataType[i][type].vdata = allocate_memory(npts);
    DataType[i][type].span = DataType[i][mix_desired_output].span;
    DataType[i][type].rotation = DataType[i][mix_desired_output].rotation;
    DataType[i][type].frequency = 990 + m;
    DataType[i][type].mix = 1;
    DataType[i][type].mixind = m+1;
    DataType[i][type].property = 0;
    DataType[i][type].active = 1;
}

for(i=0;i<16;i++) {

/* Calculate coefficients for real part */

    if(i < 8) {
        lag = nlag;
    }
    else {
        lag = 0.;
    }
    n = 0;
    mixdatl = mixdat;
    for(j=0;j<mix_nreg_pres;j++) {
        for(k=mix_pres_pt1[j];k<mix_pres_pt2[j];k++) {
            for(l=0;l<4;l++) {
                mixdatl[2*l] = DataType[i][l].hdata[(k+lag)%npts];
                mixdatl[2*l+1] = DataType[i][l].vdata[(k+lag)%npts];
            }
            prop[n] = DataType[i][mix_desired_output].hdata[(k+lag)%npts];
            rdgexp(rdg1,mixdatl,offset_flag,8,maxdegree,m);
            for(l=0;l<nterms;l++) {
                *(expmixdat + l*secdim + n) = rdg1[l];
            }
            mixdatl += 8;
            n++;
        }
    }
    for(j=0;j<mix_nreg_sup;j++) {
        for(k=mix_sup_pt1[j];k<mix_sup_pt2[j];k++) {
            for(l=0;l<4;l++) {
                mixdatl[2*l] = DataType[i][l].hdata[(k+lag)%npts];
                mixdatl[2*l+1] = DataType[i][l].vdata[(k+lag)%npts];
            }
            prop[n] = 0.;
            rdgexp(rdg1,mixdatl,offset_flag,8,maxdegree,m);
            for(l=0;l<nterms;l++) {
                *(expmixdat + l*secdim + n) = rdg1[l];
            }
            mixdatl += 8;
            n++;
        }
    }

    alsqs(expmixdat,prop,coeffr[m][i],&rsos,n,nterms,secdim);

/* Calculate coefficients for imaginary part */

    n = 0;
    mixdatl = mixdat;
    for(j=0;j<mix_nreg_pres;j++) {
        for(k=mix_pres_pt1[j];k<mix_pres_pt2[j];k++) {
            for(l=0;l<4;l++) {
                mixdatl[2*l] = DataType[i][l].hdata[(k+lag)%npts];
                mixdatl[2*l+1] = DataType[i][l].vdata[(k+lag)%npts];
            }
            prop[n] = DataType[i][mix_desired_output].vdata[(k+lag)%npts];

```

```

        rdgexp(rdg1,mixdatl,offset_flag,8,maxdegree,m);
        for(l=0;l<nterms;l++) {
            *(expmixdat + l*secdim + n) = rdg1[l];
        }
        mixdatl += 8;
        n++;
    }
}
for(j=0;j<mix_nreg_sup;j++) {
    for(k=mix_sup_pt1[j];k<mix_sup_pt2[j];k++) {
        for(l=0;l<4;l++) {
            mixdatl[2*l] = DataType[i][l].hdata[(k+lag)%npts];
            mixdatl[2*l+1] = DataType[i][l].vdata[(k+lag)%npts];
        }
        prop[n] = 0.;
        rdgexp(rdg1,mixdatl,offset_flag,8,maxdegree,m);
        for(l=0;l<nterms;l++) {
            *(expmixdat + l*secdim + n) = rdg1[l];
        }
        mixdatl += 8;
        n++;
    }
}

alsqs(expmixdat,prop,coeffi[m][i],&rsos,n,nterms,secdim);

for(j=0;j<nterms;j++) {
    printf("%d %d %f %f\n",i,j,coeffr[m][i][j],coeffi[m][i][j]);
}

for(k=0;k<npts;k++) {
/* Calculate mixed real part */

    n = 0;
    mixcal = 0.;
    if(mix_polynomial[m][0]) {
        mixcal = coeffr[m][i][0];
        n++;
    }

    for(j=0;j<4;j++) {
        if(mix_polynomial[m][2*j+1]) {
            mixcal += coeffr[m][i][n] * DataType[i][j].hdata[k];
            n++;
        }
        if(mix_polynomial[m][2*j+2]) {
            mixcal += coeffr[m][i][n] * DataType[i][j].vdata[k];
            n++;
        }
    }
    DataType[i][type].hdata[k] = mixcal;

/* Calculate mixed imaginary part */

    n = 0;
    mixcal = 0.;
    if(mix_polynomial[m][0]) {
        mixcal = coeffi[m][i][0];
        n++;
    }

    for(j=0;j<4;j++) {
        if(mix_polynomial[m][2*j+1]) {
            mixcal += coeffi[m][i][n] * DataType[i][j].hdata[k];
            n++;
        }
        if(mix_polynomial[m][2*j+2]) {

```

```

        mixcal += coeffi[m][i][n] * DataType[i][j].vdata[k];
        n++;
    }
    }
    DataType[i][type].vdata[k] = mixcal;
}
}
free(expmixdat);
free(rdg1);
}

free(mixdat);
free(prop);

return;
}

void rdgexp(float *readng, float *dat, int offset_flag, int npoly, int maxdegree, int m)
{
    int nterm, ndeg;
    int i, j, k;
    int n;
    float rdg = 0.0;
    float rdy, rinv;
    float prev_rdg;
    float *preadng;
    float *pdat;
    float *pdelta;
    double rdgpow[16][5];
    int *pdegree, *pcrstrm;
    int poly[8][8];

    for(i=0; i<8; i++) {
        for(j=0; j<8; j++) {
            if(i == j) {
                poly[i][j] = mix_polynomial[m][i+1];
            }
            else {
                poly[i][j] = 0;
            }
        }
    }

    preadng = readng;

    if(offset_flag) {
        *preadng = 1.;
        preadng++;
    }

    pdat = dat;
    for(n=0; n<npoly; n++) {
        rdgpow[n][0] = 1.0;
        rdgpow[n][1] = *(dat+n);
        for(k=2; k<maxdegree+1; k++){
            rdgpow[n][k] = pow(dat[n], k);
        }
    }

    for(i=0; i<npoly; i++) {
        for(j=0; j<npoly; j++) {
            if(poly[i][j] == 0) {
                continue;
            }
            if(i==j) {
                for(k=1; k<=poly[i][j]; k++) {
                    *preadng = rdgpow[i][k];
                    preadng++;
                }
            }
        }
    }
}

```

```

        else {
            for(k=0; k<poly[i][j];k++) {
                *preadng = rdgpow[i][poly[i][j] - k]
                    * rdgpow[j][k+1];
                preadng++;
            }
        }
    }
}
return;
}

```

```

void alsqs(float *a, float *y, float *b, float *r2, int drows, int dcols,
int ncols)

```

```

{
    int n, m;
    int mm1;
    int i, j, l, l1, ll;
    double s2;
    float ss, s, pp;
    float d;
    float *q1, *q2, *q3;

    n = drows;
    m = dcols;
    mm1 = m - 1;

    /* Reduce the least squares matrix to upper triangular form. */

```

```

    for(l=0; l<m; l++) {
        ss = 0.;
        for(i=l; i<n; i++) {
            ss += (*(a+l*ncols+i)) * (*(a+l*ncols+i));
        }
        s2 = ss;
        s = sqrt(s2);
        if(*(a+l*ncols+l) < 0.) {
            s = -s;
        }
        d = s2 + s*(*(a+l*ncols+l));
        *(a+l*ncols+l) += s;
        if(l != mm1) {
            l1 = l+1;
            for(j=l1; j<m; j++) {
                pp = 0.;
                for(i=l; i<n; i++) {
                    pp += (*(a+l*ncols+i))*(*(a+j*ncols+i));
                }
                *(a+j*ncols+n) = pp/d;
            }
            for(j=l1; j<m; j++) {
                q1 = a + j*ncols + l;
                q2 = a + l*ncols + l;
                q3 = a + j*ncols + n;
                for(i=l; i<n; i++) {
                    *q1 -= (*q2)*(q3);
                    q1++;
                    q2++;
                }
            }
        }
        *(a+l*ncols+n) = -s;
    }
}

```

```

/* Reduce the vector y. */

```

```

    for(i=0; i<n; i++) {
        *(a+m*ncols+i) = *(y+i);
    }

```



```

    }
    for(l=0; l<m; l++) {
        pp = 0.;
        for(i=l; i<n; i++) {
            pp += (*(a+l*ncols+i))*(*(a+m*ncols+i));
        }
        d = pp/(-(*(a+l*ncols+l))*(*(a+l*ncols+n)));
        for(i=l; i<n; i++) {
            *(a+m*ncols+i) -= d*(*(a+l*ncols+i));
        }
    }

/* Calculate the coefficient vector b. */

*(b+mm1) = *(a+m*ncols+mm1)/(*(a+mm1*ncols+n));
if(m != 0) {
    for(ll=0; ll<mm1; ll++) {
        l = mm1 - ll - 1;
        l1 = l + 1;
        pp = *(a+m*ncols+l);
        for(i=l1; i<m; i++) {
            pp -= *(a+i*ncols+l)*(*(b+i));
        }
        *(b+l) = pp/(*(a+l*ncols+n));
    }
}

/* Calculate r2. */

ss = 0.;
for(i=m; i<n; i++) {
    /*ss += (*(a+m*ncols+i))*(*(a+m*ncols+i));*/
    *(a+m*ncols+i) = 0.;
}
/**r2 = ss;*/

/* Perform the back calculations. */

for(ll=0; ll<m; ll++) {
    l = m - ll - 1;
    pp = 0.;
    for(i=l; i<n; i++) {
        pp += (*(a+l*ncols+i))*(*(a+m*ncols+i));
    }
    d = -pp/(*(a+l*ncols+l))*(*(a+l*ncols+n));
    for(i=l; i<n; i++) {
        *(a+m*ncols+i) -= d*(*(a+l*ncols+i));
    }
}
return;
}

/* END OF MIX ROUTINES */

```

SCAN18.C - Program to acquire data from a MIZ-18

Program SCAN18.C uses the Zetec MIZ-18 eddy current instrument to take data from a probe. The program will work with either absolute or differential circumferential probes or with a single pancake coil. The probe is positioned with the Modulynx motion controller. Communication between the computer and the MIZ-18 and motion controller is done over the GPIB bus using a Capital Equipment PC-488 board.

The program will take data at up to four frequencies. The desired frequencies should be entered in the frequency array in the program in kilohertz. To tell the program what type of probe will be used, set the element of the coils array to 0x01 for a differential bobbin or pancake coil, to 0x10 for an absolute coil, or to 0x11 for both absolute and differential bobbin coils. The variable ncoils should be set to 4 for pancake, absolute, or differential coils and to 8 if both absolute and differential data are desired.

The probe positions are read from a data file written by program DSPOSIT.C. The movement of the probe is handled by subroutine move.

Some variables of interest in the program are described below.

adat	Array containing the averaged data
config_string	The MIZ-18 configuration string returned from function fill_configuration_string
data	Array to receive the data transmitted from the MIZ-18
nrds	Number of readings to take and average together at each probe position

A complete listing of program SCAN18.C is given below.

```
/*
 * Program SCAN18.C
 * Version 14 July 1994
 *
 * This program uses the Capital Equipment PC-488 board
 * and the Modulynx motion controller to position a probe
 * on a sample and uses the Zetec MIZ-18 to take data from
 * the probe.
 *
 * The program will drive either absolute and differential
 * bobbin probes or a single pancake probe.
 *
 * The program compiles correctly under Borland C++ for DOS.
 * It must be in a project with the file IEEE488.LIB.
 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <dos.h>
#include <string.h>
#include "ieee-c.h"
```

```

void move(float *x);
void shutdown(void);
void handle_error(int);

void fill_configuration_string(char configuration_string[61]);
int freq_table_index(int my_frequency);
void select_drive(void);
void set_channel_masks(void);
void calculate_wait_time(void);
void position(float xnew[3], int naxis, int motor_bus);
void draw_line(short x1, short y1, short x2, short y2);

int frequency[4]; /* up to 4 MIZ-18 frequencies can be set */
char coils[4]; /* up to 4 coils per frequency can be set */
int trash_count; /* number of trash bytes preceding valid data */
char configl_string[61]; /* downloading this string sets up miz18 low f */
char configh_string[61]; /* downloading this string sets up miz18 hi f */
int istat; /* IEEE-488 bus status word */
int delay1; /* time between samples in microseconds */

int motor_address = 6;
char motor_off_string[] = "XE YE ZE";
char ymotor_set[] = "YA6500YB335YH2500";
char zmotor_set[] = "ZA18000ZB500ZH3000";
int naxes;
int motor_on[3] = {0,0,0};
char axis_label[80];

void main(void)
{
    FILE *position_file;
    FILE *str1;
    float x[3];
    int controller_address = 21;
    int motor_address = 6;
    int level = 0;
    int status;
    int i,j,k;
    char tmp_string[80];
    char motor_on_string[80];
    unsigned concnt = 60;
    int nrdgs = 5;
    int rdgcnt;
    int nrec;
    char eoi = 1;
    int ncoils;
    unsigned char data[1024];
    long adat[16];
    int *p;
    int bytes_per_slice;
    int use;

    /* Initialize GP-IB board */

    initialize(controller_address,level);

    /* Turn off all motors */

    send(motor_address,ymotor_set,&status);
    send(motor_address,zmotor_set,&status);
    send(motor_address,motor_off_string,&status);
    if(status) {
        handle_error(status);
    }
    sleep(2);

    str1 = fopen("ddata.dat","w");
    if(str1 == NULL) {

```

```

        printf("Error opening output data file.\n");
        exit(-1);
    }

/* Open position data file */

    position_file = fopen("dsposit.dat","r");
    if(position_file == NULL) {
        printf("Position file not found.\n");
        exit(-1);
    }

/* Read header information */

    fscanf(position_file,"%d",&naxes);
    fscanf(position_file,"%s",axis_label);

/* Remove white space from axis_label */

    i = 0;
    while (axis_label[i]) {
        if(isspace(axis_label[i])) {
            for(j=i+1;j<strlen(axis_label);j++) {
                axis_label[j-1] = axis_label[j];
            }
            i++;
        }
    }

    if(strlen(axis_label) != naxes) {
        printf("Variable naxes conflicts with axis_label.\n");
        printf("  naxes = %d\n", naxes);
        printf("  axis_label = %s\n", axis_label);
        printf("  strlen(axis_label) = %d\n", strlen(axis_label));
        exit(-1);
    }

    for(i=0;i<strlen(axis_label);i++) {
        if((axis_label[i] == 'X') || (axis_label[i] == 'x')) {
            motor_on[0] = 1;
        }
        else if((axis_label[i] == 'Y') || (axis_label[i] == 'y')) {
            motor_on[1] = 1;
        }
        else if((axis_label[i] == 'Z') || (axis_label[i] == 'z')) {
            motor_on[2] = 1;
        }
        else {
            printf("Unrecognized character %c in axis_label.\n",axis_label[i]);
            exit(-1);
        }
    }

/* Generate string to turn motors on */

    for(i=0;i<naxes;i++) {
        sprintf(tmp_string," %cD",axis_label[i]);
        if(i==0) {
            strcpy(motor_on_string,tmp_string);
        }
        else {
            strcat(motor_on_string,tmp_string);
        }
    }

/* Turn on the motors that will be used */

    send(motor_address,motor_on_string,&status);

```

```

    if(status) {
        handle_error(status);
    }
    sleep(5);

/* The elements of coils should be set to
   0x11 for absolute and differential
   0x10 for absolute only
   0x01 for differential only
   0x01 for pancake */

    coils[0] = 0x01;
    coils[1] = 0x01;
    coils[2] = 0x01;
    coils[3] = 0x01;

    frequency[0] = 520;
    frequency[1] = 260;
    frequency[2] = 130;
    frequency[3] = 65;

    delay1 = 2500;
    ncoils = 4;

    fill_configuration_string(config_string);

    bytes_per_slice = 4 * ncoils;
    rdgcnt = (nrds + 4)*bytes_per_slice + trash_count;

    dmachannel(1);

/* Read positions from file and move axes */

    for(;;) {
        for(i=0;i<naxes;i++) {
            if(fscanf(position_file,"%f",&(x[i])) == EOF) {
                shutdown();
            }
        }

        move(x);

/* Send configuration string to MIZ-18 */

        transmit("UNL UNT MTA LISTEN 1",&status);
        tarray(config_string,concnt,eoi,&status);

/* Get the data from the MIZ-18 */

        transmit("UNL UNT TALK 1 MLA",&status);

        rarray(data,rdgcnt,&nrec,&status);
        if(status) {
            printf("status = %d\n",status);
            printf("rdgcnt = %d\n",rdgcnt);
            printf("nrec = %d\n",nrec);
        }
        for(i=0;i<2*ncoils;i++) {
            adat[i] = 0;
        }

        use = 1;

        for(k=0;k<nrds;k++) {
            for(i=0;i<ncoils;i++) {
                j = (k + 4)*bytes_per_slice + 4*i + trash_count;
                adat[2*i] += (int)(256*(long)data[j] + (long)data[j+1]);
            }
        }
    }

```

```

        adat[2*i+1] += (int)(256*(long)data[j+2] + (long)data[j+3]);
    }
}

if(use) {
    for(i=0;i<2*ncoils;i++) {
        adat[i] = adat[i]/((float)nrds);
        fprintf(str1,"%ld ",adat[i]);
        printf("%ld ",adat[i]);
    }

    for(i=0;i<naxes;i++) {
        fprintf(str1,"%5.3f ",x[i]);
        printf("%5.3f ",x[i]);
    }

    fprintf(str1,"\n");
    printf("\n");
}
}

void shutdown(void)
{
    FILE *str1;
    float x[3];
    int status;
    int i;

    /* Send motors home */

    for(i=0;i<naxes;i++) {
        x[i] = 0.;
    }

    move(x);

    sleep(5);

    /* Turn all motors off */

    send(motor_address,motor_off_string,&status);
    if(status) {
        handle_error(status);
    }

    exit(0);
}

void move(float *x)
{
    /*
     * This subroutine sends the commands necessary to move the
     * axes to the positions specified in inches in the array x.
     * After the movement is completed, the positions are checked
     * and corrected if necessary.
     */
    static long iprev_pos[3] = {0,0,0}; /* Previous position in
                                         motor steps */
    long isteps[3]; /* Distance to move in
                     motor steps */

    int status;
    int i;
    long n;
    unsigned wait_time;
    char tmp_string[80];

```

```

char move_string[80];
char check_pos_string[] = "<CFA>";
char semicolon_string[] = ";";
float step_size[3];

/* The variable step_size is the number of inches per step
for axes in which the motion is translation along a line
and is the number of degrees per step for axes in which the
motion is a rotation. Normally the z axis is used for
rotation. */

for(i=0;i<naxes;i++) {
    if(axis_label[i] == 'Z') {
/* 40 steps per degree */
        step_size[i] = 0.025;
    }
    else {
/* 4 steps per mil */
        step_size[i] = 0.00025;
    }
}

/* Calculate the number of motor steps each axis must take
to reach the desired position. */

for(i=0;i<naxes;i++) {
    n = (long)((x[i] + 0.5*step_size[i])/step_size[i]);
    isteps[i] = n - iprev_pos[i];
    iprev_pos[i] = n;
}

/* Generate the command string for the motion */

for(i=naxes-1;i>=0;i--) {
    sprintf(tmp_string,"%c",axis_label[i]);
    strcat(tmp_string,"M");
    if(i == (naxes-1)) {
        strcpy(move_string,tmp_string);
    }
    else {
        strcat(move_string,tmp_string);
    }
    sprintf(tmp_string,"%ldG ",isteps[i]);
    strcat(move_string,tmp_string);
}

/* Send the motion command */

send(motor_address,move_string,&status);
if(status) {
    handle_error(status);
}

/* Pause for the greater of 1 second and 1 second per 1000
motor counts to give the motors time to complete their motion.
The variable wait_time is the number of milliseconds the program
will pause. */

wait_time = 1000;
for(i=0;i<naxes;i++) {
    if(wait_time < labs(isteps[i])) {
        wait_time = labs(isteps[i]);
    }
}
delay(wait_time);

/* Compare pulse count with encoder count and correct position
if they disagree. */

```

```

    send(motor_address,check_pos_string,&status);
    if(status) {
        handle_error(status);
    }

    send(motor_address,semicolon_string,&status);
    if(status) {
        handle_error(status);
    }

    return;
}

void handle_error(int status)
{
    if(status == 8) {
        printf("Device timeout.\n");
    }
    else {
        printf("Error: status = %d\n",status);
    }
}

/*  exit(-1);*/
}

```


SCZANA.C - Program to acquire data from an Impedance Analyzer

Program SCZANA.C uses the Hewlett-Packard 4192A LF Impedance Analyzer to take eddy current data from a probe. The program can be used with absolute or differential bobbin coils, with pancake coils, and with reflection coils. It will take data at any number of frequencies between 5 Hz and 13 MHz. It will read either the probe impedance directly or the voltage across the probe. The Modulynx motion controller is used to position the probe on the sample. Both the impedance analyzer and the motion controller are controlled by the Capital Equipment PC-488 board.

Some variables of interest in the program are described below.

data1	Array containing the averaged first component of the data
data2	Array containing the averaged second component of the data
freq	Array containing the frequencies at which data will be taken
nft	The total number of frequencies used. There is no limitation on this number.
nrds	The number of readings to take and average together at each probe position
vout	Array to hold the driving voltages at each frequency
zana	HPIB address of the impedance analyzer

A complete listing of program SCZANA.C is given below.

```
/* Program SCZANA.C  
4 August 1994
```

```
This program will use the Modulynx motion controller  
to move a probe over a sample and will take readings  
from the HP Impedance Analyzer at each step.
```

```
The first component of the data (usually the resistive  
component of the impedance or the magnitude of a voltage  
in dB) is stored in the variable data1, and the second  
component of the data (usually the reactive component  
of the impedance or the phase of a voltage) is stored  
in the variable data2.
```

```
The probe positions are read from a data file created by  
the program dsposit.c.
```

```
The program codes for the impedance analyzer are listed  
on page 3-69 of its manual. To use the impedance analyzer  
to measure the magnitude and phase of a signal relative  
to some reference signal set the value of the variable  
zana_setup to "A5B2H1V1T3F0". To use the impedance analyzer  
to measure the resistance and reactance of a probe, set  
the value of the variable zana_setup to "A2B3H1V1C2T3F0".
```

```
Variables which must be set by the user are listed below.
```

nft	The total number of frequencies at which data will be taken
nrds	The number of readings that will be taken and averaged together at each frequency

```

    freq    Array to hold the frequencies to be used

    vout    Array to hold the driving voltages for
            each frequency

*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <dos.h>
#include <string.h>
#include <math.h>
#include "ieee-c.h"

void zanardg(float *data1, float *data2, float *freq, float *vout,
    int zana, int nft, int nrdgs);
void move(float *x);
void shutdown(void);
void handle_error(int);
void setup_motors(int motor_address, char *axis_label);

int motor_address = 6;
char motor_off_string[] = " XE YE ZE";
int naxes;
int motor_on[3] = {0,0,0};
char axis_label[80];

void main(void)
{
    FILE *str1;
    FILE *position_file;
    int controller_address = 21;
    int level = 0;
    int zana = 3;
    char zana_setup[] = "A5B2H1V1T3F0";
    int nft = 3;
    int nrdgs = 2;
    float *freq;
    float *vout;
    float *data1, *data2;
    int status;
    int i, j;
    float x[3];

    /* Allocate memory for frequency array */

    freq = calloc((size_t)nft, sizeof(float));
    if(freq == NULL) {
        printf("Error allocating memory for frequency array.\n");
        exit(-1);
    }

    /* Allocate memory for driving voltage array */

    vout = calloc((size_t)nft, sizeof(float));
    if(vout == NULL) {
        printf("Error allocating memory for voltage array.\n");
        exit(-1);
    }

    /* Allocate memory for data arrays */

    data1 = calloc((size_t)nft, sizeof(float));
    data2 = calloc((size_t)nft, sizeof(float));

    if((data1 == NULL) || (data2 == NULL)) {
        printf("Error allocating memory for data arrays.\n");
    }

```

```

        exit(-1);
    }

/* Specify frequencies to be used */
#ifdef 0
    for(i=0;i<nft;i++) {
        freq[i] = 20. * (i+1);
    }
#endif

    freq[0] = 200.;
    freq[1] = 100.;
    freq[2] = 50.;

/* Specify driving voltages */

    for(i=0;i<nft;i++) {
        vout[i] = 0.5;
    }

/* Initialize the HP-IB system */

    initialize(controller_address,level);

    str1 = fopen("ddata.dat","w");
    if(str1 == NULL) {
        printf("Error open output data file.\n");
        exit(-1);
    }

    position_file = fopen("dsposit.dat","r");
    if(position_file == NULL) {
        printf("Position file not found.\n");
        exit(-1);
    }

    fscanf(position_file,"%d",&naxes);
    fscanf(position_file,"%s",axis_label);

/* Turn on the motors that will be used */

    setup_motors(motor_address,axis_label);

/* Set up the impedance analyzer */

    send(zana,zana_setup,&status);

    for(;;) {

/* Read the next position from the file and move the probe */

        for(i=0;i<naxes;i++) {
            if(fscanf(position_file,"%f",&(x[i])) == EOF) {
                shutdown();
            }
        }
        move(x);

/* Take the data */

        zanardg(data1,data2,freq,vout,zana,nft,nrdgs);

/* Write the results */

        for(i=0;i<nft;i++) {
            printf("%f %f ",data1[i],data2[i]);
            fprintf(str1,"%f %f ", data1[i],data2[i]);

```

```

    }

    for(i=0;i<naxes;i++) {
        printf("%f ",x[i]);
        fprintf(str1,"%f ",x[i]);
    }

    printf("\n");
    fprintf(str1,"\n");
}

return;
}

void zanardg(float *data1, float *data2, float *freq, float *vout,
int zana, int nft, int nrdgs)
{
    char data[80];
    char command[80];
    char ex[] = "EX";
    unsigned maxlen = 79;
    unsigned *len;
    int status;
    float d1,d2;
    char d1str[20],d2str[20];
    int i,j;

    for(i=0;i<nft;i++) {

        data1[i] = 0.;
        data2[i] = 0.;

        sprintf(command," OL%5.3fENFR%9.3fEN",vout[i],freq[i]);
        send(zana,command,&status);

        for(j=0;j<nrdgs;j++) {
            send(zana,ex,&status);
            enter(data,maxlen,len,zana,&status);
            strncpy(d1str,data+4,11);
            d1str[11] = '\0';
            strncpy(d2str,data+20,11);
            d2str[11] = '\0';
            d1 = atof(d1str);
            d2 = atof(d2str);
            data1[i] += d1;
            data2[i] += d2;
        }

        data1[i] /= nrdgs;
        data2[i] /= nrdgs;
    }

    return;
}

void shutdown(void)
{
    FILE *str1;
    float x[3];
    int status;
    int i;

    /* Send motors home */

    for(i=0;i<naxes;i++) {
        x[i] = 0.;
    }

```

```

    }

    move(x);

    sleep(5);

/* Turn all motors off */

    send(motor_address,motor_off_string,&status);
    if(status) {
        handle_error(status);
    }

    exit(0);
}

void move(float *x)
{
/*
 * This subroutine sends the commands necessary to move the
 * axes to the positions specified in inches in the array x.
 * After the movement is completed, the positions are checked
 * and corrected if necessary.
 */
    static long iprev_pos[3] = {0,0,0}; /* Previous position in motor steps */
    long isteps[3]; /* Distance to move in motor steps */
    int status;
    int i;
    long n;
    unsigned wait_time;
    char tmp_string[80];
    char move_string[80];
    char check_pos_string[] = "<CFA>";
    char semicolon_string[] = ";";
    float step_size[3];

/* The variable step_size is the number of inches per step
 * for axes in which the motion is translation along a line
 * and is the number of degrees per step for axes in which the
 * motion is a rotation. Normally the z axis is used for
 * rotation. */

    for(i=0;i<naxes;i++) {
        if(axis_label[i] == 'Z') {
/* 40 steps per degree */
            step_size[i] = 0.025;
        }
        else {
/* 4 steps per mil */
            step_size[i] = 0.00025;
        }
    }

/* Calculate the number of motor steps each axis must take
 * to reach the desired position. */

    for(i=0;i<naxes;i++) {
        n = (long)((x[i] + 0.5*step_size[i])/step_size[i]);
        isteps[i] = n - iprev_pos[i];
        iprev_pos[i] = n;
    }

/* Generate the command string for the motion */

    for(i=naxes-1;i>=0;i--) {
        sprintf(tmp_string,"%c",axis_label[i]);
        strcat(tmp_string,"M");
    }

```

```

        if(i == (naxes-1)) {
            strcpy(move_string,tmp_string);
        }
        else {
            strcat(move_string,tmp_string);
        }
        sprintf(tmp_string,"%ldG ",isteps[i]);
        strcat(move_string,tmp_string);
    }

/* Send the motion command */

    send(motor_address,move_string,&status);
    if(status) {
        handle_error(status);
    }

/* Pause for the greater of 1 second and 1 second per 1000
   motor counts to give the motors time to complete their motion.
   The variable wait_time is the number of milliseconds the program
   will pause. */

    wait_time = 1000;
    for(i=0;i<naxes;i++) {
        if(wait_time < labs(isteps[i])) {
            wait_time = labs(isteps[i]);
        }
    }
    delay(wait_time);

/* Compare pulse count with encoder count and correct position
   if they disagree. */

    send(motor_address,check_pos_string,&status);
    if(status) {
        handle_error(status);
    }

    send(motor_address,semicolon_string,&status);
    if(status) {
        handle_error(status);
    }

    return;
}

void handle_error(int status)
{
    if(status == 8) {
        printf("Device timeout.\n");
    }
    else {
        printf("Error: status = %d\n",status);
    }
}

/* exit(-1);*/
}

void setup_motors(int motor_address,char *axis_label)
{
    char motor_on_string[80];
    char motor_off_string[] = " XE YE ZE";
    char ymotor_set[] = " YA6500YB335YH2500";
    char zmotor_set[] = " ZA18000ZB500ZH3000";
    char tmp_string[80];
    int status;
    int i,j;

```

```

/* Turn off all motors */

send(motor_address,ymotor_set,&status);
send(motor_address,zmotor_set,&status);
send(motor_address,motor_off_string,&status);
if(status) {
    handle_error(status);
}
sleep(2);

i = 0;
while(axis_label[i]) {
    if(isspace(axis_label[i])) {
        for(j=i+1;j<strlen(axis_label);j++) {
            axis_label[j-1]=axis_label[j];
        }
        i++;
    }
}

if(strlen(axis_label) != naxes) {
    printf("Variable naxes conflicts with axis_label.\n");
    exit(-1);
}

for(i=0;i<strlen(axis_label);i++) {
    if((axis_label[i] == 'X') || (axis_label[i] == 'x')) {
        motor_on[0] = 1;
    }
    else if((axis_label[i] == 'Y') || (axis_label[i] == 'y')) {
        motor_on[1] = 1;
    }
    else if((axis_label[i] == 'Z') || (axis_label[i] == 'z')) {
        motor_on[2] = 1;
    }
    else {
        printf("Unrecognized character %c in axis_label.\n",axis_label[i]);
        exit(-1);
    }
}

/* Generate string to turn motors on */

for(i=0;i<naxes;i++) {
    sprintf(tmp_string," %c",axis_label[i]);
    if(i==0) {
        strcpy(motor_on_string,tmp_string);
    }
    else {
        strcat(motor_on_string,tmp_string);
    }
}

/* Turn on the motors that will be used */

send(motor_address,motor_on_string,&status);
if(status) {
    handle_error(status);
}
sleep(5);

return;
}

```

SCAN30.C and MOVE30.C- Programs to position and acquire data from a MIZ-30

Programs SCAN30.C and MOVE30.C act together with the Zetec data acquisition program to take data from the Zetec MIZ-30 eddy current instrument. Program MOVE30.C is run on a PC and controls the position of the probe during the data acquisition. Program SCAN30.C is run on the Hewlett-Packard computer where it uses Eddynet routines to request and receive data from the Zetec acquisition program. Program MOVE30.C first reads a data file written by program DSPOSIT.C to find out which motors will be used and it turns on these motors. Then the program reads the first position in the file, moves to that position, and instructs the program SCAN30.C to prompt the user to do a hardware null in the Zetec acquisition program. After program SCAN30.C signals that this has been done, program MOVE30.C moves to the next probe position and when this is complete, it tells program SCAN30.C to take request data from the Zetec acquisition program. When the data have been received, SCAN30 instructs MOVE30 to move the probe to the next position. This continues until readings have been taken at all positions in the data file.

Program SCAN30.C is based on a sample program written by Richard Warlick of Zetec.

A complete listing of program SCAN30.C is given below.

```
/*
  Program scan30.c
  Version 22 March 1994

  This is a program which operates in conjunction with a
  probe-positioning program running on a PC and the Zetec
  acquire30d program running on the HP to get data from
  the MIZ-30.

  At the time this program is started, an acquire30d
  program with the application name "rdw" must already
  be running.
*/

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <habitat.h> /* habitat.h is part of the eddynet
                      programming package */
#include "cmd_valP.h"

void handle_send_slices_command(struct parsed_cmd *);
void handle_send_data_binary(HabitatMessagePacket *);
void process_command(HabitatMessagePacket *);
void process_binary(HabitatMessagePacket *);
void process_message(void);
void exit_program(void);

int nslices = 5;
int nchannels = 4;
int nbytes;
short *data;
double *avpdata;

void main(int argc, char *argv[])
```



```

{
    HabitatTransmittal My_Transmittal;
    FILE *str1, *str2, *str3, *str4;
    char string[80];
    int i,j;
    int naxes;
    int stat;
    float axispos[3];
    char tmpstr[80];

    nbytes = nslices * nchannels * 4;

    avgdata = calloc(nchannels*2,sizeof(double));
    data = calloc(nslices*nchannels*2,sizeof(short));

    str2 = fopen("ddata.dat","w");
    if(str2 == NULL) {
        printf("Unable to open output data file.\n");
        exit(-1);
    }

    /* First we need to register this process so that we can send messages */

    strcpy(Registration.ProcessName,"CASS9"); /* Name in habitat */
    strcpy(Registration.ApplicationName,"rdw");
    Registration.HowToInterrupt = DO_NOTHING;
    Registration.MessageQueueSize = 200000;
    if(mail_system_connect(Registration) == ERROR) {
        fprintf(stdout,"Error\n");
        exit(ERROR);
    }

    /* Set the appropriate info in the My_Transmittal struct */

    strcpy(My_Transmittal.TargetProcessName,"Acquire");
    strcpy(My_Transmittal.TargetApplicationName,Registration.ApplicationName);
    My_Transmittal.MessageTypeFlag = MSG_COMMAND;
    sprintf(string,"send_slices -slices %d",nslices);
    My_Transmittal.Message = (MailMsg *)string;
    My_Transmittal.MessageSize = strlen(string) + 1;

    for(;;) {

        /* Wait here for the file ghp.tmp to be written. */

        do {
            str1 = fopen("/users/jrp/eddyne/ghp.tmp","r");
            str3 = fopen("/users/jrp/eddyne/shp.tmp","r");
            str4 = fopen("/users/jrp/eddyne/nhp.tmp","r");
        } while((str1 == NULL) && (str3 == NULL) && (str4 == NULL));

        if(str3 != NULL) {

            /* This program should exit */
            /* Wait for a little while for the PC to finish up */

            sleep(5);

            /* Delete the file shp.tmp */

            fclose(str3);
            stat = remove("/users/jrp/eddyne/shp.tmp");

            if(stat) {
                printf("Error deleting file shp.tmp\n");
            }
            exit_program();
        }
    }
}

```

```

}

if(str4 != NULL) {
/* Read naxes from file posthp.dat */

str1 = fopen("posthp.dat","r");

if(str1 == NULL) {
printf("Unable to open file posthp.dat\n");
exit_program();
}

fscanf(str1,"%d",&naxes);
if((naxes < 1) || (naxes > 3)) {
printf("Error: naxes = %d\n",naxes);
exit_program();
}

fclose(str1);

/* Prompt user for hardware null */

printf("Do a hardware null. Press return when finished.\n");
i = getchar();

/* Delete the file nhp.tmp */

fclose(str4);
stat = remove("/users/jrp/eddynet/nhp.tmp");

if(stat) {
printf("Error deleting file nhp.tmp\n");
exit_program();
}

/* Create gpc.tmp */

str1 = fopen("/users/jrp/eddynet/gpc.tmp","w");
if(str1 == NULL) {
printf("Error opening file gpc.tmp\n");
exit_program();
}
fprintf(str1,"OK\n");

fclose(str1);

continue;
}

/* Read positions */

fclose(str1);

i = 0;
do {
i++;
str1 = fopen("/users/jrp/eddynet/posthp.dat","r");
if(str1 == NULL) {
printf("Failure to open position file on pass %d\n",i);
sleep(1);
}
} while((i < 10) && (str1 == NULL));

if(str1 == NULL) {
printf("Error opening position file.\n");
exit_program();
}
}

```

```

    for(i=0;i<naxes;i++) {
        fscanf(str1,"%f",&axispos[i]);
    }

    fclose(str1);

    stat = remove("/users/jrp/edynet/posthp.dat");
    if(stat) {
        printf("Error deleting position file.\n");
        exit_program();
    }

/* Now mail the message asking for data */

    MailMessage(&My_Transmittal);

/* Now wait for the data to be mailed back. I'll wait 10 seconds
   though it shouldn't take that long. */

    for(i=0;i<10;i++) {
        if(MailSystem.Queue->QueueHeader.NumberOfMessages > 0) {
            /* We have received a message */
            process_message();
            break;
        }
        else {
            sleep(1);
        }
    }

/* Average data */

    for(i=0;i<nchannels*2;i++) {
        avgdata[i] = 0.;
    }

    for(i=0;i<nslices;i++) {
        for(j=0;j<nchannels*2;j++) {
            avgdata[j] += data[i*nchannels*2 + j];
        }
    }

    for(i=0;i<nchannels*2;i++) {
        avgdata[i] /= nslices;
    }

/* Write data to file. */

    for(i=0;i<nchannels*2;i++) {
/*        printf("%d ",(int)avgdata[i]);*/
        fprintf(str2,"%d ",(int)avgdata[i]);
    }

    for(i=0;i<naxes;i++) {
/*        printf("%f ", axispos[i]);*/
        fprintf(str2,"%f ",axispos[i]);
    }

/*    printf("\n");*/
    fprintf(str2,"\n");

/* Delete the file ghp.tmp */

    stat = remove("/users/jrp/edynet/ghp.tmp");

    if(stat) {
        printf("Error deleting file ghp.tmp\n");
        exit_program();
    }

```

```

/* Create gpc.tmp */

    str1 = fopen("/users/jrp/eddyne/gpc.tmp","w");
    if(str1 == NULL) {
        printf("Error opening file gpc.tmp\n");
        exit_program();
    }

    fprintf(str1,"OK\n");

    fclose(str1);
}

void exit_program(void)
{
    /* Now disconnect this program from the mail messaging system
       before we leave the program. It's necessary to do this
       before a program quits executing so that we clean up the
       appropriate memory segments. */

    printf("Disconnecting from mail system.\n");
    mail_system_break();

    exit(0);
}

void handle_send_slices_command(struct parsed_cmd *the_parsed_cmd)
{
    struct opt_data *option; /* points at a command */
    char *arg_pointer; /* points at the argument to an opt */

    /* This message can contain two possible options.
       They are as follows:

       1) -not_acquiring_data -> This means the miz30 acquire program
                                   could not send the data because it
                                   is not in a data acquire mode; it's
                                   probably in the review mode.

       2) -couldnt_get_data -> This means the miz30 acquire program
                                is in the data acquire mode but for
                                some reason it was unable to mail the
                                data. This could happen if you were
                                asking for more than two seconds
                                worth of data.*/

    /* Look for the -not_acquiring_data option. */

    option = NULL;
    option = get_option(the_parsed_cmd,CV_NAME,0,"-not_acquiring_data");
    if(option != NULL) {

        /* The -not_acquiring_data option is present */

        printf("Error: System is not acquiring data.\n");
        exit_program();
    }

    /* Look for the -couldnt_get_data option. */

    option = NULL;
    option = get_option(the_parsed_cmd,CV_NAME,0,"-couldnt_get_data");
    if(option != NULL) {

        /* The -couldnt_get_data option is present */

        printf("Error: System is acquiring data but cannot send it back.\n");
    }
}

```

```

        exit_program();
    }
}

void handle_send_data_binary(HabitatMessagePacket *Packet)
{
    int i;

    /* Now copy the data into my local array. We know that the
       message sent from the acquire program is stored in
       Packet->Message. We also know the overall length of
       the message; this value is stored in Packet->MessageSize. */

    if(Packet->MessageSize != nbytes) {
        printf("An unexpected number of bytes were transmitted.\n");
        printf("  transmitted = %d\n", (int)Packet->MessageSize);
        printf("  expected = %d\n", nbytes);
        exit_program();
    }

    memcpy(data, (char *)&Packet->Message, Packet->MessageSize);
    /*
    for(i=0; i<8; i++) {
        printf("%d ", (int)data[i]);
    }
    printf("\n");
    */
    return;
}

void process_command(HabitatMessagePacket *Packet)
{
    struct parsed_cmd *the_parsed_cmd; /* parsed Packet->Message field */
    char string[128];

    printf("pc message = %s\n", &Packet->Message);
    the_parsed_cmd = validate_cmd(MSG_COMMAND, &Packet->Message);

    if(the_parsed_cmd != NULL) {
        if(the_parsed_cmd->valid == CV_VALID) {
            switch(the_parsed_cmd->command_id) {
                case SEND_SLICES:
                    /* We received a send_slices message from the miz30 acquire
                       program; now we need to interpret it. */
                    handle_send_slices_command(the_parsed_cmd);
                    break;
            }
        }
        else {
            printf("Invalid command: %s\n", the_parsed_cmd->cmd_line);
            printf("Return value of validate_cmd = ");
            switch(the_parsed_cmd->valid) {
                case CV_CONTEXT_INVALID:
                    printf("CV_CONTEXT_INVALID\n");
                    break;
                case CVE_NO_MEMORY:
                    printf("CVE_NO_MEMORY\n");
                    break;
                case CVE_INVALID:
                    printf("CVE_INVALID\n");
                    break;
                case CVE_EMPTY:
                    printf("CVE_EMPTY\n");
                    break;
                case CVE_CEILING:
                    printf("CVE_CEILING\n");
                    break;
                case CVE_OPT_2_BIG:

```

```

        printf("CVE_OPT_2_BIG\n");
        break;
    case CVE_NO_TYPE:
        printf("CVE_NO_TYPE\n");
        break;
    case CVE_NO_QUOTE:
        printf("CVE_NO_QUOTE\n");
        break;
    }
}
}
}

void process_binary(HabitatMessagePacket *Packet)
{
    switch(Packet->UserFlag) {
    case EDDY_DATA:
        /* We received a send_slices message from the miz30
        acquire program, now we need to interpret it. */
        handle_send_data_binary(Packet);
        break;
    }
}

void process_message(void)
{
    HabitatMessagePacket *Packet;

    while(MailSystem.Queue->QueueHeader.NumberOfMessages > 0) {
        Packet = ReadMessage();
        if(Packet != NULL) {
            switch(Packet->MessageTypeFlag) {
            case MSG_COMMAND:
                process_command(Packet);
                break;
            case MSG_BINARY:
                process_binary(Packet);
                break;
            }
            free(Packet);
        }
    }
}
}

```

A complete listing of program MOVE30.C is given below.

```

/*
 * Program MOVE30.C
 * Version 22 March 1994
 *
 * This program uses the Capital Equipment PC-488 board
 * and the Modulynx position controller to control the motion
 * of an XYZ scanner. It operates in conjunction with a program
 * running on the HP to take data from the MIZ-30.
 *
 * The program compiles correctly under Borland C++ for DOS.
 * It must be in a project with the file IEEE488.LIB.
 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <dos.h>
#include <string.h>
#include "ieee-c.h"

```

```

void move(float *x);
void shutdown(void);
void handle_error(int);

int motor_address = 6;
char motor_off_string[] = " XE YE ZE";
char ymotor_set[] = " YA6500YB335YH2500";
char zmotor_set[] = " ZA18000ZB500ZH3000";
int naxes;
int motor_on[3] = {0,0,0};
char axis_label[80];
char motor_on_string[80];

void main(void)
{
    FILE *position_file;
    FILE *str1;
    float x[3];
    int controller_address = 21;
    int motor_address = 6;
    int level = 0;
    int status;
    int i,j;
    char tmp_string[80];

/* Initialize GP-IB board */

    initialize(controller_address,level);

/* Turn off all motors */

    send(motor_address,ymotor_set,&status);
    send(motor_address,zmotor_set,&status);
    send(motor_address,motor_off_string,&status);
    if(status) {
        handle_error(status);
    }
    sleep(2);

/* Open position data file */

    position_file = fopen("dsposit.dat","r");
    if(position_file == NULL) {
        printf("Position file not found.\n");
        exit(-1);
    }

/* Read header information */

    fscanf(position_file,"%d",&naxes);
    fscanf(position_file,"%s",axis_label);

/* Remove white space from axis_label */

    i = 0;
    while (axis_label[i]) {
        if(isspace(axis_label[i])) {
            for(j=i+1;j<strlen(axis_label);j++) {
                axis_label[j-1] = axis_label[j];
            }
            i++;
        }
    }

    if(strlen(axis_label) != naxes) {
        printf("Variable naxes conflicts with axis_label.\n");
        printf(" naxes = %d\n", naxes);
    }
}

```

```

    printf(" axis_label = %s\n", axis_label);
    printf(" strlen(axis_label) = %d\n", strlen(axis_label));
    exit(-1);
}

for(i=0;i<strlen(axis_label);i++) {
    if((axis_label[i] == 'X') || (axis_label[i] == 'x')) {
        motor_on[0] = 1;
    }
    else if((axis_label[i] == 'Y') || (axis_label[i] == 'y')) {
        motor_on[1] = 1;
    }
    else if((axis_label[i] == 'Z') || (axis_label[i] == 'z')) {
        motor_on[2] = 1;
    }
    else {
        printf("Unrecognized character %c in axis_label.\n",axis_label[i]);
        exit(-1);
    }
}

/* Generate string to turn motors on */

for(i=0;i<naxes;i++) {
    sprintf(tmp_string," %cD",axis_label[i]);
    if(i==0) {
        strcpy(motor_on_string,tmp_string);
    }
    else {
        strcat(motor_on_string,tmp_string);
    }
}

/* Turn on the motors that will be used */

send(motor_address,motor_on_string,&status);
if(status) {
    handle_error(status);
}
sleep(5);

/* Read position for hardware null */

for(i=0;i<naxes;i++) {
    if(fscanf(position_file,"%f",&(x[i])) == EOF) {
        shutdown();
    }
    printf("%f ",x[i]);
}
printf("\n");
move(x);
printf("Waiting for hardware null.\n");

send(motor_address,motor_off_string,&status);
if(status) {
    handle_error(status);
}

/* Create file to tell the HP to pause for a hardware null */

str1 = fopen("p:\\eddynet\\posthp.dat","w");
if(str1 == NULL) {
    printf("Unable to open file on HP.\n");
    shutdown();
}
fprintf(str1,"%d \n\n",naxes);
fclose(str1);

```



```

str1 = fopen("p:\\eddynet\\nhp.tmp","w");
if(str1 == NULL) {
    printf("Unable to open file on HP.\n");
    shutdown();
}
fclose(str1);

/* Wait until the file gpc.tmp is written */

do {
    str1 = fopen("p:\\eddynet\\gpc.tmp","r");
} while(str1 == NULL);

fclose(str1);

/* Read positions from file and move axes */

for(;;) {

    str1 = fopen("p:\\eddynet\\posthp.dat","w");

    for(i=0;i<naxes;i++) {
        if(fscanf(position_file,"%f",&(x[i])) == EOF) {
            fclose(str1);
            shutdown();
        }
        printf("%f ",x[i]);
        fprintf(str1,"%f ",x[i]);
    }
    printf("\n");
    fprintf(str1,"\n");

/* turn motor on prior to movement */
send(motor_address,motor_on_string,&status);
if(status) {
    handle_error(status);
}
move(x);

send(motor_address,motor_off_string,&status);
if(status) {
    handle_error(status);
}

/* turn motor off after movement */

fclose(str1);

/* Delete gpc.tmp */

status = remove("p:\\eddynet\\gpc.tmp");
if(status) {
    printf("Error deleting file gpc.tmp\n");
    shutdown();
}

/* Write ghp.tmp */

str1 = fopen("p:\\eddynet\\ghp.tmp","w");
if(str1 == NULL) {
    printf("Unable to open file on HP.\n");
    shutdown();
}
fprintf(str1," 1 \n\n");
fclose(str1);

/* Wait for gpc.tmp to be written */

do {

```

```

        str1 = fopen("p:\\eddyne\\gpc.tmp","r");
    } while(str1 == NULL);

    fclose(str1);
}

void shutdown(void)
{
    FILE *str1;
    float x[3];
    int status;
    int i;

    /* Create file shp.tmp */

    str1 = fopen("p:\\eddyne\\shp.tmp","w");
    if(str1 == NULL) {
        printf("Error creating file shp.tmp\n");
    }
    fprintf(str1," 1 \n\n");
    fclose(str1);

    /* Delete file gpc.tmp */

    status = remove("p:\\eddyne\\gpc.tmp");
    if(status) {
        printf("Error deleting file gpc.tmp\n");
    }

    /* Send motors home */

    /* turn motor on prior to movement */
    send(motor_address,motor_on_string,&status);
    if(status) {
        handle_error(status);
    }

    for(i=0;i<naxes;i++) {
        x[i] = 0.;
    }
    move(x);

    /* Turn all motors off */

    send(motor_address,motor_off_string,&status);
    if(status) {
        handle_error(status);
    }

    exit(0);
}

void move(float *x)
{
    /*
     * This subroutine sends the commands necessary to move the
     * axes to the positions specified in inches in the array x.
     * After the movement is completed, the positions are checked
     * and corrected if necessary.
     */
    static long iprev_pos[3] = {0,0,0}; /* Previous position in
                                         motor steps */
    long isteps[3]; /* Distance to move in
                    motor steps */
    int status;

```

```

int i;
long n;
unsigned wait_time;
char tmp_string[80];
char move_string[80];
char check_pos_string[] = "<CFA>";
char semicolon_string[] = ";";
float step_size[3];

/* The variable step_size is the number of inches per step
for axes in which the motion is translation along a line
and is the number of degrees per step for axes in which the
motion is a rotation. Normally the z axis is used for
rotation. */

for(i=0;i<naxes;i++) {
    if(axis_label[i] == 'Z') {
/* 10 steps per degree */
        step_size[i] = 0.1;
    }
    else {
/* 4 steps per mil */
        step_size[i] = 0.00025;
    }
}

/* Calculate the number of motor steps each axis must take
to reach the desired position. */

for(i=0;i<naxes;i++) {
    n = (long)((x[i] + 0.5*step_size[i])/step_size[i]);
    isteps[i] = n - iprev_pos[i];
    iprev_pos[i] = n;
}

/* Generate the command string for the motion */

for(i=naxes-1;i>=0;i--) {
    sprintf(tmp_string,"%c",axis_label[i]);
    strcat(tmp_string,"M");
    if(i == (naxes-1)) {
        strcpy(move_string,tmp_string);
    }
    else {
        strcat(move_string,tmp_string);
    }
    sprintf(tmp_string,"%ldG ",isteps[i]);
    strcat(move_string,tmp_string);
}

/* Send the motion command */

send(motor_address,move_string,&status);
if(status) {
    handle_error(status);
}

/* Pause for the greater of 1 second and 1 second per 1000
motor counts to give the motors time to complete their motion.
The variable wait_time is the number of milliseconds the program
will pause. */

#if(0)
    wait_time = 10;
#elseif
/*    wait_time = 1000;
    for(i=0;i<naxes;i++) {
        if(wait_time < labs(isteps[i])) {

```

```

        wait_time = labs(isteps[i]);
    }
}
delay(wait_time);*/

/* Compare pulse count with encoder count and correct position
if they disagree. */

send(motor_address,check_pos_string,&status);
if(status) {
    handle_error(status);
}

send(motor_address,semicolon_string,&status);
if(status) {
    handle_error(status);
}

return;
}

void handle_error(int status)
{
    if(status == 8) {
        printf("Device timeout.\n");
    }
    else {
        printf("Error: status = %d\n",status);
    }
}

/* exit(-1);*/
}

```


INTERNAL DISTRIBUTION

1.	W. R. Corwin	19.	C. E. Pugh
2.	L. D. Chitwood	20.	W. A. Simpson
3.	D. F. Craig	21-22.	M&C Records Office
4.	S. A. David	23.	ORNL Patent Section
5-9.	C. V. Dodd	24.	Central Research Library
10.	H. W. Hayden, Jr.	25.	Document Reference Section
11-16.	D. J. McGuire	26-27.	Laboratory Records Department
17.	R. K. Nanstad	28.	Laboratory Records (RC)
18.	J. R. Pate	29.	Nuclear Safety Information Center

EXTERNAL DISTRIBUTION

- 30-31. NRC, OFFICE OF NUCLEAR REGULATORY RESEARCH, Washington,
DC 20555
- J. Craig
M. E. Mayfield
J. Muscara
C. Z. Serpan
- 32.-33. ELECTRIC POWER RESEARCH INSTITUTE, P.O. Box 217097,
Charlotte, NC 28221
- M. Elmo
K. Kryzwosz
34. BATTELLE PACIFIC NORTHWEST LABORATORIES, P.O. Box 999,
Richland, WA 99352
- B. Ferris
35. Steven D. Brown, 23 Greensburg Street, Delmont Borough, PA 15626
36. DOE, OAK RIDGE OPERATIONS OFFICE, Oak Ridge, TN 37831-8600
- Office of Assistant Manager for Energy Research and Development
37. DOE, OFFICE OF SCIENTIFIC AND TECHNICAL INFORMATION,
P. O. Box 62, Oak Ridge, TN 37831
- 38-286. Given distribution as shown in category RF (NTIS-10)

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

1. REPORT NUMBER
(Assigned by NRC. Add Vol., Supp., Rev.,
and Addendum Numbers, if any.)

NUREG/CR-6163
ORNL/TM-13212

2. TITLE AND SUBTITLE

Computer Programs for the Acquisition and Analysis of Eddy-Current Array
Probe Data

3. DATE REPORT PUBLISHED

MONTH | YEAR

July | 1996

4. FIN OR GRANT NUMBER

B0417

5. AUTHOR(S)

J. R. Pate and C. V. Dodd

6. TYPE OF REPORT

7. PERIOD COVERED (Inclusive Dates)

8. PERFORMING ORGANIZATION - NAME AND ADDRESS (If NRC, provide Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address, if contractor, provide name and mailing address.)

Oak Ridge National Laboratory
Oak Ridge, TN 37831-6285

9. SPONSORING ORGANIZATION - NAME AND ADDRESS (If NRC, type "Same as above"; if contractor, provide NRC Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address.)

Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

10. SUPPLEMENTARY NOTES

J. Muscara, NRC Project Manager

11. ABSTRACT (200 words or less)

The objective of the Improved Eddy-Current ISI for Steam Generators Tubing program is to upgrade and validate eddy-current inspections, including probes, instrumentation, and data processing techniques for inservice inspection of new, used, and repaired steam generator tubes; to improve defect detection, classification and characterization as affected by diameter and thickness variations, denting, probe wobble, tube sheet, tube supports, copper and sludge deposits, even when defect types and other variables occur in combination; to transfer this advanced technology to NRC's mobile NDE laboratory and staff. This report documents computer programs that were developed for acquisition of eddy-current data from specially-designed 16-coil array probes. Complete code as well as instructions for use are provided.

12. KEY WORDS/DESCRIPTORS (List words or phrases that will assist researchers in locating the report.)

Array Probe
Computer Programs
Neural Networks
Steam Generators
Tubing Inspection

13. AVAILABILITY STATEMENT
Unlimited

14. SECURITY CLASSIFICATION

(This Page)

Unclassified

(This Report)

Unclassified

15. NUMBER OF PAGES

16. PRICE