

FFIU/727/161.4

Approved
Kjeller 19 May 1998



Vidar S. Andersen
Director of Research

VISUALIZATION OF SONAR PERFORMANCE

NILSEN Erik Hamran, TVEIT Elling, WEGGE Jon

FFI/RAPPORT-98/02542


FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
Postboks 25, 2007 Kjeller, Norge

FORSVARETS FORSKNING SINSTITUTT (FFI)
Norwegian Defence Research Establishment
 P O BOX 25
 N-2007 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
 (when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER FFI/RAPPORT-98/02542 1a) PROJECT REFERENCE FFIU/727/161.4	2) SECURITY CLASSIFICATION UNCLASSIFIED 2a) DECLASSIFICATION/DOWNGRADING SCHEDULE	3) NUMBER OF PAGES 55		
4) TITLE VISUALIZATION OF SONAR PERFORMANCE (VISUALISERING AV SONARYTELSE)				
5) NAMES OF AUTHOR(S) IN FULL (surname first) NILSEN Erik Hamran, TVEIT Elling, WEGGE Jon				
6) DISTRIBUTION STATEMENT Approved for public release. Distribution unlimited (Offentlig tilgjengelig)				
7) INDEXING TERMS IN ENGLISH: <table style="width: 100%; border: none;"> <tr> <td style="width: 50%; vertical-align: top;"> a) <u>Computer programs</u> b) <u>Sonar</u> c) <u>Simulation</u> d) <u>Diagrams</u> e) _____ </td> <td style="width: 50%; vertical-align: top;"> IN NORWEGIAN: a) <u>Dataprogram</u> b) <u>Sonar</u> c) <u>Simulering</u> d) <u>Diagrammer</u> e) _____ </td> </tr> </table>			a) <u>Computer programs</u> b) <u>Sonar</u> c) <u>Simulation</u> d) <u>Diagrams</u> e) _____	IN NORWEGIAN: a) <u>Dataprogram</u> b) <u>Sonar</u> c) <u>Simulering</u> d) <u>Diagrammer</u> e) _____
a) <u>Computer programs</u> b) <u>Sonar</u> c) <u>Simulation</u> d) <u>Diagrams</u> e) _____	IN NORWEGIAN: a) <u>Dataprogram</u> b) <u>Sonar</u> c) <u>Simulering</u> d) <u>Diagrammer</u> e) _____			
THESAURUS REFERENCE: NASA SP-7064 (Vol 1)				
8) ABSTRACT The document describes several methods for sonar performance visualization, and how prediction data can be used by an operator to adjust sonar parameters. A computer program has been implemented to create most of the plots and diagrams in this document. A users guide to the program and a detailed description of the code are included.				
9) DATE 19 May 1998	AUTHORIZED BY This page only  Vidar S. Andersen	POSITION Director of Research		

ISBN 82-464-0260-9

UNCLASSIFIED

CONTENTS

		Page
1	INTRODUCTION	5
2	VISUALIZATION METHODS	6
2.1	Contour plots	6
2.2	Isosurfaces	7
2.3	Volume visualization	8
2.4	Maximum speed	10
2.5	Optimal sensor parameter	11
3	GSMD USERS GUIDE	14
3.1	Installation and startup	14
3.2	Organization of data files	14
3.3	Parameter selection	15
3.4	Contour plots	17
3.5	3D visualization	18
4	CONCLUSION	20
	References	21
APPENDIX		
A	IMPLEMENTATION OF GSMD	22
A.1	Interactive Data Language (IDL)	22
A.2	Class structures in GSMD	22
B	SOURCE CODE SUMMARY	24
B.1	GSMServer	24
B.2	Sensor	28
B.3	Scalarfield	30
B.4	Towed array	31
B.5	Hull mounted sonar	32
B.6	Variable depth sonar	33
B.7	Dipping sonar	34
B.8	Main procedure GSMD	35
B.9	Main window	36
B.10	2D graphics window	38
B.11	3D graphics window	39
B.12	Trackball	43

B.13	Sonarmap	47
B.14	Object graphics contour plots	48
B.15	Plot routines	50
B.16	Data loading routines	51
B.17	Utilities	52

VISUALIZATION OF SONAR PERFORMANCE

1 INTRODUCTION

In modern sonar systems, computer models will be used to predict the sonar performance. Ideally, the model should be updated very frequently, and the environmental parameters should be changed to reflect the new position and time. The prediction data should be presented for the sonar operator, in such a way that decisions can be made to adjust sonar parameters, and eventually the speed and course of the vessel. The intention of this report is to give a few examples of how the presentation can be done, and to illustrate the type of information that are available from sonar prediction.

The document is divided in two parts. The first part concentrates on methods and graphic algorithms for displaying sonar prediction data. The second part is a description of a computer application which has been developed as a part of the project. This experimental application has been used to create the plots and examples in this document. The application may also give some ideas on how support tools for sonar operators can be designed.

2 VISUALIZATION METHODS

In this chapter we will show some methods for visualization of sonar prediction data. The prediction data is generated by Generic Sonar Model from Naval Underwater Warfare Center (NUWC), USA. We have used signal excess as a measure for sonar performance. Figure 2.1 defines a color scale for signal excess values with corresponding probability of detection (when the probability of false alarm is 0.01 % and 0 dB corresponds to 50% probability of detection).

We will use the following sonars in the examples:

- HMS: Hull mounted sonar, 7 kHz, variable tilt, self noise depends on bearing.
- ATAS: Towed array, 2 kHz, variable sensor depth, noise from towing vessel depends on bearing.

All parameters in the simulations are independent of range, i.e. we assume constant depth in the whole area and no horizontal variation of the sound speed profile.

2.1 Contour plots

The signal excess is usually computed in cylindrical coordinates, i.e. as a function of range, bearing and target depth. The resulting three dimensional scalarfield may be visualized as cuts in different orientations. Figure 2.1 shows an example of a horizontal cut, drawn as a polar contour plot.

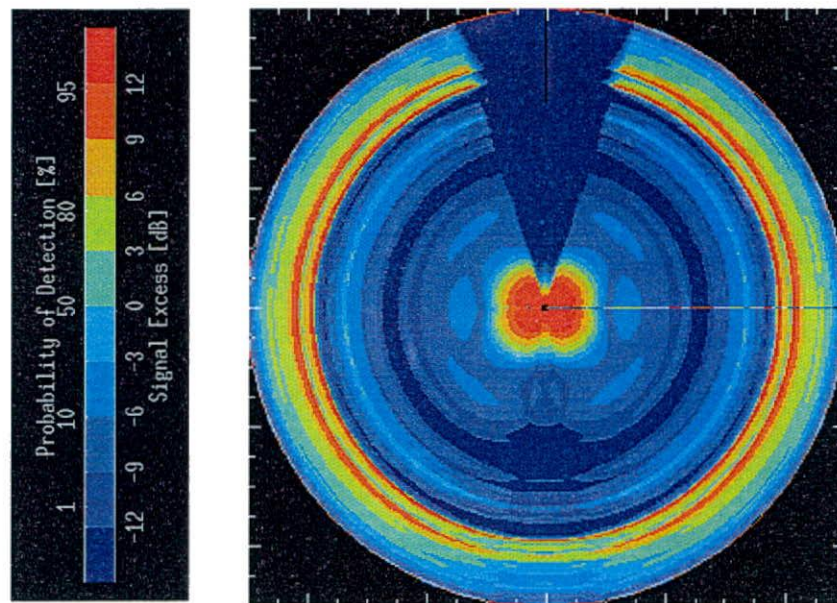


Figure 2.1 Signal excess for ATAS (Norwegian Sea, Winter, SS2) as a function of range and bearing. Target depth is 10 m, and the speed of own ship is 10 knots.

The towing ship is moving upward in the figure. The sonar performance is poor in this direction because of the noise from the towing ship. The scale in the plot is 50 km range, and there is a distinct convergence zone at about 40 km.

In figure 2.1 there is a line drawn for the bearing 90 degrees (to the right), and a circle of 50 km radius. We have plotted the vertical cuts along these lines, i. e. signal excess as a function of range and target depth, and signal excess as a function of bearing and target depth. The result is shown in figure 2.2.

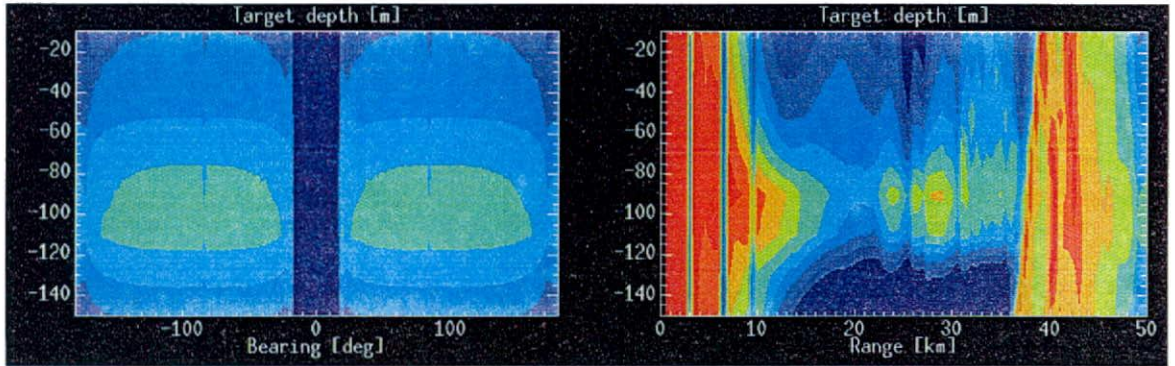


Figure 2.2 Signal excess as a function of bearing and target depth (to the left) at 50 km range. The other plot shows signal excess as a function of range and target depth at 90 degrees bearing. The other parameters are the same as in figure 2.1.

The examples shown here are very cheap to implement, because the computations are carried out in cylindrical coordinates. The main disadvantage is that we can see the sonar performance only in some parts of space. Anyway, the method might be useful when the signal excess is similar for all bearings, or when the approximate position of a target is known.

2.2 Isosurfaces

An alternative method to visualize three dimensional scalar data is to create isosurfaces. Isosurfaces are surfaces with constant signal excess value. There are several methods to generate isosurfaces, and many of them are based on the “marching cubes” algorithm. See (1) to get a brief introduction to this algorithm. We have used the IDL function “SHADE_VOLUME”.

Figure 2.3 shows an example of an isosurface representing 0 dB signal excess (the white surface). A sector is removed to reveal some internal structures. The isosurface has holes, which has been filled using signal excess contour plots.

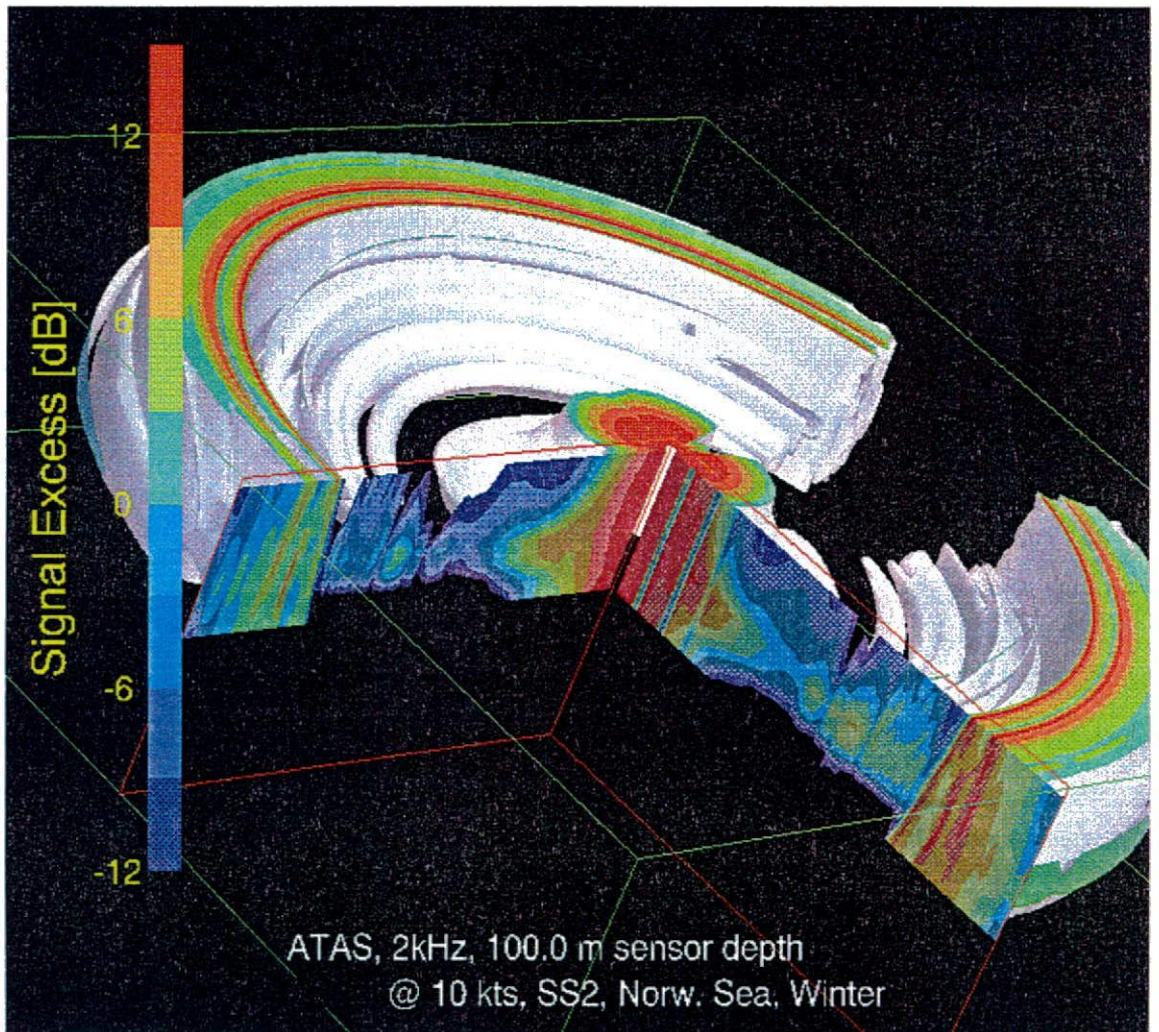


Figure 2.3 *Isosurface (0 dB) and cuts with contour plots for visualization of signal excess data.*

2.3 Volume visualization

Volume visualization can be used to show all parts of the three dimensional signal excess data. Figure 2.4 is an example of this method, where we have used the same data as in last section. First the data is converted to a regular grid. Then we use tables to convert signal excess values to red, green and blue color components, together with a transparency value. The final image is then created by drawing many overlapping partial transparent images. The transparency can be adjusted to form compact or blurred objects. (See (1) for more information about volume visualization algorithms).

Volume visualization is expensive in both memory and processing. On conventional computers creating an image might take several minutes, but using specialized graphics hardware we can achieve a rate of several frames per second.

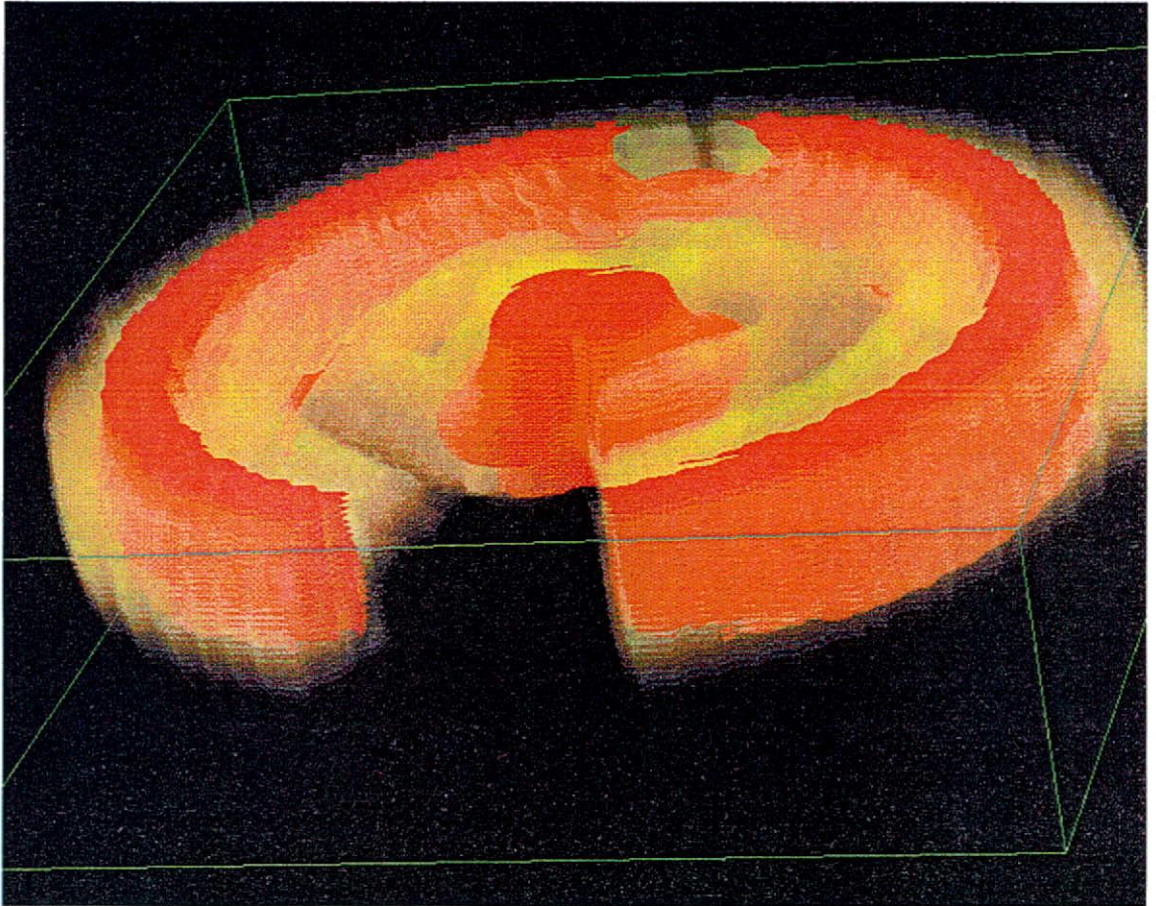


Figure 2.4 Volume visualization of signal excess for ATAS (Norwegian Sea, Winter, SS2). The sensor depth is 100 m, and the towing ship is moving towards us at speed of 10 knots. Red solid areas have signal excess above 6 dB. Yellow, transparent areas have signal excess between -6 and 6 dB. The resolution is $64 \times 64 \times 8$.

The main disadvantage for volume visualization is that it is difficult to create good conversion tables for color and transparency (conversion tables which gives clear and reasonable sharp images). But when we find good tables, volume visualization will compress a lot of information into one image.

2.4 Maximum speed

In most situations the speed of own vessel will influence the performance of the sonar, because noise from machinery and propulsion may reach the sensor. In addition flow noise directly on the sensor might be of importance. For a sonar operator it is important to have an idea of how the sonar performance is influenced by the speed. In figure 2.5 we have chosen a specific direction and target depth and plotted the signal excess as a function of range and own speed for a hull mounted sonar. Generally we have that the performance is decreasing when the speed is increasing.

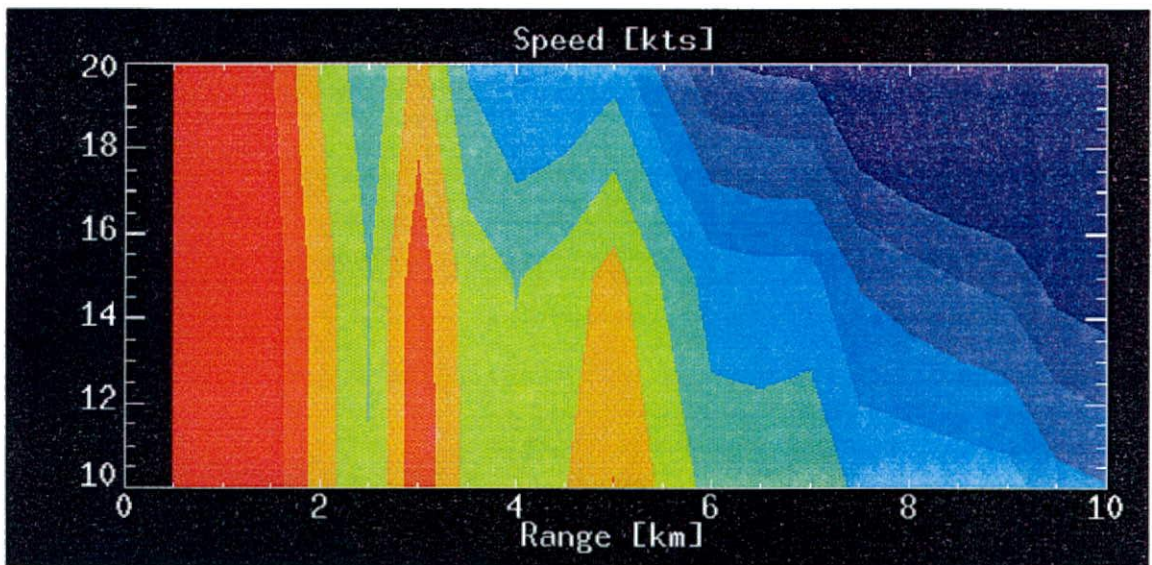


Figure 2.5 Signal excess as a function of range and own speed (HMS, 0 degrees bearing, 40 m target depth).

In figure 2.6 we have tried to show the speed dependence for all bearings, ranges and target depths. We have simulated the sonar performance for three different speeds – 10, 15 and 20 knots. Then for each range, bearing and target depth we have found the maximal allowed speed, provided that the signal excess is greater than 0. The result is the solid object shown in figure 2.6, where green means that we have reasonably good performance at 20 knots, in the yellow area the maximum speed is 15 knots, and in the red area 10 knots.

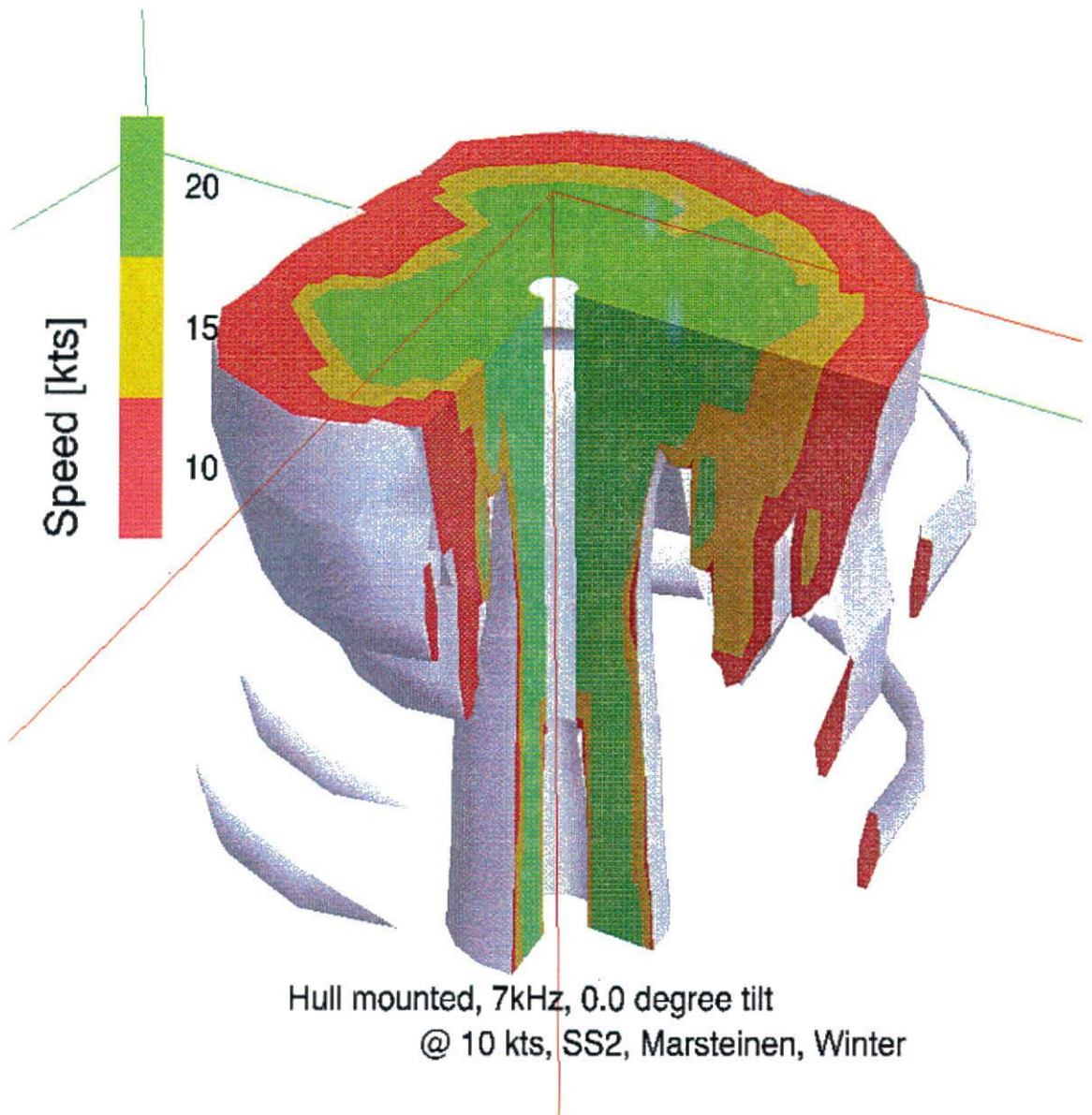


Figure 2.6 Maximum allowed speed, when we require probability of detection greater than 50%.

The method gives best results when the typical behaviour is that the signal excess is decreasing when the speed is increasing. This is not obvious for all sensors and target positions.

2.5 Optimal sensor parameter

Usually it is possible to change the behaviour of a sonar by changing different parameters (e.g. sensor depth or tilt). In this section we will give some ideas on how to create tools which can help sonar operators to choose optimal parameter values. We will use simulations of a towed array, where the sensor depth has been set to 50, 100 and 140 m.

Figure 2.7 shows an example where we plotted the signal excess as a function of range and sensor depth for three different target depths (50, 90 and 130 m). In this case the optimal choice of sensor depth seems to be about the same as the expected target depth.

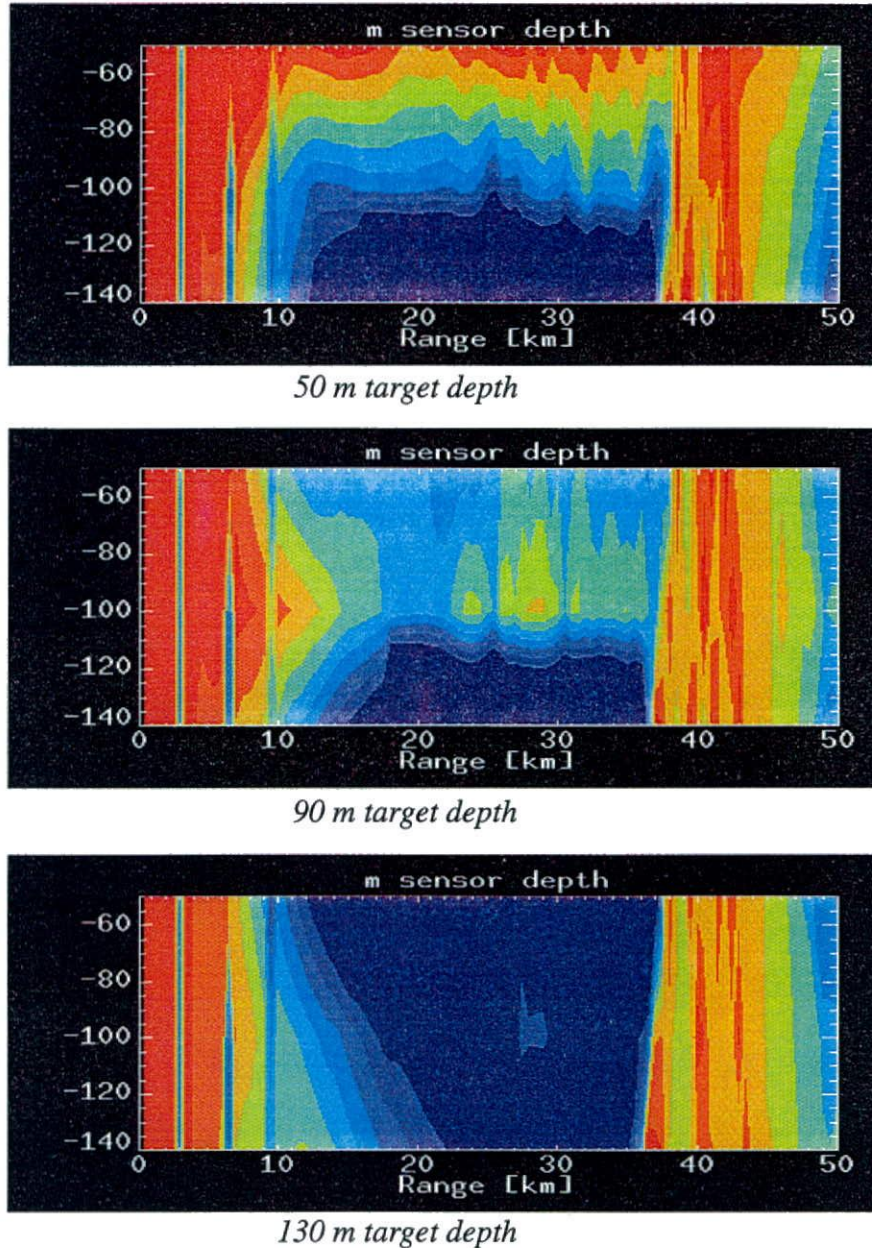


Figure 2.7 Signal excess as a function of range and sensor depth for different target depths (ATAS, winter, Norwegian Sea, SS2, bearing 90 degrees).

It is difficult to get a total overview of the situation using the plots in figure 2.7. We want to make a single plot where the dependence of bearing and all target depths are included. In figure 2.8 we have plotted the optimal sensor depth for each bearing, range and target depth. A natural way to define the optimal sensor depth is to find the sensor depth which gives the highest signal excess value for each point in space. We have modified this definition because it could lead to many unnecessary adjustments of the sensor depth. Instead

we define the current sensor depth as the optimal as long as signal excess is greater than 0. When the signal excess falls below 0 we choose the sensor depth which gives the highest signal excess greater than 0.

In figure 2.8 the current sensor depth is 100 m, and the yellow area shows the corresponding detection area. The red and green areas are not covered by the current sensor depth, but will be covered if the sensor depth is set to 50 or 140 m respectively.

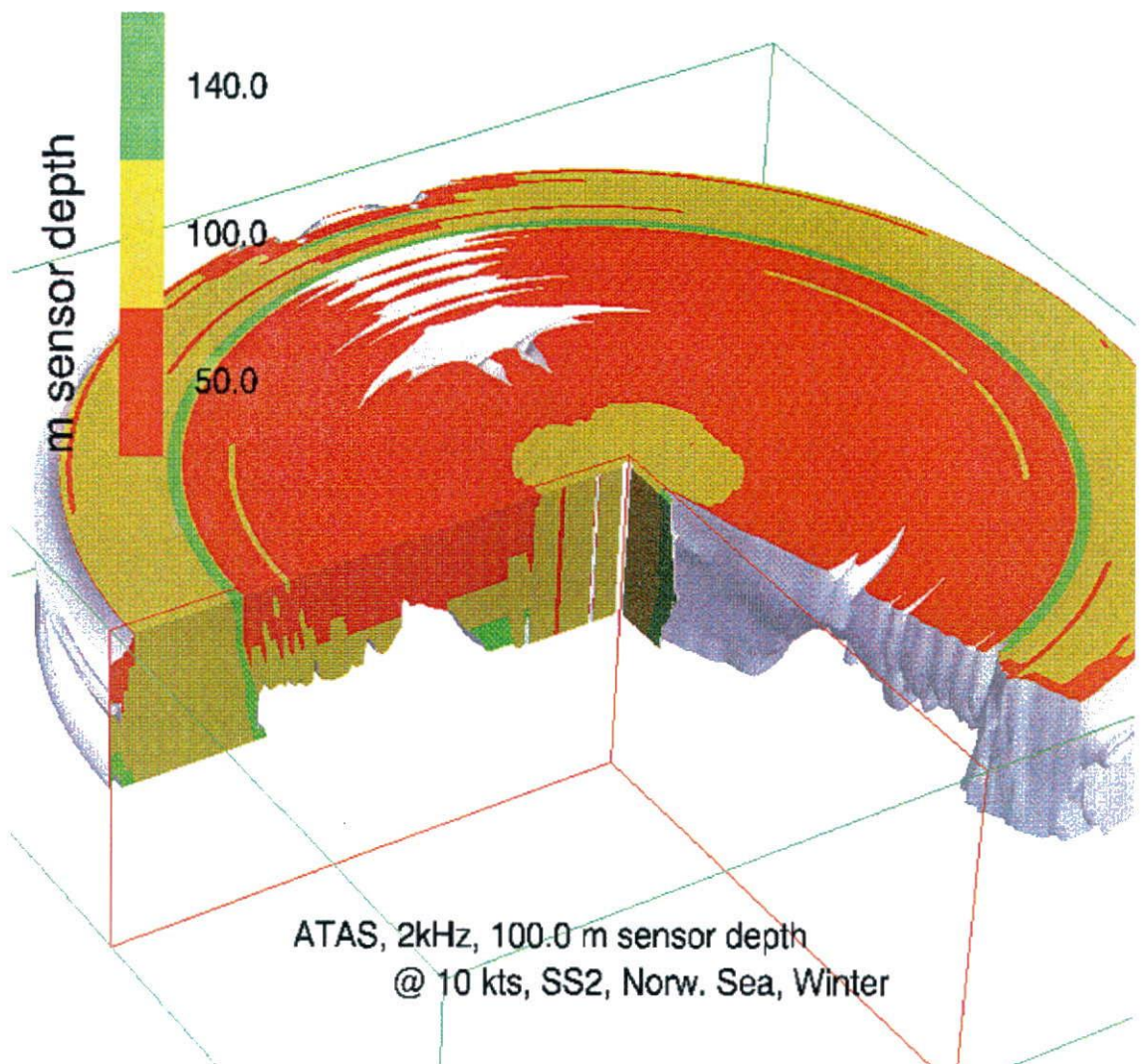


Figure 2.8 Optimal sensor depth for a towed array.

3 GSMD USERS GUIDE

GSMD is a visualization program for displaying signal excess data generated by “Generic Sonar Model” from NUWC. GSMD is specially designed to let the user browse through a large number of experiments, and to visualize results which depend on both bearing, range and target depth.

3.1 Installation and startup

First change to the directory where GSMD should be installed, and unpack the archive file using “tar xvf gsmd.tar”. The archive contains IDL (Interactive Data Language) source code, data directories and shell scripts. The resource file “gsmdrc” in “gsmd/etc/setup” should be edited so that all system variables point to correct directories. Run the setup script by entering “source gsmd/etc/setup/gsmc” (This line could also be included in the users “.tshrc” file). Note that “gsmdrc” is written for tcsh.

Now it should be possible to start IDL with the command “runidl”, and then call the IDL procedure “GSMD” to start the main program. Alternatively GSMD can be started directly from the shell by the command “rungsmd”.

The archive also include scripts and data which can be used together with the sonar simulation package GSM (Generic Sonar Model) to simulate sonar systems (the “gsmd/sim” directory).

3.2 Organization of data files

A collection of sonar simulation results is included in “gsmd.tar”. New data can be added, but the files has to be named and formatted in a special way, and the source code of GSMD has to be changed.

Each sonar simulation is identified by a standard set of parameters.

Parameter Name	Examples	Comments
self noise level	high, low	noise from own ship
location	MT, ND	Marsteinen, Norskehavet
season	W, S	Winter, Summer
sea state	2, 5	
speed	10, 15, 20	speed of own ship [knots]
sensor parameter 1 sensor parameter 2	50, 100, 140 / 0, 1, 2, 3	e.g. sensor depth/tilt or signal processing
range	0.5, 1.0, ..., 50.0	Distance to target [km]
bearing	0, 2, ...,180	relative bearing
target depth	30, 70, 100, 150	depth [m]

The first seven parameters is used to define the names of the datafiles, and each file contains a three dimensional signal excess dataset in cylindrical coordinates (range, bearing, target depth).

The format of a GSMD data file is described in the following table:

Type	Number of elements	Name	Description
int	1	nrange	Number of range values
int	1	ntheta	Number of bearing values
int	1	ndepth	Number of target depths
float	nrange	range	List of range values
float	ntheta	theta	List of bearing values
float	ndepth	depth	List of target depth values
float	nrange * ntheta * ndepth	data	3D data array (SE, PD or other values)

The file should be stored in XDR format (hardware independent binary format). The C program “xdrconv.c” in “gsmd/tools” directory can be used to convert ASCII data into XDR format.

The document (3) shows how the sonar simulation and the formatting of data have been done for a hull mounted sonar and a towed array.

When the user wants to add a new sensor to GSMD, a new class corresponding to the sensor should be created. The new class should inherit the “sensor” class. Two member functions, “GET_FILENAME” and “INIT” must be implemented. “INIT” initializes the object and “GET_FILENAME” creates a filename from the first seven parameters above.

In addition the main procedure defined in “gsmd.pro” should be edited and the new sensor object added to the list. See the source code “gsmd.pro”. Any of the sensors (“XAD”, “XSP” et.c.) can be used as template for new sensor classes.

3.3 Parameter selection

The main window in GSMD is shown in figure 3.1. The user can change all parameters using the pulldown buttons and sliders on the left. The plots on the right shows the corresponding sound speed profile and the signal excess as a polar contour plot.

From the top left the parameters are self noise level, ocean location, season and sea state. Below there are buttons to choose the sonar and its parameters (sensor depth, tilt or signal processing). There is also a button to change the speed of own ship.

When parameters are changed the plots are automatically updated. The polar contour plot shows the signal excess as a function of range and bearing, given a specific target depth.

The target depth may be adjusted using a slider. It is possible to specify a position (range and bearing) using the left button in the polar plot. This position will be used to create other plots, as we will see below.

On the menubar there is two pulldown menus. Use the "File" menu to quit the application. The "Window" menu can be used to open additional plot windows.

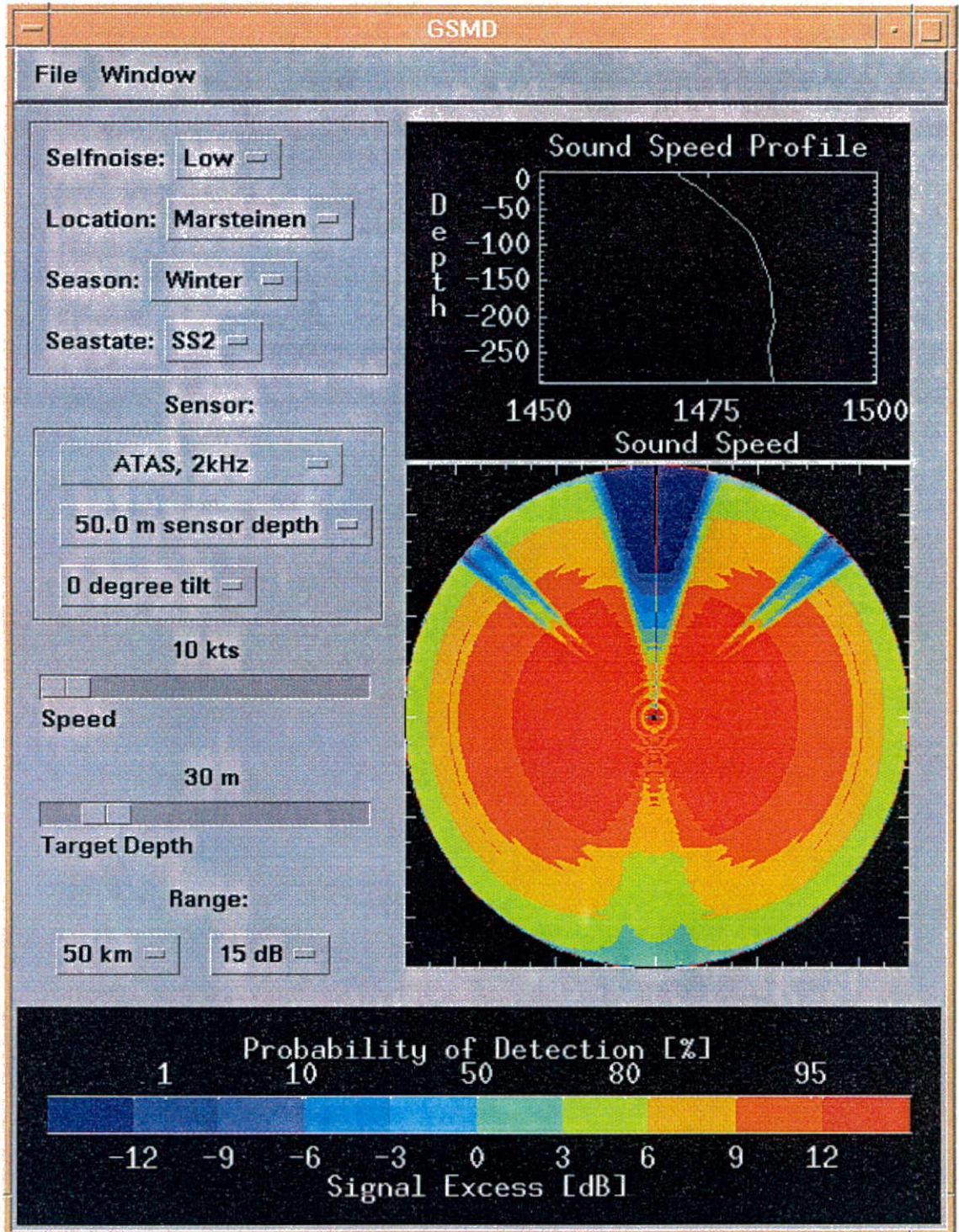


Figure 3.1 Main window of the GSMD application.

3.4 Contour plots

The “2D View” button opens an additional window for contour plots. Figure 3.2 shows an example.

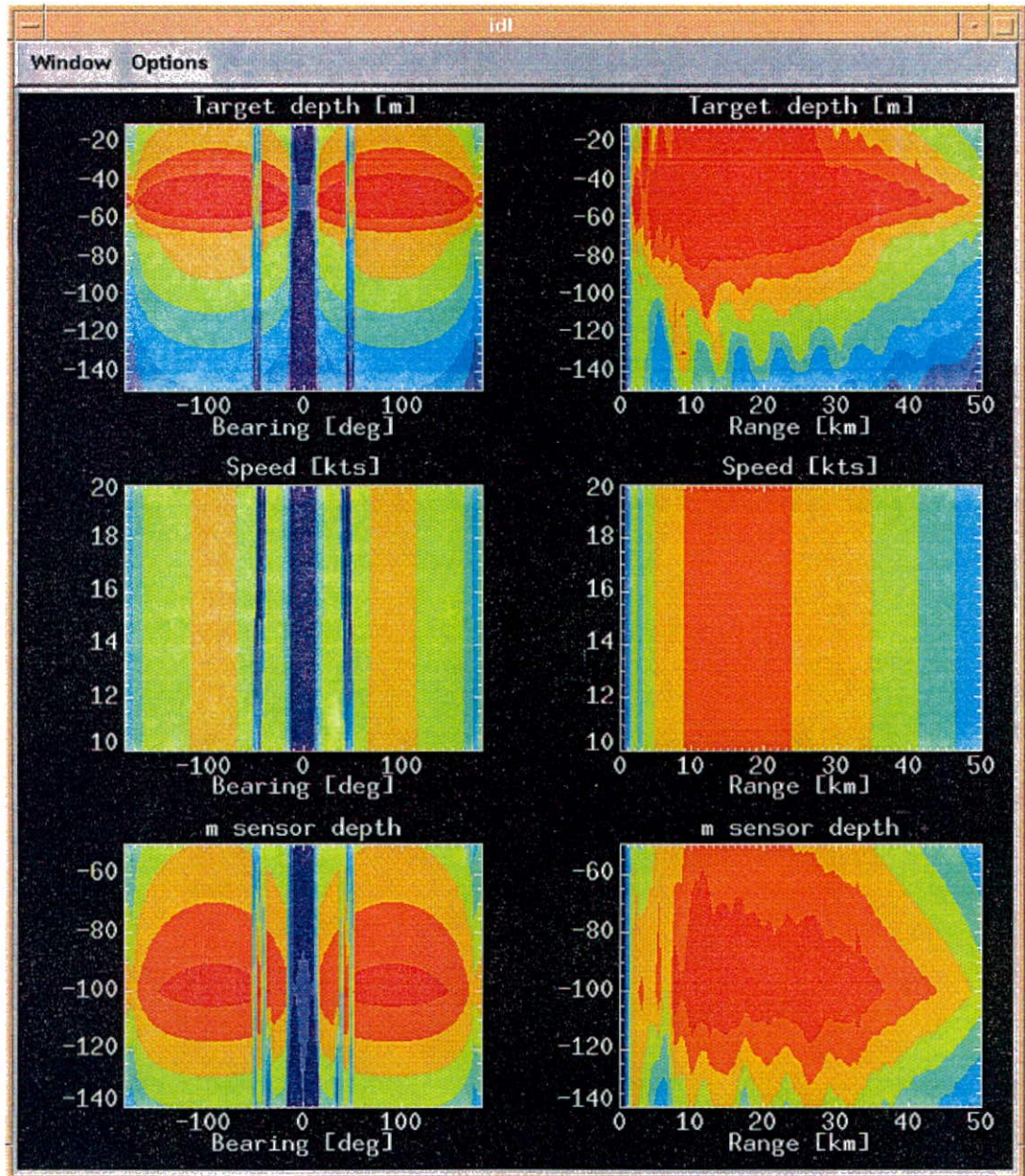


Figure 3.2 Window for 2D contour plotting (ATAS, Marsteinen, Winter, SS2, 10 kts speed, 50 m sensor depth, 35 km range, 90 degrees bearing, 90 m target depth).

The window may contain up to six different contour plots at the same time. The plots show signal excess data, and the colors are the same as in the main window. The left column has “bearing” on the x-axis and the right column always has “range” on the x-axis. The first row shows signal excess as a function of bearing/range and target depth, the second row signal excess as a function of bearing/range and speed, and the last row signal

excess as function of bearing/range and a sensor parameter (usually tilt or sensor depth). The type and number of plots may vary for different choices of sensor.

3.5 3D visualization

GSMD also supports 3D visualization of sonar prediction data. The submenu “3D view” of the main window will open the window shown in figure 3.3. This resizable window will display a green wireframed box. The box can be rotated using the left mouse button in the drawing area, the middle button will zoom the display and the right button will translate the box.

Several objects can be displayed and manipulated in the 3D window:

- Cuts with contour plots can be drawn correctly positioned in the 3D space. This makes it easy to see where the cuts belong geometrically. Cuts can be created horizontally and radially, and also as a cylinder (bearing vs. target depth). The position of the cuts is controlled by the cursors and sliders in the main window.
- Isosurfaces can be generated for different choices of signal excess levels, and visualized as white surfaces. A sector of the surface can be removed to expose internal details.
- Volume visualization can be used to display all details of the dataset.

Figure 3.3 shows an example of an isosurface where one half is removed to reveal the sonar performance for target depths in the forward and backward directions.

Options

- Plot Data: Three different types of data can be displayed – signal excess, maximum speed and optimal sensor parameter. The interpretation of the last two options is given in the previous chapter.
- Render Detail: The ‘fast’ option will hide most of the graphic objects to allow faster image update. Use ‘best’ to see the final result.
- Isosurface: The isosurface can be added or removed, and the signal excess level adjusted.
- Remove Sector: Gives possibility to remove a sector from the data, and the size of the sector can be adjusted.
- Volume Rendering: Is used to turn volume rendering on or off.

The window will remember its state when it is closed and then reopened.

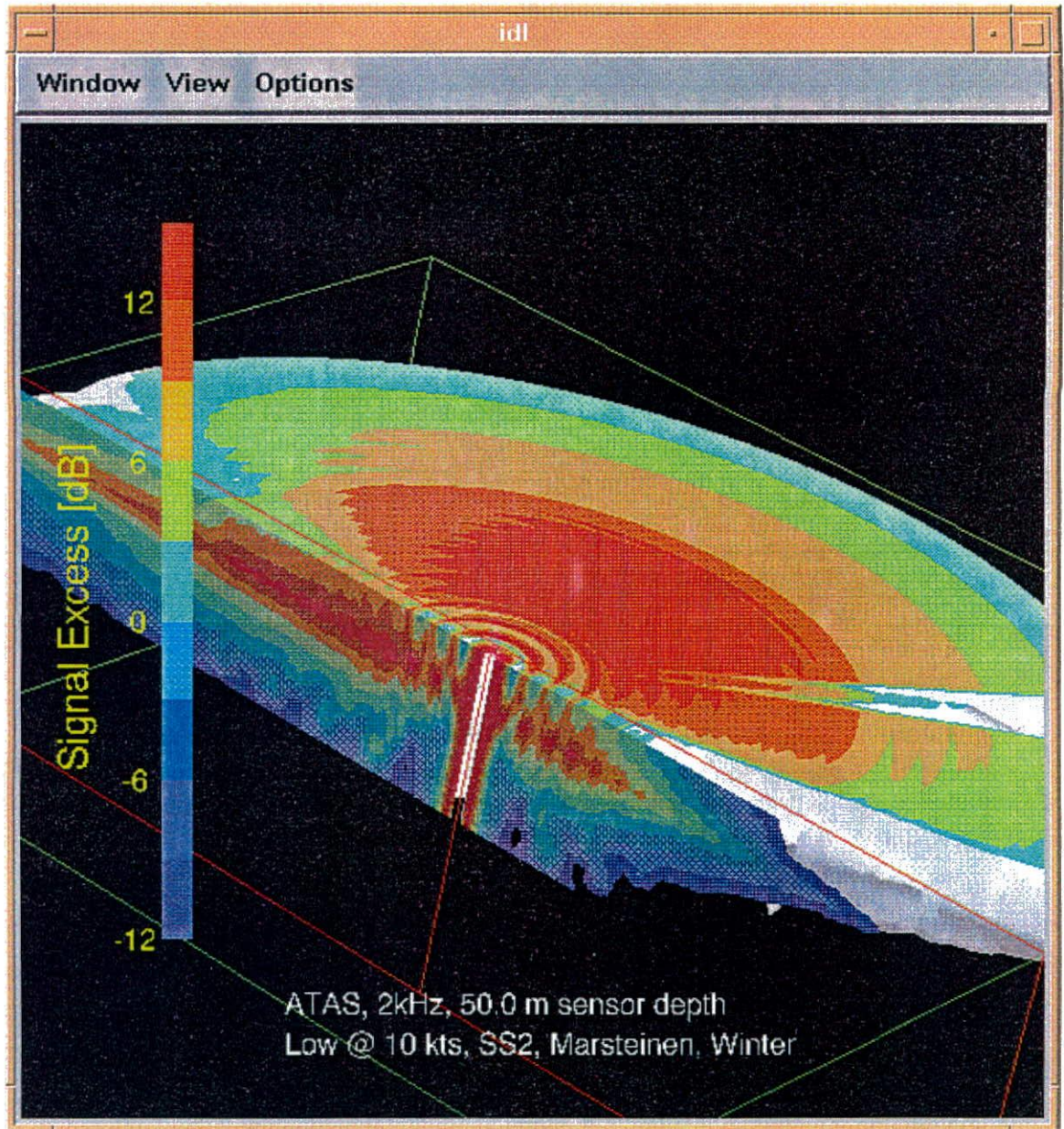


Figure 3.3 3D visualization of sonar prediction data.

4 CONCLUSION

We have shown some examples of how sonar prediction data can be visualized. One of the goals has been to give some ideas on how to create support tools for sonar operators. We have seen the importance of three dimensional visualization when the sonar performance varies for different bearings.

When the sonar performance is computed in Generic Sonar Model, we have to keep all parameters independent of the range (i.e. constant depth and sound speed profile for the whole area). The only parameters which varies for different bearings are the noise from the ship, and possibly the beampattern for the antenna. In real life environmental variables will also vary. A more accurate model would probably give much more variation for different bearings (especially in coastal areas), and 3D visualization would be even more important.

The application GSMD can be used to visualize sonar prediction data in real time, but the computation of the signal excess has to be done in advance. A challenge for the future is to be able to compute the sonar prediction data in real time.

Most of the visualization methods we have used can be combined with visualization of sonar echo and geographical data (like bottom depth, type et.c.). Hopefully some of the ideas in this report can be usefull in the design of Man–Machine Interfaces for new sonar systems.

References

- (1) Foley, van Dam, Feiner, Hughes: Computer Graphics. Principles and Practice, 2nd Edition, Addison Wesley 1990
- (2) IDL documentation: Objects and Object Graphics, IDL Version 5.0, March, 1997 Edition.
- (3) Erik Hamran Nilsen: Using the script language PERL for Generic Sonar Model simulations, FFI/NOTAT-98/02543

APPENDIX

A IMPLEMENTATION OF GSMD

A.1 Interactive Data Language (IDL)

GSMD is written entirely in IDL from Research Systems Inc. Version 5 of IDL provides tools for developing object oriented applications, and we have chosen to use objects for the implementation of GSMD. For the readers who are not familiar with object oriented design we will describe some common object oriented concepts:

Classes and objects

Objects are created as instances of a class, which is defined as an IDL structure and a collection of procedures and functions. The procedures and functions are often referred to as *methods*. The only way to access the data in an object is through its methods.

Inheritance

Inheritance lets the programmer use an existing class as a base for a new class. The new class will contain all data and methods from the base class, and the programmer can add new functionality through additional data and methods.

Some typical object oriented concepts like virtual functions, does not have any meaning in IDL version 5, and the polymorphism works differently from for example C++. Another drawback of object oriented design in IDL is that structures can not be redefined without restarting IDL. On the other hand, object oriented design is a great help for creating structured and readable code.

Further information about object oriented programming in IDL can be found in (2).

A.2 Class structures in GSMD

Figure A.1 shows a part of the class hierarchy in GSMD. Black arrows corresponds to inheritance and dotted lines tells us that a class contain one or more references to another class. In other words, classes XAD, XSP et.c. are inherited from the general "Sensor" class. "GSMServer" is the class which organizes the sensors and all relevant parameters. Therefore "GSMServer" contains references to "Sensor" objects. The "Scalarfield" class is a storage class for 3D signal excess data. Each "Sensor" contains one or more "Scalarfield" objects.

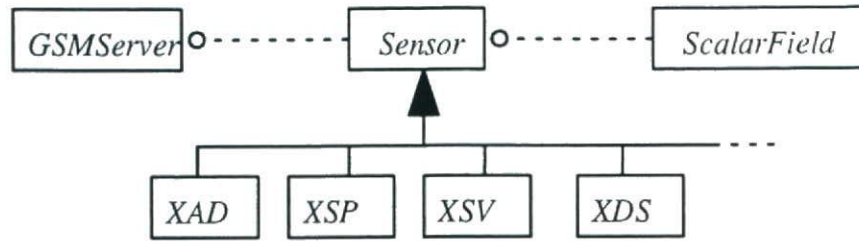


Figure A.1 Overview of some classes in GMSD

The user interface of GMSD is consisting of three classes, one for each window. In addition some extensions to “IDL object graphics” has been implemented, for example a trackball and filled contour plots.

B SOURCE CODE SUMMARY

The listing below contains class descriptions and lists of all methods of the classes in GSMD. The implementation of the procedures and functions is not included. When the name of a method is succeeded by paranthesis, the method is a function. Otherwise it is a procedure. Parameters and keywords can be destinguished by an equal sign at the end of all keywords.

B.1 GSMServer

 FILE: gsmserver__define.pro

OBJECT: GSMSERVER

PURPOSE:

The object is a part of an interface to visualize data from "Generic Sonar Model". GSMServer can hold several "sensor"-objects, and provides subroutines for extraction of signal excess data. A list of sensor object references should be supplied for in

The SE(signal excess) data depends on several parameters, which can be accessed through GET_PROPERTY/SET_PROPERTY. Each parameter is an index in the range [0, max], where max is defined individually for each "sensor" object and parameter.

Environmental parameters:

OCEAN, SEASON, SEASTATE, SHIP

Sensor parameters:

SENSOR_PAR1, SENSOR_PAR2 : Two parameters describing the sensor state, f.ex. sensor depth or tilt.

SPEED : Speed of own ship.

Geometric parameters:

BEARING, RANGE, TARGET_DEPTH : Specifies a target position of special interest.

The sensor objects should provide storage for SE data, and the following methods (with specified keywords):

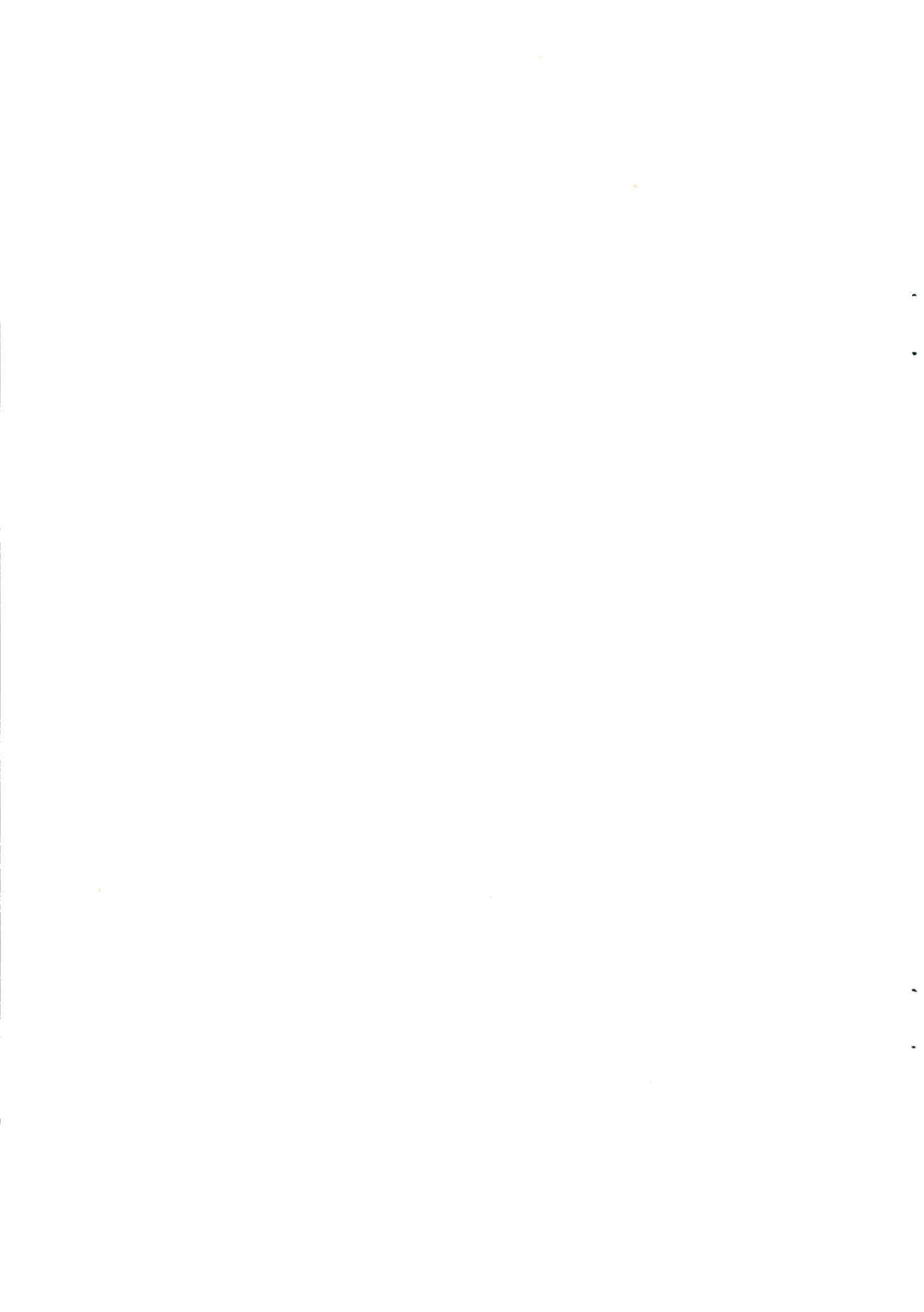
READ_DATA : GSMSERVER calls this procedure when the value of OCEAN, SEASON, SEASTATE or SHIP has been changed. The intention is to give the sensor object a chance to read (or compute) new data.

KEYWORDS : OCEAN, SEASON, SEASTATE, SHIP

GET_SE() : Returns a pointer to a SCALARFIELD object, containing SE data as a function of BEARING, RANGE and TARGET_DEPTH (cylindrical coordinates).

KEYWORDS : SPEED, SENSOR_PAR1, SENSOR_PAR2

GET_PROPERTY : Should provide lists of strings or scalars, describing the legal ranges for all the 10 parameters above.



```

KEYWORDS   : SHIP_NAMES, OCEAN_NAMES, SEASON_NAMES,
             SEASTATE_NAMES,
             SENSOR_PAR1_LIST, SENSOR_PAR2_LIST, SPEED_LIST,
             BEARING_LIST, RANGE_LIST, TARGET_DEPTH_LIST

```

END

SEE ALSO:

SENSOR, SCALARFIELD

END

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 13 Jan 1998

OBJECT DEFINITION:

```

struct = { GSMserver, $
           speed:      0, $ ; Indices describing the current
           bearing:   0, $ ; state of the sonar system
           range:     0, $ ; :
           target_depth: 0, $ ; :
           nsensor:   0, $ ; :
           sensor_par1: 0, $ ; :
           sensor_par2: 0, $ ; :
           ship:      0, $ ; :
           seastate:  0, $ ; :
           ocean:     0, $ ; :
           season:    0, $ ; :
           sensor: PTR_NEW(), $ ; Pointer to a list of sensor objects
           main_dir:  '' $ ; Location of data
        }

```

MEMBER PROCEDURES/FUNCTIONS:

```

READ_DATA      Is called whenever new values for
                SENSOR, OCEAN, SEASON, SEASTATE or SHIP are defined

GET_SSP        Find the current Sound Speed Profile
  depth
  speed

GET_RB         Get two dimensional matrix containing SE as a
                function of range and bearing.
  r
  theta
  data

GET_BS         Get SE as a function of bearing and own speed.
  bearing
  speed
  data

GET_RS         Get SE as a function of range and own speed
  range
  speed
  data

```

GET_BD	Get SE as a function of bearing and target depth
bearing	
depth	
data	
GET_RD	Get SE as a function of range and target depth
range	
depth	
data	
GET_BP1	Get SE as a function of bearing and the first sensor parameter
bearing	
sensor_list	
data	
GET_RP1	Get SE as a function of range and the first sensor parameter
range	
sensor_list	
data	
GET_BP2	Get SE as a function of bearing and the second sensor parameter
bearing	
sensor_list	
data	
GET_RP2	Get SE as a function of range and the second sensor parameter
range	
sensor_list	
data	
GET_TD	
sensor_depth	
target_depth	
data	
GET_SENSOR()	Get a pointer to a sensor object
SENSOR=	
GET_SE()	Returns a pointer to the current range/bearing/depth data
GET_BEST_SE()	
SPEED=	
GET_OPTIMAL_PAR()	Returns a scalarfield object, where each point contains the index of the optimal value of sensor parameter 1 (in other words the value of sensor par 1 which gives the highest PD).
LIMIT=	default=0.0

```

GET_MAXIMUM_SPEED() Returns a scalarfield object, where each point
                    contains the index of the maximum allowed speed,
                    when we require that SE > LIMIT.

                    LIMIT=                default=0.0

SET_PROPERTY        Set properties for object.
    DIRECTORY=
    NSENSOR=
    SHIP=
    OCEAN=
    SEASON=
    SEASTATE=
    SPEED=
    SENSOR_PAR1=
    SENSOR_PAR2=
    RANGE=
    BEARING=
    TARGET_DEPTH=

GET_PROPERTY        Get sensor properties
    SENSOR_NAMES=
    SHIP=
    OCEAN=
    SEASON=
    SEASTATE=
    SPEED=
    SENSOR_PAR1=
    SENSOR_PAR2=
    BEARING=
    RANGE=
    TARGET_DEPTH=
    NSENSOR=

CLEANUP

INIT()              Initialization of the GSMserver object. The SENSOR_LIST
                    keyword must be supplied.

                    SENSOR_LIST=         List of sensor object references
                    DIRECTORY=          Main data directory

```


B.2 Sensor

```
-----
FILE:   sensor_define.pro
OBJECT: SENSOR
-----
```

PURPOSE:

SENSOR is a superclass for implementation of sensor objects. This version assumes that SE data is generated by GSM, and the result depends on the environmental variables SHIP, OCEAN, SEASON, SEASTATE, plus SPEED and two additional parameters SENSOR_PAR1 and SENSOR_PAR2. The object holds data for all values of SPEED, SENSOR_PAR1, SENSOR_PAR2 simultaneously in memory, and new data is reloaded when other parameters are changed. The SE field is stored in cylindrical coordinates (bearing, range, target_depth).

SEE ALSO:

GSMSEVER, SCALARFIELD, XAD, XSP, XVS, XDS

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1998

OBJECT DEFINITION:

```
struct = ( sensor,  $
           name:    ', $
           ship_names: PTR_NEW(), $
           ocean_names: PTR_NEW(), $
           season_names: PTR_NEW(), $
           seastate_names: PTR_NEW(), $
           speed_list: PTR_NEW(), $
           sensor_par1_list: PTR_NEW(), $
           sensor_par2_list: PTR_NEW(), $
           par1_title:  ', $
           par2_title:  ', $
           target_depth_list: PTR_NEW(), $
           range_list: PTR_NEW(), $
           bearing_list: PTR_NEW(), $
           file_type:   0B, $
           file_sym:   0B, $
           data:       PTR_NEW(), $
           data_dir:   ' ' $
           )
```

MEMBER PROCEDURES/FUNCTIONS:

REMOVE_DATA removes all loaded data, if necessary

READ_DATA reads data

OCEAN=

SEASON=

SEASTATE=

SHIP=

```

GET_SE()           returns SE data (scalarfield object)
  SPEED=
  SENSOR_PAR1=
  SENSOR_PAR2=

NAME()

CLEANUP

INIT()
  NAME=             Sensor name
  DATA_DIR=       Location of data files
  SHIP_NAMES=
  OCEAN_NAMES=
  SEASON_NAMES=
  SEASTATE_NAMES=
  SPEED_LIST=
  SENSOR_PAR1_LIST=
  SENSOR_PAR2_LIST=
  PAR1_TITLE=      Description of par1 and
  PAR2_TITLE=      par2, e.g. "tilt", "sensor depth"
  TARGET_DEPTH_LIST=
  RANGE_LIST=
  BEARING_LIST=
  FILE_TYPE=       0B: ASCII file, 1B: XDR binary file
  FILE_SYM=        Data is symmetric about theta=0

GET_PROPERTY
  SHIP_NAMES=
  OCEAN_NAMES=
  SEASON_NAMES=
  SEASTATE_NAMES=
  SPEED_LIST=
  TARGET_DEPTH_LIST=
  SENSOR_PAR1_LIST=
  SENSOR_PAR2_LIST=
  PAR1_TITLE=
  PAR2_TITLE=
  RANGE_LIST=
  BEARING_LIST=

```

B.3 Scalarfield

```
-----
FILE:  scalarfield_define.pro
OBJECT: SCALARFIELD
-----
```

PURPOSE:

The class provides storage of scalars given on a regular grid in 3D space.

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1998

OBJECT DEFINITION:

```
struct = { scalarfield, $
           x: PTR_NEW(), $
           y: PTR_NEW(), $
           z: PTR_NEW(), $
           v: PTR_NEW(), $
           polar: 0, $
           irregular: 0 $
         }
```

MEMBER PROCEDURES/FUNCTIONS:

```
REMOVE_DATA          Removes data from object

read_gsm             Read ASCII data generated by "Generic Sonar Model"
  filenames

read_xdr             Read XDR binary file, format specified in
  independent procedure read_xdr
  filename
  SYMMETRIC=

clear               Create a zero field

GET_ZSLICE          Extract a slice perpendicular to the z axis.
  ZPOS=
  DATA=            OUT: slice
  X=
  Y=

GET_SUBARR()        Extract a part of the field.  If all three keywords
                    are defined a scalar is returned, if two keywords are
                    defined a vector is returned.
  ZPOS=
  XPOS=
  YPOS=

GET_PROPERTY
  XAXIS=
  YAXIS=
  ZAXIS=
  DATA=
```



```

CLEANUP

INIT()
  X=          x axis
  Y=          y axis
  Z=          z axis
  DATA=     3D array
  POLAR=     1B: polar xy, 0B: regular
  IRREGULAR=

```

B.4 Towed array

```

-----
FILE:   xad__define.pro
OBJECT: XAD
-----

```

PURPOSE:

XAD is a specialized SENSOR object, and provides format and filenames for the SE data created by "Generic Sonar Model". The sensor used is a 2 kHz ATAS.

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1998

OBJECT DEFINITION:

```

struct = { XAD, $
           INHERITS sensor $
         }

```

MEMBER PROCEDURES/FUNCTIONS:

```

GET_FILENAME()      Returns filename of SE data file
TARGET_DEPTH=
SENSOR_PAR1=
SENSOR_PAR2=
SHIP=
SPEED=
OCEAN=
SEASON=
SEASTATE=

```

```

INIT()
  DATA_DIR=      Path to data directory
  FILE_TYPE=     0 for ASCII, 1 for XDR.
  BEARING_LIST=  default is 0,1..359
  TARGET_DEPTH_LIST= default is 0,1

```

B.5 Hull mounted sonar

```
-----
FILE:   xsp__define.pro
OBJECT: XSP
-----
```

PURPOSE:

XSP is a 7 kHz Hull mounted sonar. This object provides format and filenames for SE data created by "Generic Sonar Model".

Name of data files:

```
...dir/XSP[noise]_F7000_S[speed]_T[tilt]_[ocean][season]_SS[seastate].SE
```

where the brackets should be substituted with each element in the arrays:

```
[noise]   = '', 'Q'
[speed]   = '01', '02', '03'   (actually 10, 15 and 20 kts)
[tilt]    = '0.0', '3.5', '7.0', '10.5', '14.0'
[ocean]   = 'MT', 'ND'
[season]  = 'W', 'S'
[seastate] = '2', '5'
(A total of 240 files.)
```

Each file should be a binary data file of the format specified in "read_xdr", with

```
range      = 0.5, 1.0, ..., 50.0
bearing    = 0, 20, ..., 340
target_depth = 20, 40, ..., 160
```

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1998

OBJECT DEFINITION:

```
struct = { XSP, $
           INHERITS sensor $
         }
```

MEMBER PROCEDURES/FUNCTIONS:

GET_FILENAME() Returns filename of SE data file

TARGET_DEPTH=

SENSOR_PAR1=

SENSOR_PAR2=

SHIP=

SPEED=

OCEAN=

SEASON=

SEASTATE=

_EXTRA=

INIT()

DATA_DIR=

B.6 Variable depth sonar

FILE: xsv__define.pro

OBJECT: XSV

PURPOSE:

XSV is a specialized SENSOR object. 12 kHz Variable depth sonar, and provides format and filenames for the SE data created by "Generic Sonar Model".

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1998

OBJECT DEFINITION:

```
struct = { XSV, $
          INHERITS sensor $
        }
```

MEMBER PROCEDURES/FUNCTIONS:

```
GET_FILENAME()
  TARGET_DEPTH=
  SENSOR_PAR1=
  SENSOR_PAR2=
  SHIP=
  SPEED=
  OCEAN=
  SEASON=
  SEASTATE=
```

INIT()

```
  DATA_DIR=
```


B.7 Dipping sonar

```
-----  
FILE:   xds__define.pro  
OBJECT: XDS  
-----
```

PURPOSE:

XDS is a specialized SENSOR object, Dipping sonar,
and provides format and filenames for the SE data created
by "Generic Sonar Model".

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1998

OBJECT DEFINITION:

```
struct = ( XDS, $  
          INHERITS sensor $  
          )
```

MEMBER PROCEDURES/FUNCTIONS:

```
GET_FILENAME()  
  TARGET_DEPTH=  
  SENSOR_PAR1=  
  SENSOR_PAR2=  
  SHIP=  
  SPEED=  
  OCEAN=  
  SEASON=  
  SEASTATE=
```

```
INIT()  
  DATA_DIR=
```

B.8 Main procedure GSMD

FILE: gsmd.pro

PROCEDURES/FUNCTIONS:

control_event
sEvent

control_cleanup
wTopBase

gsmd Main procedure for visulization of "Generic Sonar Model" results. If the keyword BLOCK is set equal to 1, the procedure will not return until the user has pressed the quit button. If BLOCK is equal to 0, the procedure will return soon after the user interface has been set up.

The keyword DATA_DIR is optional, and if it is not specified, GSMD will use the environmental variable \$GSMD_DATA instead.

DATA_DIR= Location of data files
BLOCK= 1: block

B.9 Main window

```
-----
FILE:   paramwin__define.pro
OBJECT: PARAMWIN
-----
```

PURPOSE:

Main window of the GSMD application. Lets the user choose all simulated parameters interactively. The window contains a plot of the Sound Speed Profile and a polar plot of SE as a function of range and bearing. Using the left mouse button in the polar plot, the user can specify a bearing and range of special interest.

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1998

OBJECT DEFINITION:

```
struct = ( PARAMWIN, $
           wSensorList: 0L, $
           wShip: 0L, $
           wOcean: 0L, $
           wSeason: 0L, $
           wSeaState: 0L, $
           wSensorPar1: 0L, $
           wSensorPar2: 0L, $
           wProcessing: 0L, $
           wSpeed: 0L, $
           wSpeedLab: 0L, $
           wDraw: 0L, $
           saveimage: PTR_NEW(), $
           wScale: 0L, $
           wSSP: 0L, $
           wTarget: 0L, $
           wTargetLab: 0L, $
           StartColor: 0, $
           NColors: 0, $
           NLevels: 0, $
           v_range: [0.0, 0.0], $
           r_range: 0.0, $
           btndown: 0b, $
           bearing: 0.0, $
           range: 0.0, $
           bearing_index: 0, $
           range_index: 0, $
           win3d: OBJ_NEW(), $
           plot: OBJ_NEW(), $
           SMserver: OBJ_NEW() $
         )
```

MEMBER PROCEDURES/FUNCTIONS:

```
PROCESS_MENU    Handles events created by the menubar and its submenus.
event
```


DRAW_EVENT	Handles motion and button events from the polar SE plot.
event	
PROCESS_EVENT	Process button press and slider events.
event	
CLEANUP	
DRAW_TARGETPOS	Draws a cursor indicating the current range and bearing.
XPOS=	
YPOS=	
ERASE=	
DRAW_RB	Draw SE as a function of Range and Bearing (polar plot).
DRAW_SCALE	Draws a color scale with SE [dB] and PD (probability of detection) axes.
DRAW_SSP	Draw the current Sound Speed Profile
GET_INFOTEXT()	Find an array of strings, describing the current state.
UPDATE	Update buttons and sliders for the current sensor.
GET_PROPERTY	
C_LEVELS=	Contour levels
C_COLORS=	Contour colors
V_RANGE=	Contour range
MAX_RANGE=	Max distance
INIT()	
SMserver	Pointer to GSMSEVER object
TITLE=	Window title
TLB=	
MANAGE=	
GROUP_LEADER=	
NON-MEMBER PROCEDURES/FUNCTIONS:	
paramwin_Menu_event	
event	
paramwin_draw_event	
event	
paramwin_event	
event	
paramwin_cleanup	
tlb	

B.10 2D graphics window

```
-----
FILE:   smwindow_define.pro
OBJECT: SMWINDOW
-----
```

PURPOSE:

SMWINDOW is a resizable graphic window, which uses 2D graphic to display data given by the GSMSEVER object. SE data is plotted as functions of bearing, range, target depth, own speed, and different sensor parameters.

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1996

OBJECT DEFINITION:

```

struct = ( SMWINDOW, $
;           Zmin: 0.0, $
;           Zmax: 0.0, $
;           NLevels: 0, $
;           MaxRange: 0.0, $
;           Range: 0.0, $
;           Bearing:0.0, $
;           StartColor: 0, $
;           NColors: 100, $
;           label1: 0L, $
;           label2: 0L, $
;           graph1: 0L, $
;           graph2: 0L, $
           drawBplot: 0B, $
           drawRplot: 0B, $
           tlb: 0L, $
           visible: 0B, $
           wDraw: 0L, $
           window: 0L, $
           mainwin: OBJ_NEW(), $
           SMserver: OBJ_NEW() $
)

```

MEMBER PROCEDURES/FUNCTIONS:

```
PROCESS_EVENT      Process menu events.
    event
```

```
SHOW
    flag           0B: hide window, 1B: show window
```

```
BEARING_SHIFT      Shift bearing axis 180 degrees, to make 0 degrees
                    the center of the plots.
    bearing
    data
```

```

DRAW                Draw graphics.

INIT()
  SMserver           pointer to GSMSEVER
  mainwin
  XSIZE=
  YSIZE=
  TITLE=
  STARTCOLOR=
  NCOLORS=
  MAP=
  GROUP_LEADER=

```

NON-MEMBER PROCEDURES/FUNCTIONS:

```

SMWindow_Cleanup   This is the clean up routine called when the TLB dies.
  tlb

SMWindow_Expose_Events  Handle window expose events here.
  event

SMWindow_Event
  event

```

B.11 3D graphics window

```

-----
FILE:   win3d_define.pro
OBJECT: WIN3D
-----

```

PURPOSE:

WIN3D defines a resizable IDL object graphics window, and methods to visualize 3D signal excess data in cylindrical coordinates. The class provides its own pulldown menus to let the user choose different parameters.

The following types of 3D objects are used:

```

Frame:           A box surrounding the volume. At the moment fixed to
                  [-50, 50] km in both x and y direction. The depth
                  interval is set to [0,300] m.

Isosurface:     Surface defined by  $S(x,y,z)=const$ , where  $S$  is the
                  scalarfield.

RBCont:         Contour plot of range vs. bearing. (Slice perpendicular
                  to the z axis.)

RDCont:         Contour plot of range vs. depth.

BDCont:         Contour plot of bearing vs. depth.

Frontpanel:     Colorbar and text strings can be showed floating on
                  top of the other objects

```


SEE ALSO: CONT, SCALARFIELD

OBJECT DEFINITION:

```

struct = ( WIN3D, $
    plot:      OB,      $ ; 0: SE, 1: Max speed, 2: Opt sens.par.
    detail:    OB,      $ ; 0B=minimal detail, 1B=full detail
    SMserver:  OBJ_NEW(), $ ; GSMserver object (NOT destroyed).
    oScene:    OBJ_NEW(), $ ; Contains oFront, oGeo and lightsources
    oFront:    OBJ_NEW(), $ ; IDLgrModel for the colorbar and info.
    oCScale:   OBJ_NEW(), $ ; Colorbar
    oCSAxis:   OBJ_NEW(), $ ; Tick values for the colorbar
    oCSTitle:  OBJ_NEW(), $ ; Colorbar title
    infoStrings: PTR_NEW(), $ ; Figure text strings
    oInfoText: OBJ_NEW(), $ ; IDLgrText object to display strings
    legendStr: PTR_NEW(), $
    legendTitle: '', $
    oGeo:      OBJ_NEW(), $ ; IDLgrModel for the SE data
    oIsoSurf:  OBJ_NEW(), $ ; Isosurface object
    oTop:      OBJ_NEW(), $ ;
    oBottom:   OBJ_NEW(), $ ;
    oInner:    OBJ_NEW(), $ ;
    isoStyle:  OB,      $ ; BIT 0: visibility on/off
    threshold: 0.0,    $ ; Isosurface threshold
    cuts:      OB,      $ ; if 1: Draw cuts
    volume:    OB,      $
    frame:     OB,      $
    oRBCont:   OBJ_NEW(), $ ; Contour plot - range vs. bearing
    oVolume:   OBJ_NEW(), $
    sectorStart: 0,    $
    sectorStop: 0,    $
    sectorSize: 0.0,  $
    sectorStyle: OB,   $
    oRD1Cont:  OBJ_NEW(), $ ; Contour plot - range vs. target depth
    oRD2Cont:  OBJ_NEW(), $ ;
    oRD1Frame: OBJ_NEW(), $ ; Polyline that emphasizes pos of RD1Cont
    oRD2Frame: OBJ_NEW(), $ ;
    oBDCont:   OBJ_NEW(), $ ; Contour plot - bearing vs. target depth
    se_data:   OBJ_NEW(), $ ; Signal Excess scalarfield(not destroyed)
    target_depth: 0,    $ ; \
    range:     0,      $ ; ) target position
    bearing:   0,      $ ; /
    c_cols:    PTR_NEW(), $ ; List of contour colors
    v_range:   [0.0, 0.0], $
    tlb:       0L,     $ ; Top widget
    visible:   OB,     $ ; =0 if window is closed, 1 otherwise
    view:      OBJ_NEW(), $
    vrect:     FLTARR(4), $
    oTrackball: OBJ_NEW(), $
    printer:   OBJ_NEW(), $
    window:    OBJ_NEW() $
)

```

MEMBER PROCEDURES/FUNCTIONS:

CLEANUP

EXPOSE_EVENTS Handle window expose and motion/button events here.
 event

PROCESS_MENU Handles events created by the menubar and its submenus.
 event

EVENT Resize event
 event

SHOW
 flag 0: Hide window, 1: Show window

SET_TRANSFORM
 TYPE=

GET_COLTAB()
 n

GET_TEXTURE() Creates the texture (IDLgrImage object) used for the
 contour plots. If COL_TAB is a scalar, a 1D texture is
 created. If COL_TAB is a 2 element vector, a 2D texture
 is returned.

COL_TAB=
 N_COLORS=
 TRANS=

DRAW_SE

CUT_DATA
 DATA=
 XAXIS=
 YAXIS=
 ZAXIS=
 SECTOR_BEARING=

BUILD_CUTS
 DATA1=
 DATA2= optional
 XAXIS=
 YAXIS=
 ZAXIS=

BUILD_ISOSURF
 DATA=
 XAXIS=
 YAXIS=
 ZAXIS=

BUILD_VOLUME

```

BUILD_CSCALE          Creates a vertical colorbar with tickvalues and title

BUILD_LEGEND

BUILD_INFOTEXT        Creates the figure text

SET_PROPERTY
  C_COLORS=           [4,n] array (RGBA)
  RANGE=              Signal Excess [min, max]
  STRINGS=            Array of strings to display

INIT()                Initialization of WIN3D object.
                    All plotted data is taken from the GSMServer object.

  smsserver           GSMserver reference
  GROUP_LEADER=
  XSIZE=
  YSIZE=              Default size of window
  TITLE=              Window title
  C_COLORS=           [4,n] array (RGBA), used for contour plots
  MAP=                0: Hide window, 1: Show window

```

NON-MEMBER PROCEDURES/FUNCTIONS:

```

Win3D_Cleanup         This is the clean up routine called when the TLE dies.
  tlb

Win3D_Expose_Events
  event

win3d_menu_event
  event

Win3D_Event
  event

```


B.12 Trackball

 FILE: xtrackball__define.pro

OBJECT: XTRACKBALL

PURPOSE:

This object translates widget events for draw widgets into transformations that emulate a virtual trackball (for transforming object graphics in three dimensions).

CATEGORY:

Object Graphics.

CALLING SEQUENCE:

To initially create:

oTrackball = OBJ_NEW('Trackball', Center, Radius)

To update the trackball state based on a widget event:

oTrackball->Update, sEvent

To re-initialize the trackball state:

oTrackball->Reset, Center, Radius

To destroy:

OBJ_DESTROY, oTrackball

INPUTS:

XTRACKBALL::INIT:

Center: A two-dimensional vector, [x,y], representing the requested center (measured in device units) of the trackball.

Radius: The requested radius (measured in device units) of the trackball.

XTRACKBALL::UPDATE:

sEvent: The widget event structure. The event type indicates how the trackball state should be updated.

XTRACKBALL::RESET:

Center: A two-dimensional vector, [x,y], representing the requested center (measured in device units) of the trackball.

Radius: The requested radius (measured in device units) of the trackball.

KEYWORD PARAMETERS:

XTRACKBALL::INIT:

AXIS: Set this keyword to indicate the axis about which rotations are to be constrained if the CONSTRAIN keyword is set to a nonzer value. Valid values include:

0 = X-Axis

1 = Y-Axis

2 = Z-Axis (default)

CONSTRAIN: Set this keyword to a nonzero value to indicate that the trackball transformations are to be constrained about a given axis (as specified by the **AXIS** keyword). The default is zero (no constraints).

MOUSE: Set this keyword to a bitmask to indicate which mouse button to honor for trackball events. The least significant bit represents the leftmost button, the next highest bit represents the middle button, and the next highest bit represents the right button. The default is 1b, for the left mouse button.

XTRACKBALL::UPDATE:

TRANSFORM: Set this keyword to a named variable that upon return will contain a floating point 4x4 array if a transformations matrix is calculated as a result of the widget event.

XTRACKBALL::RESET:

AXIS: Set this keyword to indicate the axis about which rotations are to be constrained if the **CONSTRAIN** keyword is set to a nonzero value. Valid values include:

0 = X-Axis

1 = Y-Axis

2 = Z-Axis (default)

CONSTRAIN: Set this keyword to a nonzero value to indicate that the trackball transformations are to be constrained about a given axis (as specified by the **AXIS** keyword). The default is zero (no constraints).

MOUSE: Set this keyword to a bitmask to indicate which mouse button to honor for trackball events. The least significant bit represents the leftmost button, the next highest bit represents the middle button, and the next highest bit represents the right button. The default is 1b, for the left mouse button.

OUTPUTS:

XTRACKBALL::UPDATE:

This function returns a 1 if a transformation matrix is calculated as a result of the widget event, or 0 otherwise.

EXAMPLE:

Create a trackball centered on a 512x512 pixel drawable area, and a view containing the model to be manipulated:

xdim = 512

ydim = 512

wBase = WIDGET_BASE()

```

wDraw = WIDGET_DRAW(wBase, XSIZE=xdim, YSIZE=ydim, $
    GRAPHICS_LEVEL=2, /BUTTON_EVENTS, $
    /MOTION_EVENTS, /EXPOSE_EVENTS, RETAIN=0 )
WIDGET_CONTROL, wBase, /REALIZE
WIDGET_CONTROL, wDraw, GET_VALUE=oWindow

oTrackball = OBJ_NEW('Trackball', [xdim/2.,ydim/2.], xdim/2.)
oView = OBJ_NEW('IDLgrView')
oModel = OBJ_NEW('IDLgrModel')
oView->Add, oModel

XMANAGER, 'TrackEx', wBase

```

In the widget event handler, handle trackball updates.
 As the trackball transformation changes, update the transformation
 for a model object (instance of IDLgrModel), and redraw the view:

```

PRO TrackEX_Event, sEvent
...
bHaveXform = oTrackball->Update( sEvent, TRANSFORM=TrackXform )
IF (bHaveXform) THEN BEGIN
    oModel->GetProperty, TRANSFORM=ModelXform
    oModel->SetProperty, TRANSFORM=ModelXform # TrackXform
    oWindow->Draw, oView
ENDIF
...
END

```

MODIFICATION HISTORY:

Written by: DD, December 1996

Changed by: EHN, Feb 1998

Implemented zoom and translation, changed name to XTRACKBALL

OBJECT DEFINITION:

```

struct = {xtrackball, $
    btndown: 0b, $
    axis: 0, $
    constrain: 0b, $
    mouse: 0b, $
    center: LONARR(2), $
    radius: 0.0, $
    zoombutton: 0b, $
    zoompos: 0, $
    zbdown: 0b, $
    zoomfactor: 0.0, $
    transbutton: 0b, $
    transx: 0, $
    transy: 0, $
    tbdown: 0b, $

```



```
    transfactor: 0.0, $  
    world: OBJ_NEW(), $  
    pt0: FLTARR(3), $  
    pt1: FLTARR(3) $  
}
```

MEMBER PROCEDURES/FUNCTIONS:

```
UPDATE()  
    sEvent  
    TRANSFORM=
```

```
INIT()  
    center  
    radius  
    AXIS=  
    CONSTRAIN=  
    MOUSE=  
    WORLD=  
    ZOOMBUTTON=  
    TRANSBUTTON=
```

```
CLEANUP
```

```
RESET  
    center  
    radius  
    AXIS=  
    CONSTRAIN=  
    MOUSE=  
    ZOOMBUTTON=  
    TRANSBUTTON=
```

NON-MEMBER PROCEDURES/FUNCTIONS:

```
XTRACKBALL_CONSTRAIN()  
    pt  
    vec
```

B.13 Sonarmap

```
-----
FILE:   sonarmap__define.pro
OBJECT: SONARMAP
-----
```

PURPOSE:

The intention of SONARMAP is to visualize the ocean bottom and surface, with axes and own position. At the moment the class is only capable of drawing a box.

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 17 Feb 1998

OBJECT DEFINITION:

```
struct = { sonarmap, $
            INHERITS IDLgrModel, $
            depth: 0.0, $
            oBottom: OBJ_NEW(), $
            oSurface: OBJ_NEW(), $
            oBox: OBJ_NEW(), $
            oGrid: OBJ_NEW(), $
            limit: [ 0.0, 0.0, 0.0, 0.0 ] $
          }
```

MEMBER PROCEDURES/FUNCTIONS:

```
INIT()
  DEPTH=
  LIMIT=
  GRID=
  _EXTRA=
```

```
CLEANUP
```

```
SET_PROPERTY
  THICK=
```

B.14 Object graphics contour plots

```
-----
FILE:   cont__define.pro
OBJECT: CONT
-----
```

PURPOSE:

Cont is inherited from IDLgrModel (IDL 5 object graphics) and may be used to create contour plots. The contouring is implemented using one dimensional texture mapping.

The contour plots may be visualized in three different forms:

- cartesian coordinates (in xy plane)
- polar coordinates (in xy plane)
- on a cylinder (cylinder axis parallel to z axis)

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1998
 Irregular mode not implemented

OBJECT DEFINITION:

```
struct = { cont, $
           INHERITS IDLgrModel, $
           polar:    0, $
           irregular: 0, $
           cyl_rad:  0.0, $
           xd:       PTR_NEW(), $
           yd:       PTR_NEW(), $
           zd:       PTR_NEW(), $
           zd2:      PTR_NEW(), $
           range:    [0.0, 0.0], $
           range2:   [0.0, 0.0], $
           elevation: 0.0, $
           texture:  OBJ_NEW(), $
           texrange: [0.0, 0.0], $
           texrange2: [0.0, 0.0], $
           oPoly:    OBJ_NEW(), $
           axisOn:   0b, $
           oAxis:    OBJ_NEW() $
           }
```

MEMBER PROCEDURES/FUNCTIONS:

READ_GSM Reads ASCII data file. See the independent procedure read_gsm for details.

 filename

BUILDPOLY

BUILDPOLY_REG

BUILDPOLY_POLAR

BUILDPOLY_CYL

BUILDAxis

SET_TEXTURE
 texture [4,n] array (RGBA) or IDLgrImage pointer

SET_PROPERTY For a description of the keywords, see INIT method.
 ZDATA1=
 ZDATA2=
 XDATA=
 YDATA=
 RANGE1=
 RANGE2=
 CYL=
 TEXTURE=

INIT() Initialization of CONT object.
 XDATA= X axis (1D array)
 YDATA= Y axis (1D array)
 ZDATA1= Grid data (2D array)
 ZDATA2= Additional grid data
 POLAR= 0: Cartesian coords, 1: Polar coords
 CYL= Radius of cyl
 AXIS= 1: put axes on plot
 IRREGULAR= Irregular data
 RANGE1= Z data range
 RANGE2= Z2 data range
 ELEVATION= if != 0.0 plot 3D
 TEXTURE= Same as in SET_TEXTURE
 _EXTRA= Additional parameters will be passed on to IDLgrModel

CLEANUP

B.15 Plot routines

 FILE: polcont.pro

PROCEDURES/FUNCTIONS:

polcont
 data
 rmin
 rmax
 argmin
 argmax
 _EXTRA=

 FILE: nncontour.pro

PURPOSE:

nncontour (Nearest Neighbour contour) creates a contour plot where the data is approximated by the nearest neighbor method. Data has to be defined on a regular grid.

XRANGE may be specified, but the YRANGE is always the same as the y axis.

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1998

PROCEDURES/FUNCTIONS:

nncontour
 data 2D array
 x x axis
 y y axis
 LEVELS= contour levels
 C_COLORS= and corresponding colors
 XRANGE=
 _EXTRA= Additional parameters is passed on to "contour"

B.16 Data loading routines

 FILE: read_gsm.pro

PROCEDURES/FUNCTIONS:

```

read_gsm
  filename
  r
  theta
  data
  TARGET_DEPTH=
  LATITUDE=
  LONGITUDE=
  DATE=
  TIME=
  CLOSED=
  REFLECT=
  
```

 FILE: read_xdr.pro

PURPOSE:

read_xdr is a procedure to read 3D data stored in binary xdr format (single precision floating point).

The format is:

```

int dim1
int dim2
int dim3      Number of data points in each dimension
flt x_1
:   :         Array of n=dim1 points
flt x_n
flt y_1
:   :         Array of m=dim2 points
flt y_m
flt z_1
:   :         Array of p=dim3 points
flt z_p
flt data[n*m*p] Array of n*m*p floats
  
```

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 23 Feb 1998

PROCEDURES/FUNCTIONS:

```

read_xdr
  filename      Name of file to read
  range         OUT: Range axis (1D array)
  theta         OUT: Theta axis
  depth         OUT: Depth axis
  data          OUT: 3D data array
  SYMMETRIC=
  CLOSED=
  
```


B.17 Utilities

This section contains descriptions of procedures/classes which can be used to generate polar contour plots of signal excess data. The result can be stored on postscript files.

```
-----
FILE:   gsmreport.pro
-----
```

```
PROCEDURES/FUNCTIONS:
```

```
  xds_report
  server
```

```
gsmreport          Creates a GSMServer object and calls the procedure
                    xds_report which prints a collection of polar
                    contour plots to PostScript files - 12 plots on
                    each page.
```

```
-----
FILE:   plotarr_define.pro
-----
```

```
OBJECT: PLOTARR
```

```
PURPOSE:
```

The object provides methods to plot several polar contour plots on a single page, together with a color bar. The color bar can be placed horizontally below the plots or vertically on the left hand side.

The data to plot must be given in a 3D array of dimension [n, n_range, n_theta] where n is the total number of plots on the page, and [n_range, n_theta] is the dimension of the data for each plot, i.e. all plots must have the same grid / number data points.

The class has been tested only for the layouts [3,4] and [3,1].

```
SEE ALSO:
```

```
MODIFICATION HISTORY:
```

Written by Erik Hamran Nilsen, 6 May 1998

```
OBJECT DEFINITION:
```

```
struct = { PLOTARR, $
           config: [0, 0], $ ; layout of plots [ N_x, N_y ]
           data: PTR_NEW(), $ ; data array [n, r, theta]
           rad: PTR_NEW(), $ ; radial axis values
           theta: PTR_NEW(), $ ; arg. values
           title: "", $ ; Main title string
           subtitle: PTR_NEW(), $ ; List of subtitles
           TextSize: 0.0, $ ; Size of titles
           NLevels: 0, $ ; number of levels in contour plots
```

```

Zmin:      -15.0, $ ; max and
Zmax:      15.0, $ ; min level
Range:     50.0, $ ; radial range
PlotXSize: 250, $ ; \
PlotYSize: 250, $ ; ) defines the relative sizes of
CScaleWidth: 120, $ ; | the different parts
BottomMargin: 20, $ ; /
CScaleHor: 1B, $ ; 1: horizontal, 0: vertical
PaperX:    22.0, $ ; Papersize (for PostScript output)
PaperY:    14.0 $ ;
}

```

MEMBER PROCEDURES/FUNCTIONS:

```

DRAW          Draw multiple plots and color bar
  CTSIZE=     max colortable entry to use

```

```

DRAW_PS      Plot to color PostScript file
  PSFILE=    Name of file

```

```

SET_PROPERTY  Set object properties
  CONFIG=    plotarr layout [ N_x, N_y ]
  DATA=     data array [n, n_range, n_theta]
  RAD=       radial axis
  THETA=     arg.
  TITLE=     main title string
  SUBTITLE=  list of titles for each plot
  TEXTSIZE=  size of text (default = 1.0)
  RANGE=     radial range

```

```

INIT()       Initialization of object

```

```

-----
FILE:  xspslide.pro
-----

```

PROCEDURES/FUNCTIONS:

```

  xsp_slide_report
  server

```

```

xspslide      Creates a postscript file with three contour
               plots and a color bar. The sensor used is hull
               mounted, 3.5 kHz, and the three contour
               plots shows the signal excess at 10, 15 and
               20 kts for the location "Marsteinen" in January,
               sea state SS2 and 30 m target depth.

```

```

-----
FILE:  xspslide__define.pro

```

```

OBJECT: XSPSLIDE
-----

```

PURPOSE:

```

XSPSLIDE is a 3.5 kHz Hull mounted sonar. This object provides format
and filenames for SE data created by "Generic Sonar Model".

```

Name of data files:

```
data_dir/XSP4/XSP[speed]F0405[ocean][season]E22.SE
```

where the brackets should be substituted with each element in the arrays:

```
[speed]   = '01', '02', '03'   (actually 10, 15 and 20 kts)
[ocean]   = 'MT'
[season]  = 'W', 'S'
```

Each file should be a binary data file of the format specified in "read_xdr", with

```
range      = 0.5, 1.0, ..., 50.0
bearing    = 0, 20, ..., 340
target_depth = 10, 30, 75, 150
```

MODIFICATION HISTORY:

Written by Erik Hamran Nilsen, 07 May 1998

OBJECT DEFINITION:

```
struct = ( XSPSLIDE, $
           INHERITS sensor $
         )
```

MEMBER PROCEDURES/FUNCTIONS:

```
GET_FILENAME()      Returns filename of SE data file
TARGET_DEPTH=
SENSOR_PAR1=
SENSOR_PAR2=
SHIP=
SPEED=
OCEAN=
SEASON=
SEASTATE=
_EXTRA=
```

INIT()

```
DATA_DIR=          location of data
```