NPS-ME-02-005

NAVAL POSTGRADUATE SCHOOL
Monterey, California USA

3D MODEL OF THE ARIES AUTONOMOUS UNDERWATER VEHICLE (AUV), JAVADOC FOR DYNAMICS, SOFTWARE, AUV MISSION-VISUALIZATION WORKBENCH, AND AUV DYNAMICS CONTROL WORKBENCH IN MATLAB

by

Adrien Gruneisen and Yann Henriet

22 October 2002

Prepared for: Naval Postgraduate School, Monterey, CA and
Ecole Nationale d'Ingénieurs de Tarbes, Tarbes Cedex, France

[This page intentionally left blank]

This report was prepared for Naval Postgraduate School, Monterey, CA 93943
and Ecole Nationale d'Ingénieurs de Tarbes, 65016 Tarbes Cedex, France

and funded in cooperation with NPS Center for AUV Research, Naval Postgraduate School, Monterey, California, USA

This report was prepared by:

Adrien GRUNEISEN
Ecole Nationale d'Ingénieurs de Tarbes

Yann HENRIET
Ecole Nationale d'Ingénieurs de Tarbes

Reviewed by:

Associate Professor Don Brutzman
Undersea Warfare Research Group

Professor Anthony J. Healey
Department of Mechanical Engineering

Released by:

D. W. Netzer
Associate Provost and
Dean of Research

REPORT DOCUMENTATION PAGE			Form approved OMB No 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** 21 October 2002	**3. REPORT TYPE AND DATES COVERED** Technical Memorandum, February – July 2002	
4. TITLE AND SUBTITLE 3D Model of the Aries Autonomous Underwater Vehicle (AUV), JavaDoc for Dynamics, Software, AUV Mission-Visualization, and AUV Dynamics Control Workbench in Matlab			**5. FUNDING** **Office of Naval Research (ONR)** **N0001401AF00002**
6. AUTHORS Adrien Gruneisen and Yann Henriet			
7. PERFORMING ORGANIZATION NAMES AND ADDRESSES Naval Postgraduate School Center for AUV Research, 800 Dyer Road, Monterey, CA 93943 and Ecole Nationale d'Ingénieurs de Tarbes, 47 Avenue d'Azereix BP 1629, 65016 Tarbes Cedex			**8. PERFORMING ORGANIZATION REPORT NUMBER** NPS-ME-02-005
9. SPONSORING/MONITORING AGENCY NAMEAND ADDRESS Naval Postgraduate School, Monterey, CA 93943-5000			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** **A**
13. ABSTRACT A 3D Model of the research vehicle ARIES Autonomous Underwater vehicle (AUV) was developed to provide more realistic visual simulation capabilities using commercial 3D authoring tools. The model was then translated into Virtual Reality Modeling Language/X3D format for web portability and interactivity. Java code was developed and JavaDoc for Dynamics documentation was written to support the ongoing software development efforts at the Center for AUV Research. A preliminary integration of various tools used in mission planning and visualization, called the AUV Mission Visualization Workbench was developed to aid in mission planning and visualization. The workbench is a first pass on an integrated development environment and graphical user interface for multiple vehicle platforms, using dynamics algorithms and mission control planning tools. This work also included the integration of the AUV Dynamics Control Workbench in MATLAB.			
14. SUBJECT TERMS Autonomous Underwater Vehicle (AUV), Control Algorithms, Virtual Reality Modeling Language (VRML), Extensible 3D Graphics (X3D)			**15. NUMBER OF PAGES** **75**
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5800

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18

3D MODEL OF THE ARIES
AUTONOMOUS UNDERWATER VEHICLE (AUV),
JAVADOC FOR DYNAMICS, SOFTWARE
AUV MISSION-VISUALIZATION WORKBENCH, AND
AUV DYNAMICS CONTROL WORKBENCH IN MATLAB

ABSTRACT

Operating an Autonomous Underwater Vehicle (AUV) in real-world conditions is time consuming, expensive and prone to failure, because of the complexity of the tasks and of the machinery associated with AUV operations. A virtual world offers many advantages for testing and development of an AUV. The challenges are fewer and the barriers to practicing in a virtual world are less strenuous than in the hazardous undersea environment. Yet, it is still difficult work.

This technical report details one approach to modeling AUV operations in a virtual world. A 3D Model of the research vehicle ARIES AUV was developed to provide more realistic visual simulation capabilities using commercial 3D authoring tools. The model was then translated into Virtual Reality Modeling Language/X3D format for web portability and interactivity. Java code was developed and JavaDoc for Dynamics documentation was written to support the ongoing software development efforts at the Center for AUV Research. A preliminary integration of various tools used in mission planning and visualization, called the AUV Mission Visualization Workbench was developed to aid in mission planning and visualization. The workbench is a first pass on an integrated development environment and graphical user interface for multiple vehicle platforms, using dynamics algorithms and mission control planning tools. This work also included the integration of the AUV Dynamics Control Workbench in MATLAB.

3D MODEL OF THE ARIES AUTONOMOUS UNDERWATER VEHICLE (AUV), JAVADOC FOR DYNAMICS, SOFTWARE AUV MISSION-VISUALIZATION WORKBENCH, AND AUV DYNAMICS CONTROL WORKBENCH IN MATLAB

TABLE OF CONTENTS

Table of Figures

I. INTRODUCTION

A. BACKGROUND

Autonomous Underwater Vehicles (AUVs) are designed to independently accomplish complex tasks either in deep oceans or shallow water. A meticulous design must be followed during conception of the AUV, since little or no communication with distant human supervisors is possible during regular operations. Thus, the underwater domain imposes many limitations and restrictions on hardware.

The Center for AUV Research at the Naval Postgraduate School (NPS) has been working for 14 years on several AUV prototypes, with each improvement showing further success. The latest NPS AUV is called Acoustic Radio Interactive Exploratory Server (ARIES) and is fully operational. Currently ARIES operates for short missions in Monterey Bay, California USA.

During operations, data sets gathered from the ARIES include track positions, bathymetry (for each sample point), sonar and video data, contact coordinates, image, etc.

All of this data helps to reconstruct what happened during a mission. Nevertheless those information streams are merely raw data and it is very difficult to observe AUV operations. Thus an underwater virtual world is needed to comprehensively model all AUV missions and all characteristics of the real world where the AUV moves around.

B. MOTIVATION

A virtual world using 3D graphics for the ARIES provides an excellent design alternative to observe and understand its operations. Because of its high level of realism, a virtual world has the potential to completely change how people observe and analyze post-mission data.

The Virtual Reality Modeling Language (VRML), specially created to design virtual worlds, is a good choice for designing such tasks. Not only suited to 3D virtual worlds, VRML is also a good way to share information and make these experiments available via the World Wide Web. Extensible 3D (X3D) improvements to VRML provide further benefits.

For our purposes, the counterpoint to use the virtual world 3D is the "Real-time Model." Real-time in this context is defined by the requirement that a vehicle maneuvering, within the virtual world, describe essentially the same path and postures as the vehicle maneuvering in the real world. This requires that the robot hardware and software receive the same responsiveness from the virtual world as from the real world. To allow the same behavior of the robot, whether operating in the real world or the virtual world, two software languages have been used: Java and MATLAB.

C. ORGANIZATION OF THE REPORT

This report describes a number of tools created in order to support AUV software development and mission visualization. It is organized into seven chapters:

- Chapter I is the present introduction.
- Chapter II is a NPS AUV overview and a presentation of related works to the ARIES.
- Chapter III explains the modeling of the AUV using 3D Studio Max and X3D-Edit.
- Chapter IV is about the dynamics program which is able to simulate the expected AUV response in the real environment.
- Chapter V describes the new interface for launching the simulation built using JAVA for robot control-software development.
- Chapter VI is about the MATLAB dynamics program and the new interface.
- Chapter VII provides conclusions and recommendations for future work.

Appendices and associated research products comprise the final section of this report.

II. NPS OVERVIEW

A. INTRODUCTION

Research on Autonomous Underwater Vehicles (AUVs) has been an ongoing project at the Naval Postgraduate School (NPS) in Monterey, California USA since 1987. Several AUVs followed one another, increasing operational capabilities and becoming more robust as they become more sophisticated in terms of hardware and computer software.

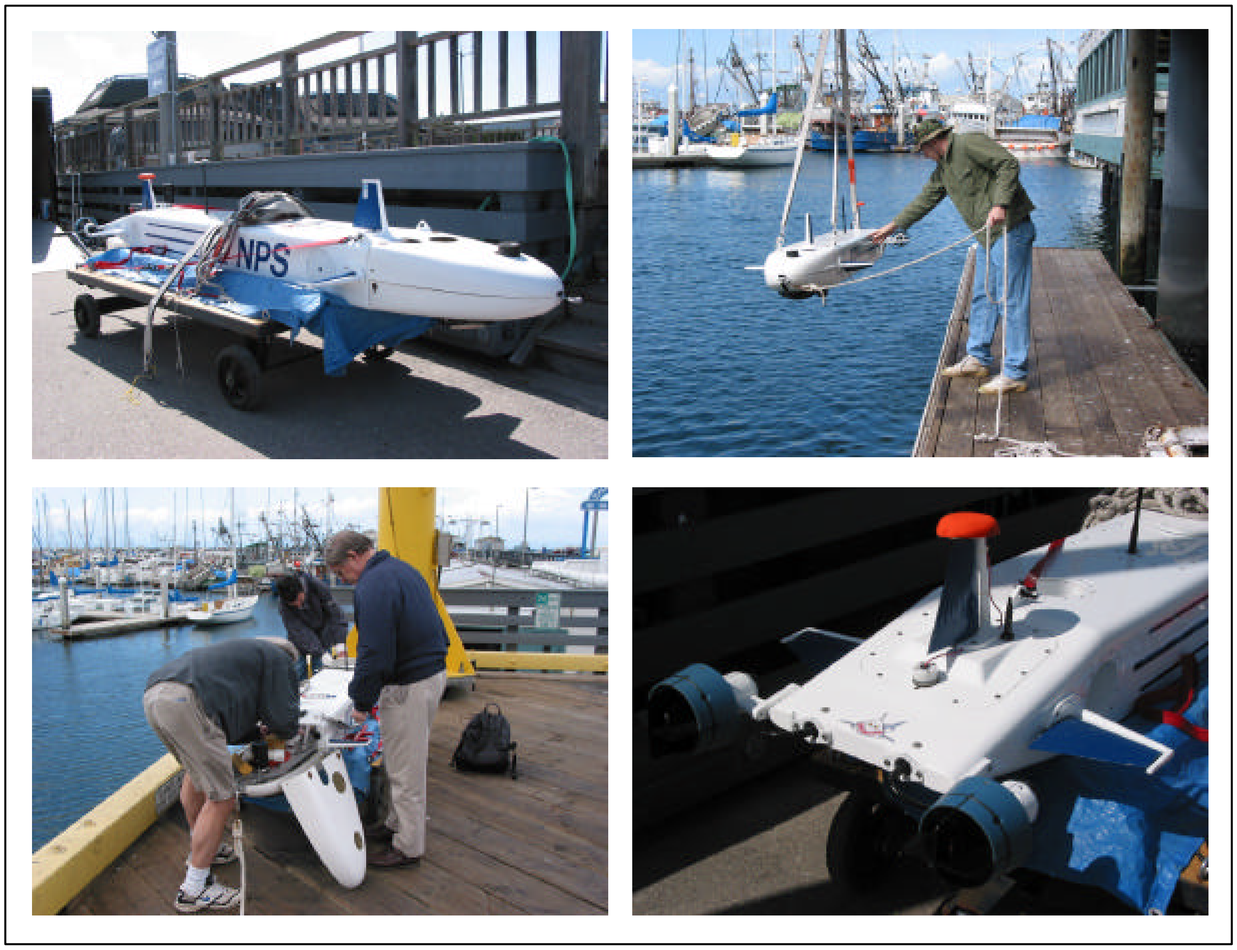


Figure 1. Autonomous Underwater Vehicle Acoustic Radio Interactive Exploratory Server.

The latest NPS vehicle is named Acoustic Radio Interactive Exploratory Server (ARIES). This vehicle is a student-research test bed for shallow-water minefield-mapping missions, operating in the literal ocean. Currently the vehicle operates regularly in Monterey Bay.

The following section is a general overview of the NPS AUV. It provides a general description of the hardware and the software architecture of the vehicle.

1. AUV ARIES Presentation

a. ARIES Hardware

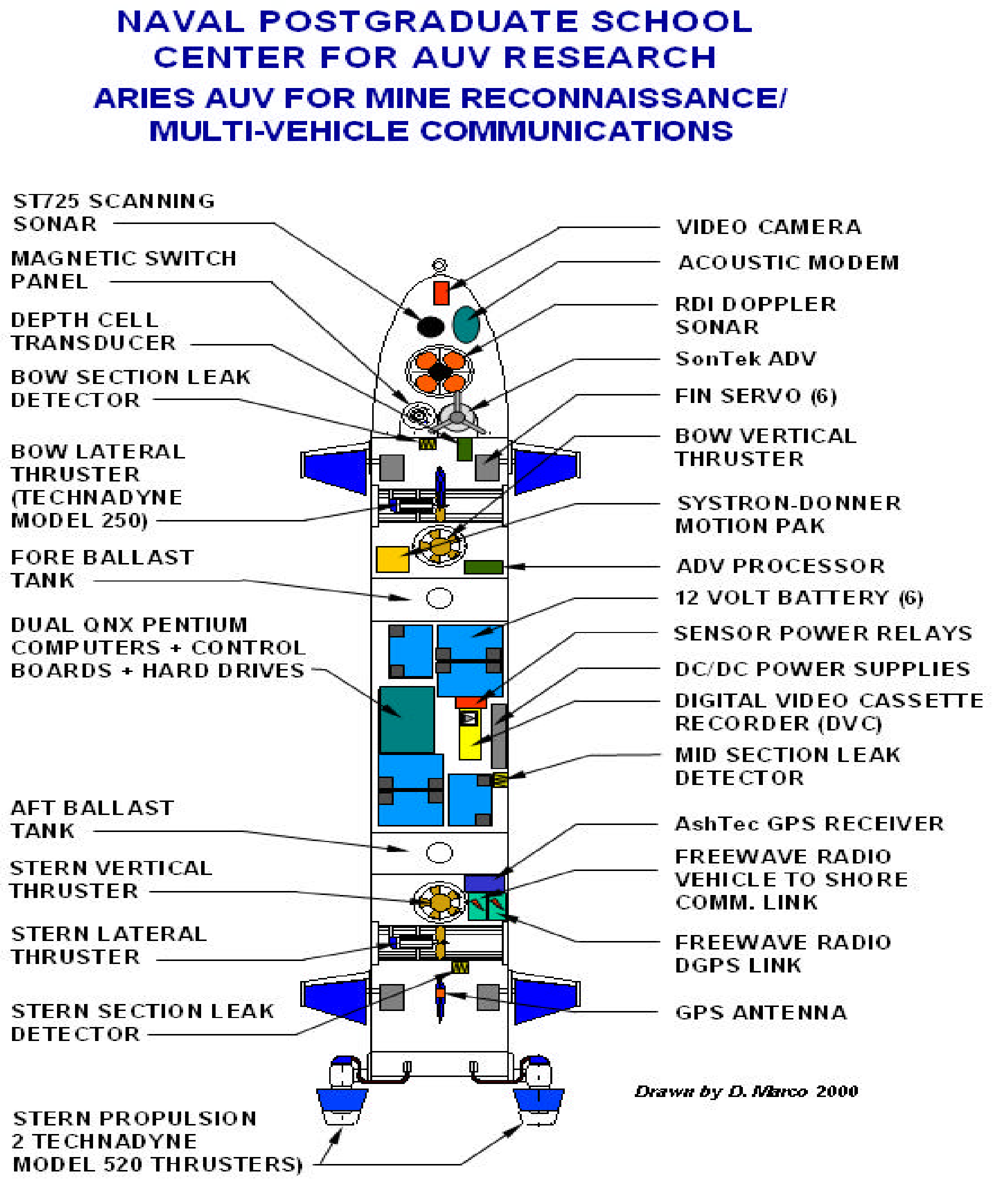

NAVAL POSTGRADUATE SCHOOL
CENTER FOR AUV RESEARCH
ARIES AUV FOR MINE RECONNAISSANCE/
MULTI-VEHICLE COMMUNICATIONS
ST725 SCANNING SONAR
MAGNETIC SWITCH PANEL
DEPTH CELL TRANSDUCER
BOW SECTION LEAK DETECTOR
BOW LATERAL THRUSTER (TECHNADYNE MODEL 250)
FORE BALLAST TANK
DUAL QNX PENTIUM COMPUTERS + CONTROL BOARDS + HARD DRIVES
AFT BALLAST TANK
STERN VERTICAL THRUSTER
STERN LATERAL THRUSTER
STERN SECTION LEAK DETECTOR
STERN PROPULSION 2 TECHNADYNE MODEL 520 THRUSTERS)
VIDEO CAMERA
ACOUSTIC MODEM
RDI DOPPLER SONAR
SonTek ADV
FIN SERVO (6)
BOW VERTICAL THRUSTER
SYSTRON-DONNER MOTION PAK
ADV PROCESSOR
12 VOLT BATTERY (6)
SENSOR POWER RELAYS
DC/DC POWER SUPPLIES
DIGITAL VIDEO CASSETTE RECORDER (DVC)
MID SECTION LEAK DETECTOR
AshTec GPS RECEIVER
FREEWAVE RADIO VEHICLE TO SHORE COMM. LINK
FREEWAVE RADIO DGPS LINK
GPS ANTENNA
Drawn by D. Marco 2000

Figure 2. Hardware Components of the NPS ARIES.

Dimensions and Endurance: The Vehicle weighs 225 Kg and measures approximately 3 m long wide and 0.25 m high. The hull is constructed of 6.35 mm thick type 6061 aluminum and forms the main pressure vessel that house all electronics, computers and batteries. A flooded fiberglass nose is used to house the external sensors, key-controlled power "on/off" switches and status indicators. ARIES is capable of a top speed of 3.5 knots and is powered by six 12-volt rechargeable lead-acid batteries. Vehicle endurance is approximately 4 hours at top speed, with 20 hours endurance under "hotel load" only. The ARIES is primarily designed for shallow-water operations and can operate safely down to depths of 30 meters.

Propulsion and Motion Control Systems: Main propulsion is achieved using twin ½ Hp electric drive thrusters located at the stern. During normal submerged flight, heading and depth are controlled using upper bow and stern rudders plus a set of bow planes and stern planes. Since the control fins are ineffective during very slow (or zero) forward-speed maneuvers, vertical and lateral cross-body thrusters are used to control surge, sway, heave, pitch and yaw motions.

Navigation Sensors: The sensor suite used for navigations includes a 1200 kHz Instruments (RDI) Navigator Doppler Velocimeter Log (DVL) that also contains a TCM2 magnetic compass. This instrument measures the vehicle ground speed, altitude, and magnetic heading. Angular rates and accelerations are measured using a Systron Donner 3-axis Motion pak IMU. While surfaced, Global Positioning System (GPS) inputs is provided by a carrier-phase differential GPS (DGPS CP) system, available during surfaced operation to correct any navigational errors accumulated during the submerged phases of a mission.

Sonar and Video Sensors: Tritech ST725 scanning sonar and an ST1000 profiling sonar is used for obstacle avoidance and target acquisition/reacquisition. The sonar heads can scan continuously through 360 degree of rotation or swept through a predefined angular sector. A fixed-focus wide-angle video camera is located in the nose and is connected to a DVC recorder. The computer is interfaced to the recorder which controls on/off and start/stop record functions. While recording images, data for date, time, vehicle position, depth and altitude is superimposed on the video image.

Vehicle/Operator Communications: Radio modems are used for high-bandwidth command, control and system monitoring while the vehicle is deployed and surfaced. While submerged, an acoustic modem is used for low-bandwidth communications. In the laboratory environment, a 10 Mbps thin-wire Ethernet connection is used for software development and mission data upload and download.

b. Computer Hardware Architecture

The dual-computer system unit measures approximately 28 x 20 x 20 cm. It consists of two Ampro Little Board 166 MHz Pentium computers with 64 MB RAM, four serial ports, a network adapter and a 2.5 GB hard drive each. Two DC/DC voltage converters for powering both computer systems and peripherals are integrated into the computer package. The entire computer system draws a nominal 48 Watts. Both systems use TCP/IP sockets over thin-wire Ethernet for inter-processor communications as well as connections to an external LAN. The sensor data-collection computer is designated QNXT. The second is named QNXE and executes the various auto-pilots for servo-level control.

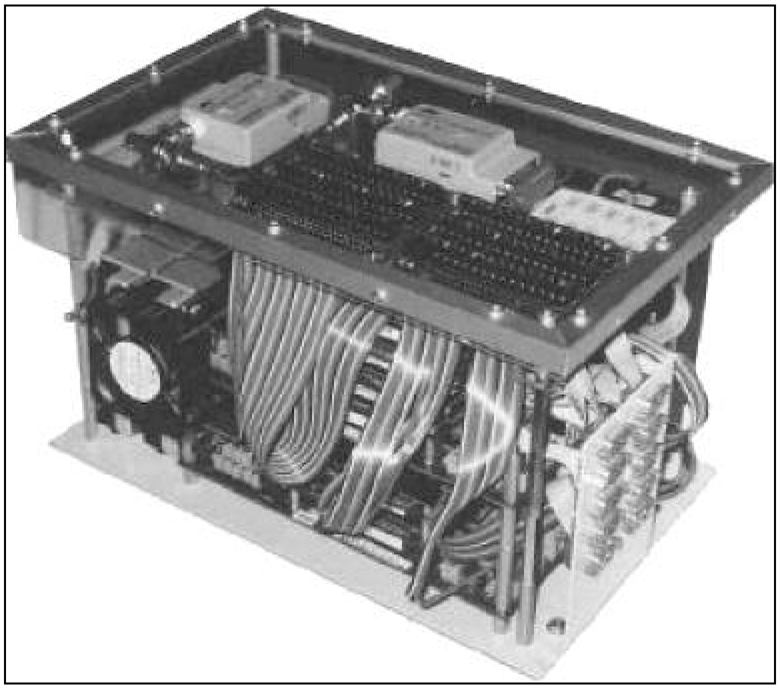


Figure 3. Dual Computer System Unit.

c. Computer Software Architecture

The ARIES AUV has used a tri-level software architecture called the Rational Behavior Model (RBM). RBM divides responsibilities into areas of open-ended strategic planning, soft-real-time tactical analysis and hard-real-time execution-level control. The RBM architecture has been created as a model of a manned submarine operational structure. The correspondence between the three levels and a submarine crew is shown in Figure 4 below.

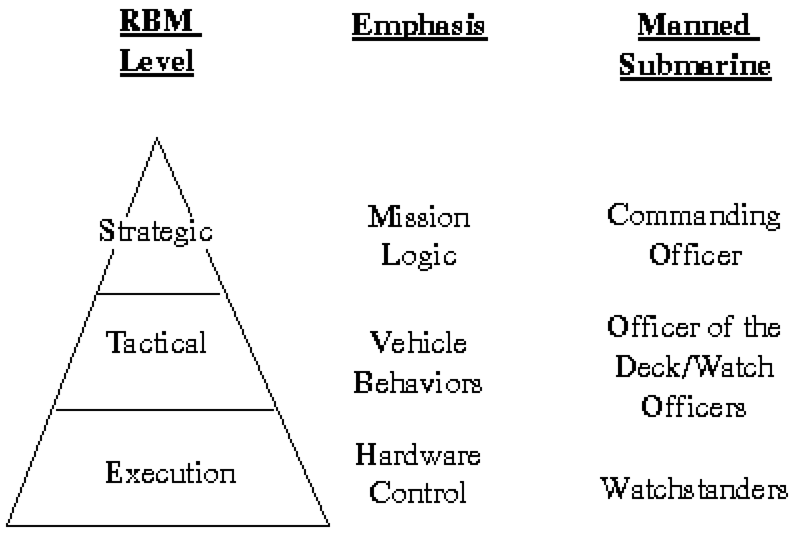


Figure 4. Relational Behavior Model [Brutzman 94].

This figure represents the tri-level software hierarchy with level emphasis and submarine equivalent listed. A functional summary of each level follows.

The **Execution Level** assures the interface between hardware and software. Its tasks are to maintain the physical and operational stability of the vehicle, to control the individual devices and to provide data to the tactical level. These tasks are currently performed by on-board host QNXS computer.

The **Tactical Level** provides a software level that interfaces with both the Execution Level and the Strategic Level. Its chores are to give to the Strategic level indications of vehicle state, completed tasks and execution level commands. The Tactical level selects the tasks needed to reach the goal imposed by the Strategic level. It operates in terms of discrete events.

The **Strategic Level** controls the completion of the mission goals. The mission specifications are inside this level.

2. ADS Capabilities

ADS is the acronym for AUV Data Server system. It is a software system developed at NPS and used to gather and translate AUV data into a format, suitable for input into the Mine Warfare Environmental Decision Aids Library (MEDAL) system. This format is used by the US Navy to evaluate asset positions, mine-like contacts, snipped images of those contacts identified as mines and bathymetry maps. Thus, data gathered by ADS from the AUV are track positions, bathymetry at each point, sonar and data video processing, image files for contact as well as their locations. Data are converted into Message Transfer Format (MTF) message formats and imported into MEDAL.

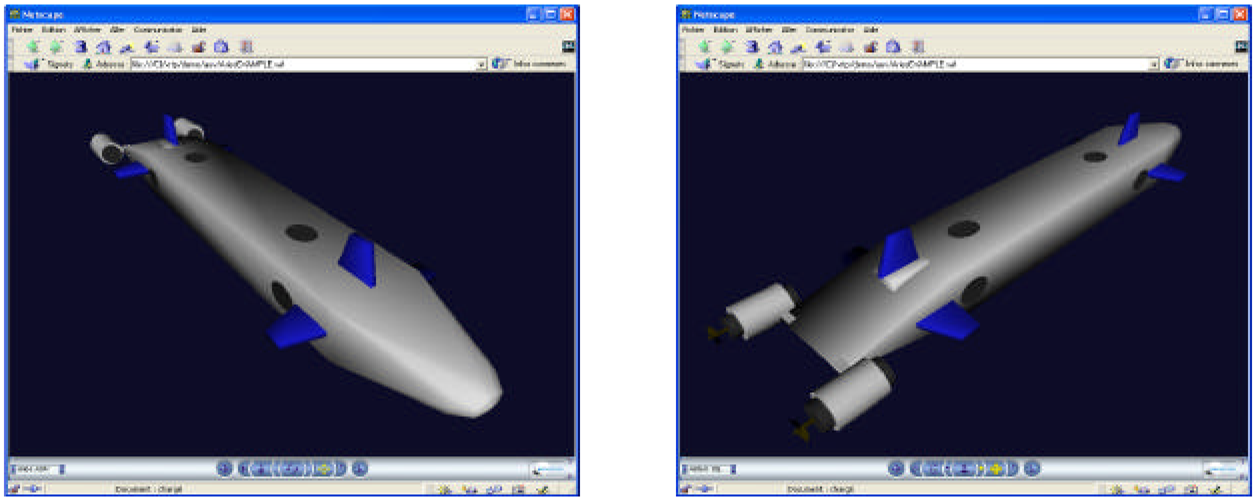


Figure 5. The Old ARIES AUV Model.

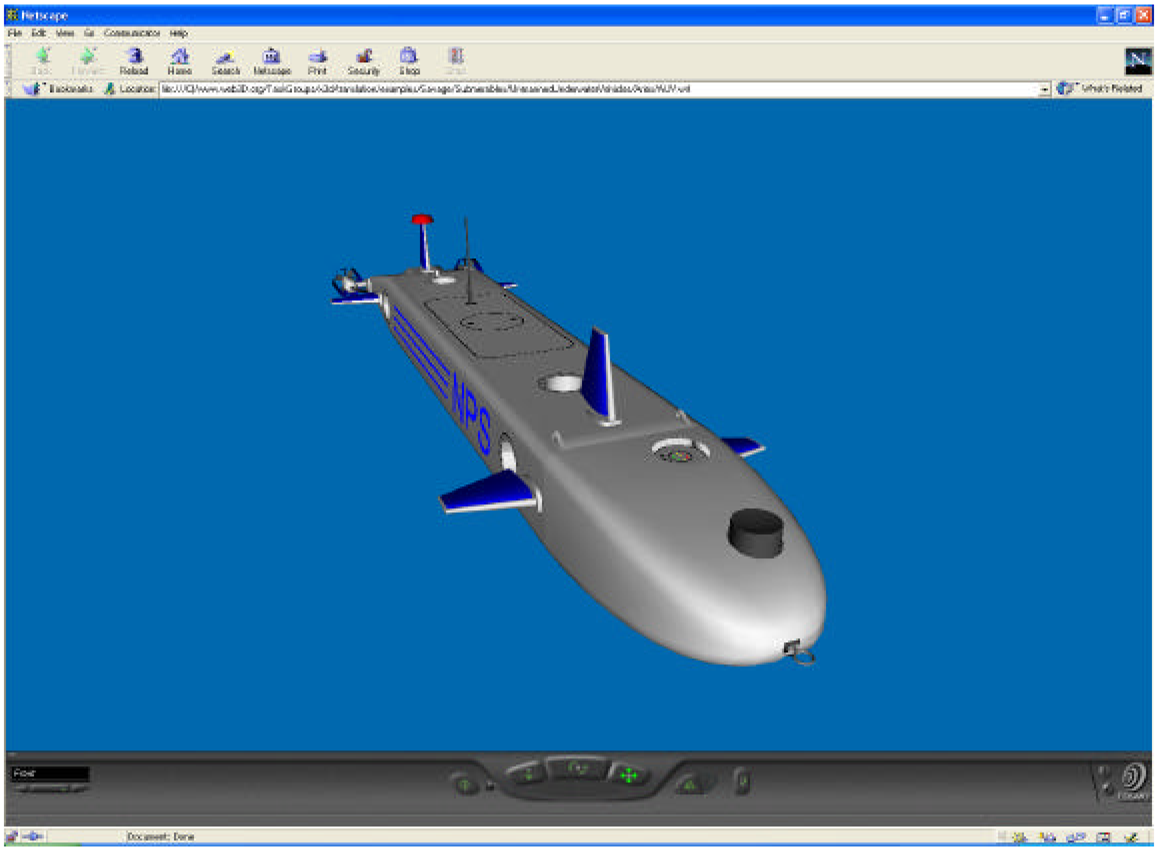


Figure 6. New Aries Model.

III. 3D MODELING OF THE ARIES AUV

A. INTRODUCTION

This chapter describes how the new virtual ARIES AUV model is created using the Virtual Modeling Language (VRML) and 3D Studio Max. The first part presents the VRML language, X3D-Edit and 3D Studio Max. The second part explains the modeling and the X3D file.

1. Motivation

The current VRML model of the ARIES AUV was built by Don Brutzman. This model has the advantage of being very light and thus bearable by the majority of the computers. However, the computers are now more and more powerful, so the creation of a more realistic model is now possible and makes the virtual simulation more realistic as well.

2. Software and Programming Language

a. The VRML Language

(1) VRML History

The Virtual Reality Modeling Language (VRML) was an idea conceived by Mark Perce and Tony Parisi, initially presented at the First International Conference of the World Wide Web in 1994. VRML was intended to be a platform independent language for web-based 3D graphics, and implemented on the internet.

The language needed to be able to place objects in 3D space, as well as include attributes such as shape, color and size. Since VRML was to be used in the Internet, all platforms needed to be able to support it: UNIX workstations, personal computers, etc.

The Silicon Graphics Open Inventor format was the initial basis for the VRML file format and after numerous improvements VRML was widely accepted. VRML 1.0 was introduced in 1995. In 1996, VRML 2.0 became the new VRML specification. In 1997, the revised language was certified by the International Organization for Standardization (ISO) as ISO/IEC and was commonly referred to as VRML 97 [reference – VRML 97 Specification].

(2) Presentation

Using VRML, an author can create 3D virtual worlds for display on the web. While VRML 1.0 had static worlds, which is to say that it allowed for no arbitrary behaviors for objects in the VRML world, VRML 97 provides for dynamic behaviors by adding Java and JavaScript support, as well as sound and animation.

The feature of VRML 97 is that it enables to create dynamic worlds and an interactive environment on the Internet, including the ability to:

- animate objects in the VRML world
- play sounds and movies
- allow users to interact with VRML worlds
- control and enhance worlds with scripts

Since authors are able to create effective 3D virtual worlds, VRML is an appropriate language for moderately complex global scene renderings. Nevertheless VRML is not a Computer Aided Design (CAD) tool. Creating complex shapes with level of detail implies using a professional CAD tool like a mechanical engineering program or professional 3D-design software. Nevertheless VRML is a good way for scientist, hobbyist and application developers to produce 3D models for use over the World Wide Web.

(3) Browsers and VRML

To present sophisticated multimedia, such as 3D VRML worlds, web browsers (like Microsoft Internet Explorer or Netscape Navigator) need help from compatible applications, called "plug-ins" that specifically understand content of different file formats. They enable users to view non-HTML information within the Web browser window.

Many VRML plug-ins are available as 3D browsers, including:

•	Nexternet: Pivoron player	http://www.nexternet.com(now defunct)
•	Cosmosoftware: Cosmoplayer	http://ca.com/cosmo
•	Parallel Graphics: Cortona player	http://www.parallelgraphics.com/cortona
•	Blaxxun: Contact browser	http://www.blaxxun.com
•	Xj3D – Open source browser	http://www.web3d.org/TaskGroups/source/Xj3D.html

VRML remains the preferred language to build non-proprietary virtual worlds and to present such work across the internet.

(4) Creating a Simple Object with VRML

VRML scenes can be created using a simple text editor. More developed VRML editors like X3D-Edit or Parallel Graphics VrmlPad are highly recommended (especially for the novice).

```
#VRML V2.0 utf8
Group {
   children [
      Viewpoint {
         description "initial view"
         position        6 -1 0
         orientation     0 1 0 1.57
      }
      Shape {
         geometry Sphere {
            radius 1
         }
         appearance Appearance {
            texture ImageTexture { url "earth.png" }
         }
      }
      Transformation {
         translation     0 -2 1.25
         rotation        0 1 0 1.57
         children [
            Shape {
               Geometry Text {
                  String [ "Hello" "world!" ]
               }
               appearance Appearance {
                  material Material {
                     diffuseColor 0.1 0.5 1
                  }
               }
            }
         ]
      }
   ]
}
```

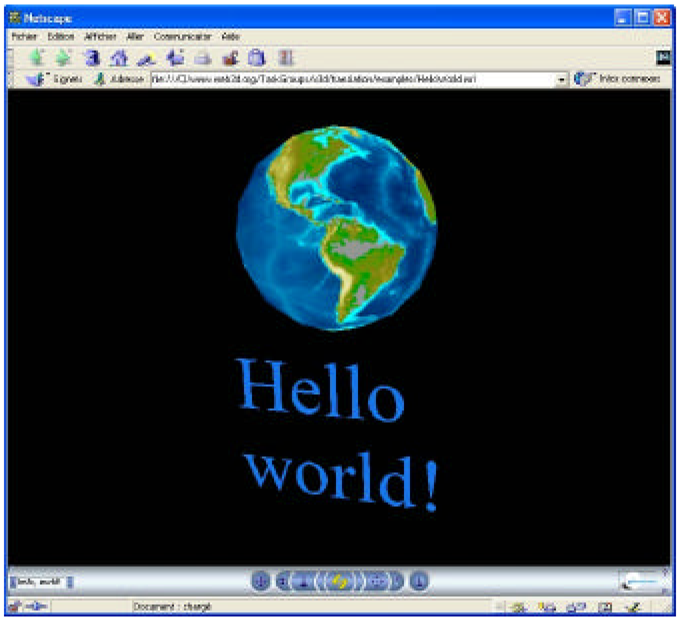


Figure 7. Exemplar Scene in Netscape 4.77, Using Pivoron Browser Plugin from Nexternet. (Above). VRML Encoding (Left).

b. *X3DScene Graph Editing Tool*

(1) Overview

X3D-Edit is an Extensible 3D (X3D) graphics file editor that uses the X3D Document Type Definition (DTD) in combination with Sun's Java, IBM's Xeena XML editor building application and an editor profile configuration file. X3D-Edit enables simple error-free editing, authoring and validation of X3D or VRML scene-graph files. The author of this useful XML editor is Don Brutzman from the Naval Postgraduate School (NPS). [http://www.web3d.org/TaskGroups/x3d/translation/README.X3D-Edit.html]

X3D-Edit is constructed using Xeena, IBM's tool-building application, and uses Xeena interface. Xeena is a valid XML editor and a generic Java application for editing valid XML documents derived from any valid DTD. The editor takes as input a given DTD and automatically builds a palette containing the elements defined in the DTD. Users can create, edit and expand any document derived from that DTD, by using a visual tree-directed paradigm.

Xeena features include:

- Intuitive viewing and editing of X3D documents in a tree control view.
- Editing multiple X3D documents.
- XML source viewer.
- Direct translation from X3D to VRML97 syntax using XML Styles Sheet X3dToVrml97.xsl
- Direct translation from X3D to documentation Html.
- Restrictions about adding and editing of features according to the DTD, and validity checking of produc ed documents.
- Easy customization of display.
- Element position and attribute value checking
- Context-sensitive tooltips in multiple languages (English, French, German, Spanish)

Therefore, all those features are automatically included in X3D-Edit. Since X3D-Edit is based on Xeena, users also need to install the Java Development Kit (JDK) or Java Runtime Environment (JRE), as Xeena is built on top of Java technology.

(2) X3D-Edit Interface

X3D-Edit has a user-friendly interface which is intuitive to use. An action toolbar allows editing, saving and validating XML files. A toolbar palette exposes various node profiles required to build a VRML scene.

Every time an object (node, field, comment, etc.) is selected and inserted by the author, it is inserted as directed using a visual tree-directed paradigm into the active document inside the work area. A corresponding attribute array appears in the edit area for the selected node. This is the place where field values are inserted. A message area points out whether there are syntax errors when validating the constructed scene.

It is very easy to build a scene with X3D-Edit because it is possible to copy, paste and move a node or group of nodes inside the view tree. When you insert a node, only children nodes and fields are available in the sidebar palette so as to avoid fatal syntax errors. Working with a tree paradigm even allows users who do not know the VRML syntax to build scenes.

Once the X3D file is created, it can be converted into a VRML file. X3D-Edit can make this conversion and launch the VRML browser (Internet Explorer or Netscape) automatically to see the result of the VRML scene. It can also convert XML files into HTML files that are easily readable and can be put on the Internet as scene documentation.

X3D-Edit also includes tooltips that provides users with the fundamental basis of VRML syntax as well as node and field definition, type, etc. in a context sensitive way. One ongoing objective for X3D-Edit is to further internationalize context-sensitive node and field tooltips by translating them in many languages (in the profile configuration). Currently, English, French, German and Spanish language tooltips are available.

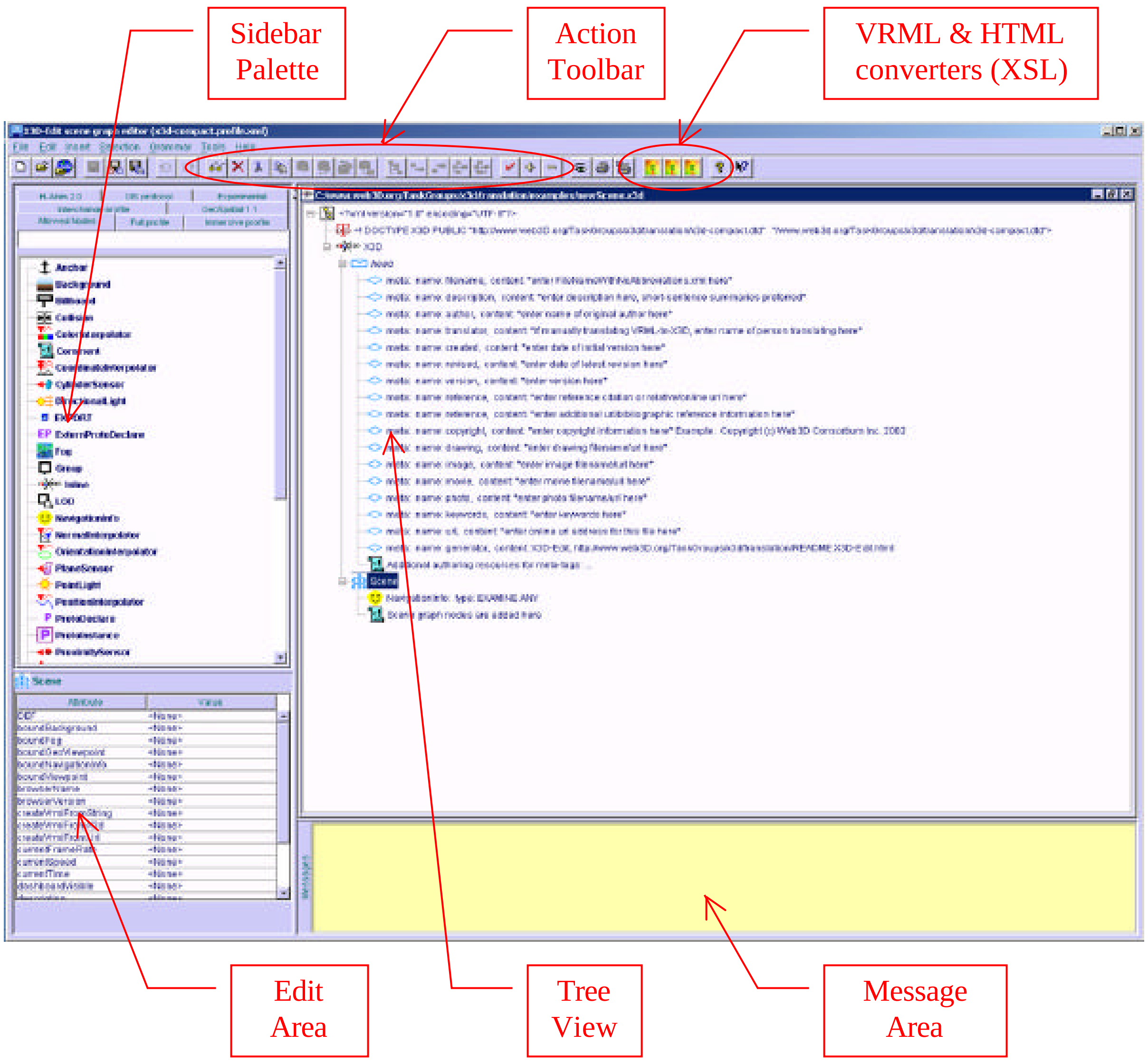


Figure 8. X3D-Edit Interface, Annotated.

c. 3D Studio Max

(1) Overview

3D Studio Max (3dsmax) is used to build and animate 3D graphics scenes. It is one of the leaders in this kind of software. It is frequently used for 3D graphics in general, animation movies (like “Shrek” or “Final Fantasy”), architecture or video games. This software is developed by Discreet: http://www.discreet.com

This popular software allows users to:

- Design 3D objects in two different modes: mesh or nurbs. There are several functions which allow creation and modification of the 3D objects.
- Make very realistic 3D scenes using the powerful material editor and the different environmental tools like lights, shadow, special effects, etc…
- Animate or create fixed scene.
- Render the 3D scenes in different formats like: jpeg, gif, avi, mpeg, etc…
- Import or export scenes in several kind of formats like: 3ds, dwg, dxf, iges, vrml97, etc…
- Extensive additional functionality is provided.

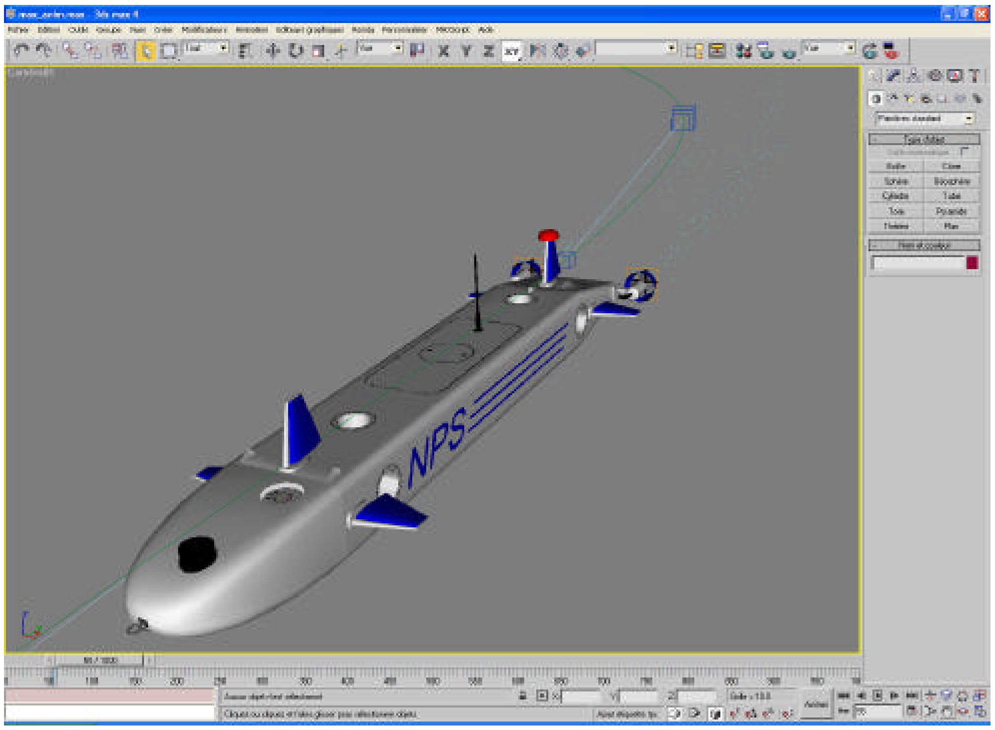


Figure 9. 3D Studio Max Interface Screen-Shot.

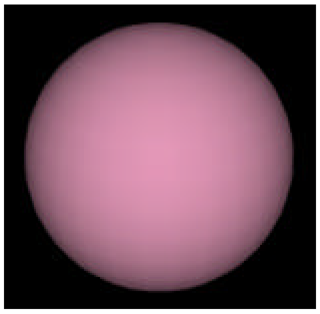
(2) Why Use 3dsmax Software?

X3D-Edit and VRML do not support direct modeling of complex shapes; only simple shapes are available like Box, Sphere, Cone, etc. To create complex shapes (like the different elements of the submarine), VRML uses different lists to build one by one the components of a 3D IndexedFaceSet.

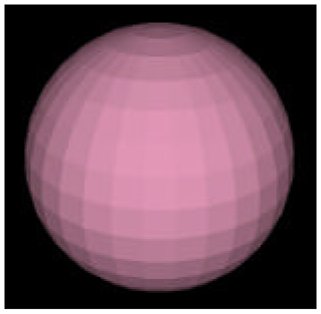
- one list of points coordinates

- one list of points indexes
- one list of normals coordinates
- one list of normals indexes

The list of Normals gives a smoothly shaded appearance to the shape:



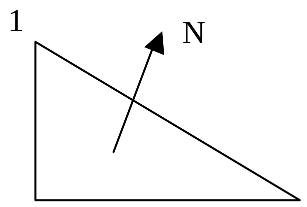
With normals



Without normals

Figure 10. Sphere with Normals and without Normals

For example, to create a simple mesh, VRML97 needs:



```
geometry IndexedFaceSet {
     coordIndex [ 1, 2, 3 ]
     normalIndex [ 1, 1, 1 ]
     coord Coordinate {
          point [ x1 y1 z1, x2 y2 z2, x3
     y3 z3 ]
     }
     normal Normal {
          vector [ xN yN zN ]
     }
```

Figure 11. IndexedFaceSet Definition Using VRML Syntax.

Thus, CAD software (like 3dsmax) is needed to make these lists of coordinates and indexes. 3dsmax has got many functions to build mesh objects and a VRML97 exporter which can make these lists. So this software corresponds perfectly for the modeling of the robot submarine.

3. 3D Modeling Using 3D Studio Max

a. Design and Realization

This chapter describes the different functions used to create and optimize the meshes of the shapes of the AUV.

Some duplicate 3D elements are modeled just once (fins and propellers) because X3D-Edit and VRML allows duplication of shapes using the same definitions (same lists) which make the VRML file smaller.

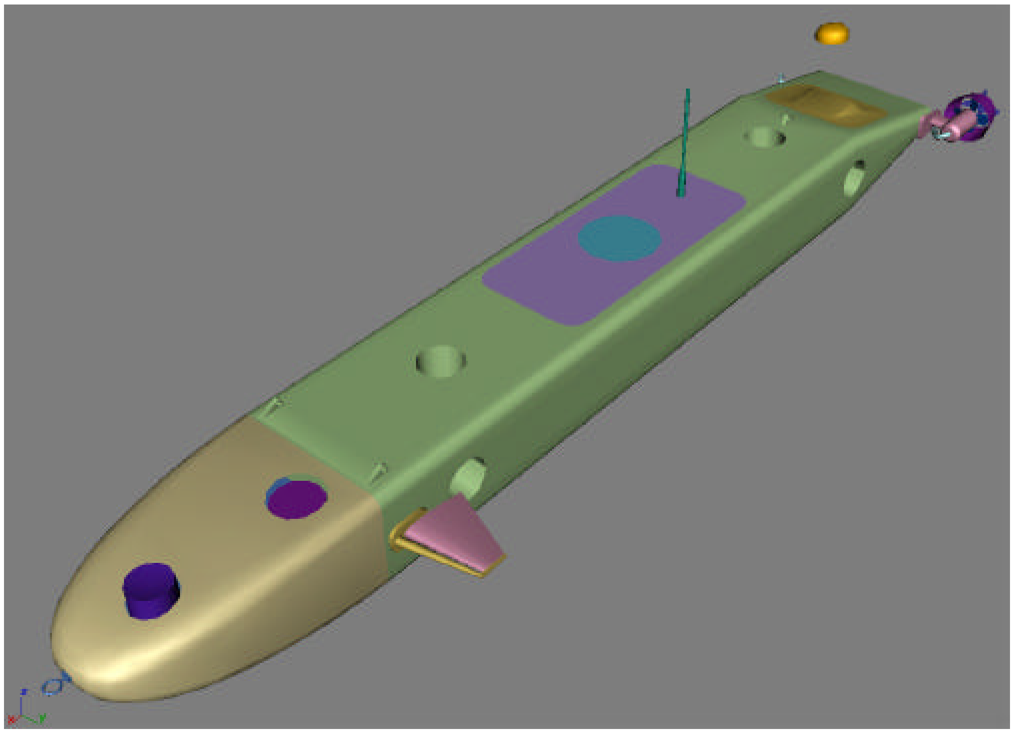


Figure 12. Screen-Shot of the Different Elements of the AUV On 3D Studio Max, Duplicate Copy-By-Reference Elements Are Not Shown.

(1) Modeling Functions

To modelize the different parts of the submarine, standard functions were used like:

- **Extrude**: which make a solid from a 2D profile.
- **Revolution**: which make a 3D object by “rotating” 2D splines.
- **Surface**: which build a surface from 2 or 4 splines.
- **Boolean operations**: add or subtract two objects.

After the modeling of the submarine, the difficult task of optimization is necessary. This additional work is necessary because the standard functions of 3dsmax generate forms with a too high level of detail (too many points and meshes) for efficient real-time rendering.

(2) MeshEdit Function

This function allows users to work directly on the mesh of the shape. Every point and face can be created, deleted, moved, and rotated. Since some functions like optimize do not always work very well in automatic modes, the user can modify manually the mesh of the 3D object to correctly apply optimization according to the function of the object.

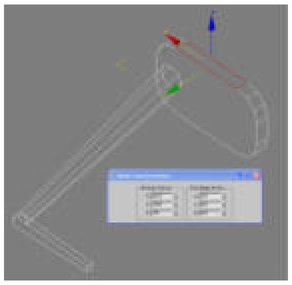


Figure 13. Optimization of the Fin Guard.

(3) MeshSmooth Function

This function allows users to generate the list of normals to smooth the surface of the shapes.

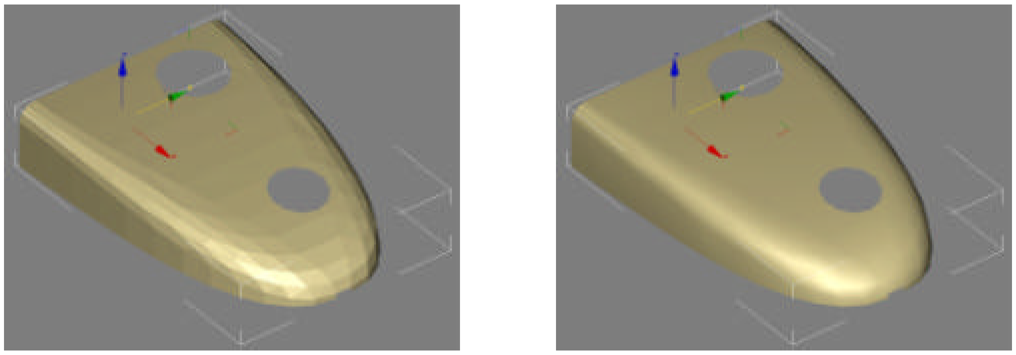


Figure 14. Before and After Using the MeshSmooth Function.

(4) Optimize Function

This function allows users to optimize a shape, this tool will delete automatically all the meshes which are not visible, simplify and delete superfluous points and meshes without greatly altering the shape.

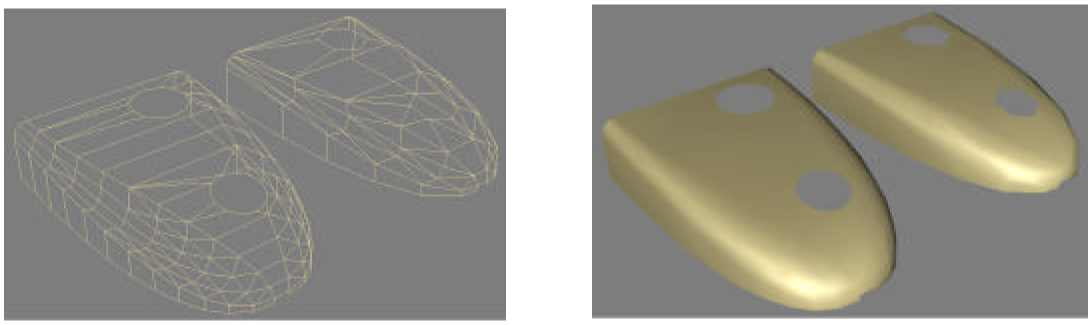
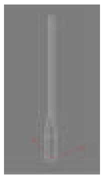
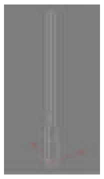
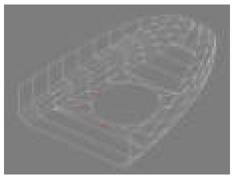
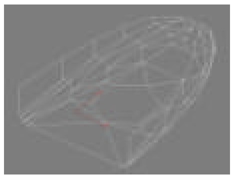
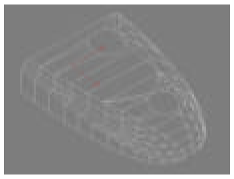
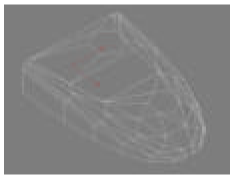
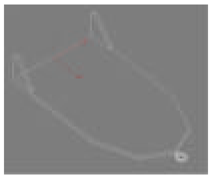
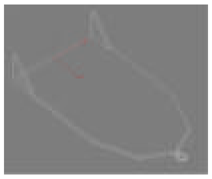
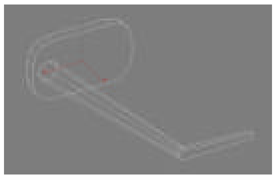
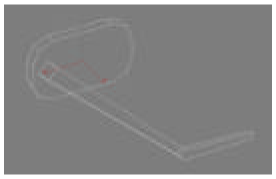
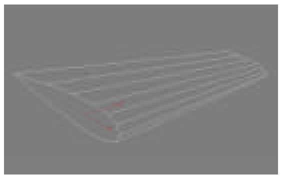
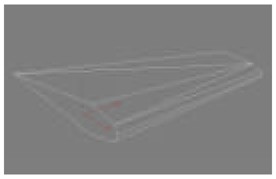
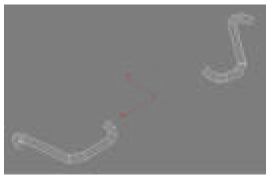
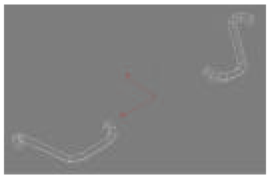
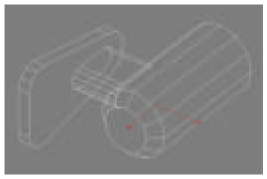
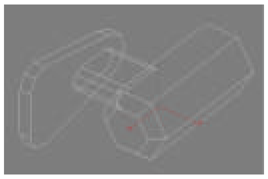


Figure 15. Before Optimization, 159 Points and 260 Meshes (Left). After Optimization, 71 Points and 108 Meshes (Right).

(5) The Two Levels of Detail

In order to optimize and to make the VRML file lighter in calculation, the majority of the elements are designed with two levels of detail. When the user is located close to an element, this element will appear automatically in high level of detail. On the other hand, when the user is located far from an element, an item will appear in a low level of detail. Thus, this method reduces the time of calculation for the processor and makes the navigation faster. Multiple levels of detail are also possible. However, this method makes the VRML file bigger because each element is defined multiple times. We will see how to integrate these levels of detail when importing the elements in X3D-Edit.

The table below shows the list of the elements using two levels of detail:

High Level Of Detail			Low Level Of Detail		
Screen Shot	**Points**	**Faces**	**Screen Shot**	**Points**	**Faces**
	32	54		16	26
	160	262		72	108
	159	260		71	108
	146	276		58	110
	59	110		39	74
	24	44		12	20
	160	290		80	144
	94	184		60	116

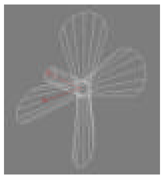
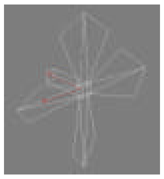
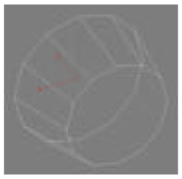
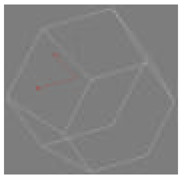
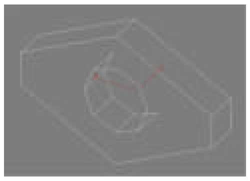
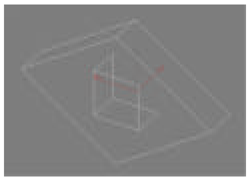
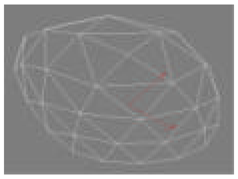
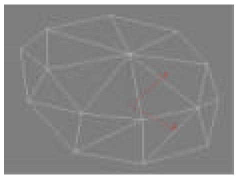
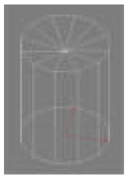
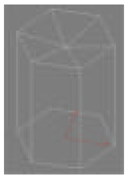
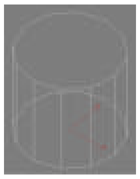
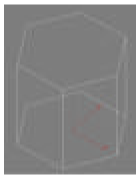
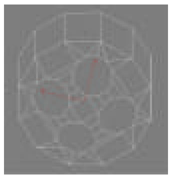
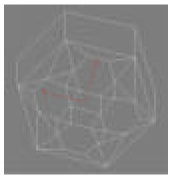
	76	130		28	39
	24	24		12	12
	28	56		16	32
	41	78		18	30
	25	36		13	18
	25	36		12	16
	72	100		36	52

Figure 16. Meshes and Geometry of Aries Model.

4. Integration Using X3D-Edit

a. Organization of the Different Files

In order to increase the legibility of the principal file (AUV.x3d), all elements of the robot submarine were created in different files (they are stored in the X3D folder and VRML files created in VRMLs folder) and are connected to the main file by the relative URL links (thus, it is necessary to not move them). This method offers the advantages of being able to modify an element without modify the main file or, for example, being able to create another AUV by using only necessary elements.

There are two others folders: Maps and Wavs which respectively contain the textures and the sounds used by the main file AUV.x3d.

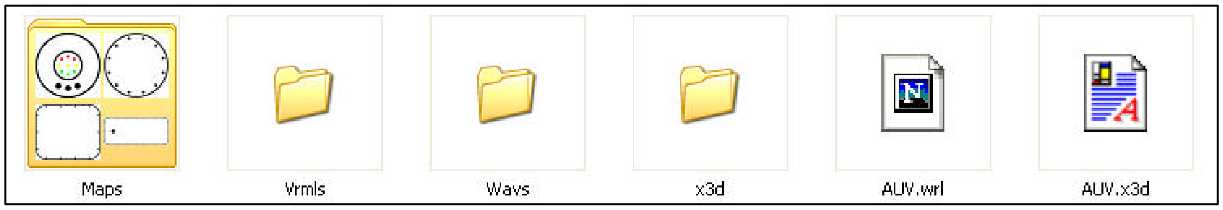


Figure 17. The Different Folders and Files of the Project.

b. How to Import the Shapes from 3dsmax

X3D-Edit has got VRML97 import function which can build the XML schema for X3D-Edit from a VRML file. After, we can copy from this new X3D file the parts which interest us: for example the different lists of coordinates and indexes and paste them in others files.

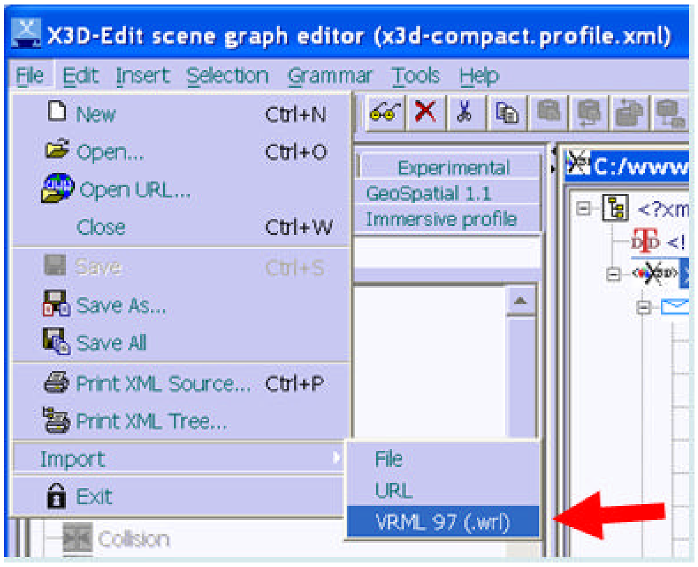
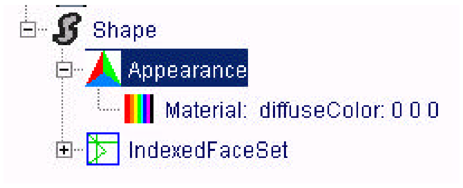


Figure 18. X3D-Edit Import Function.

c. *Material and Mapping*

(1) Material

The nodes Appearance and Material were used to color the elements of the submarine using diffuseColor attribute in RGB values.



Shape
Appearance
Material: diffuseColor: 0 0 0
IndexedFaceSet

Figure 19. Shape, Appearance and Material Nodes.

(2) Mapping

To make the scene more realistic, some materials were put on the submarine. All these maps were design using Photoshop. The used format is Portable Network Graphics (png) because this format allows making transparency effects (for example, here, all the white parts of the maps are transparent; only the colored parts are visible).

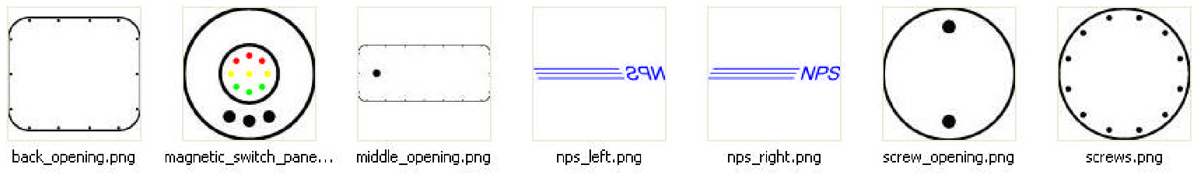


Figure 20. Texture Maps Used for the AUV.

There are several techniques to apply textures on a 3D object in a VRML, such as applying the texture to the polygons but this technique is very complicated when you need to put different textures are needed on the same element. In our case, all the surfaces to be textured are planes, thus we will use a very simple technique. To make the illusion that the texture is mapped on the 3D object, we will create a simple face with a texture mapped on it. This simple face will be positioned very close to the 3D object. As the white is transparent in the PNG file format, we will have the visual impression that the texture is applied directly on the surface and not on another object.

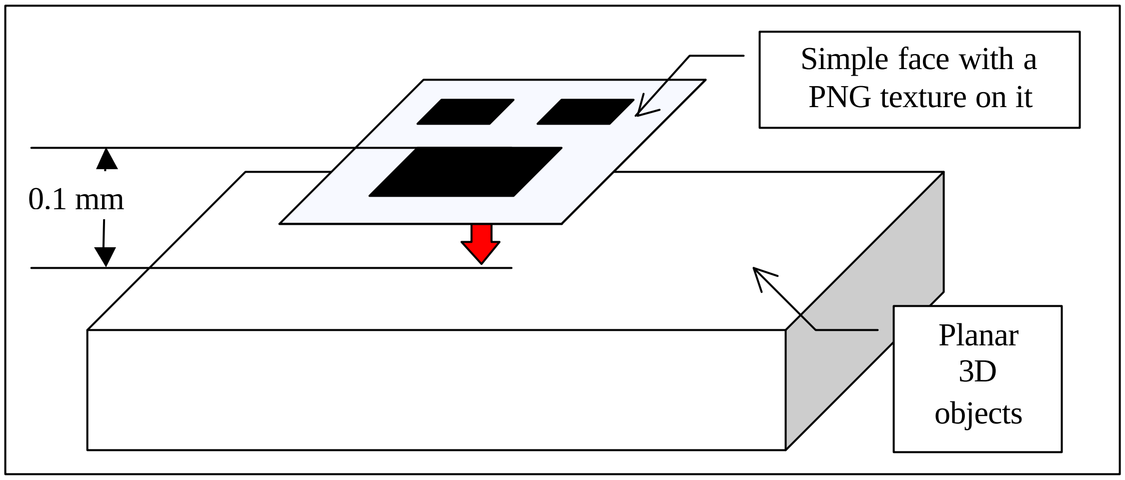
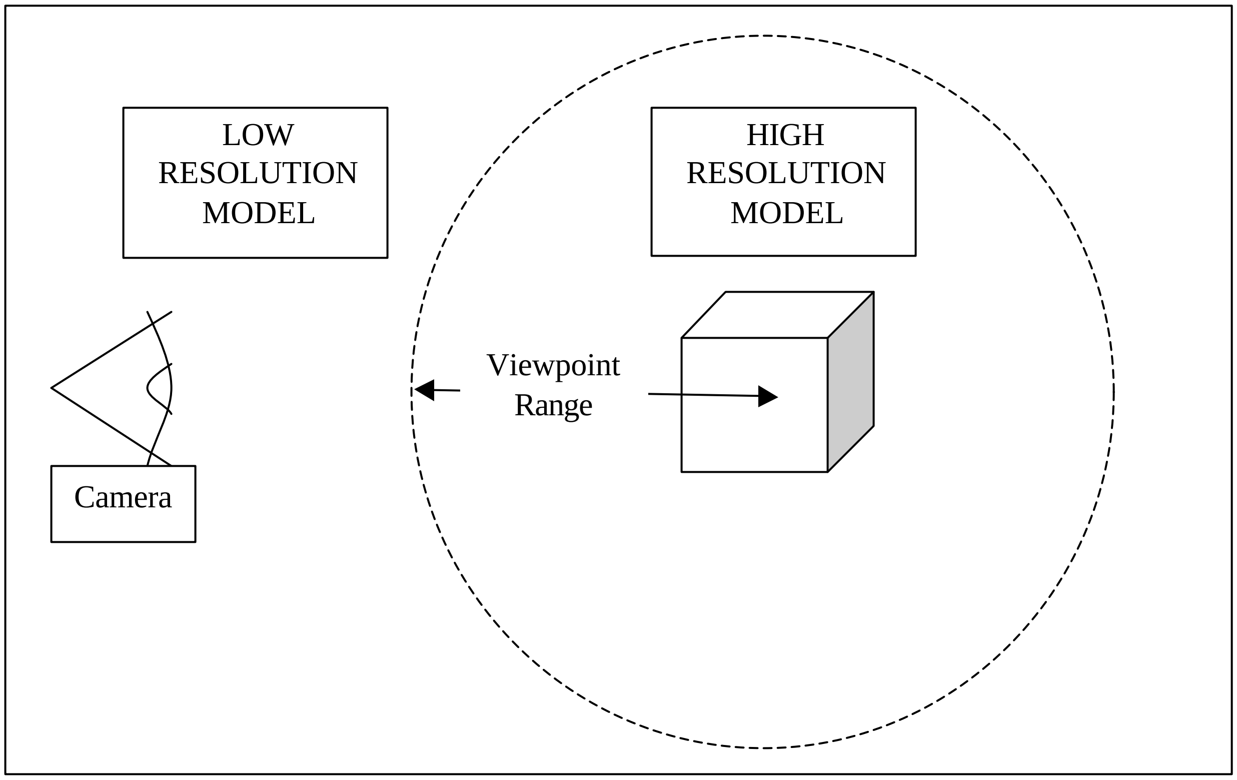
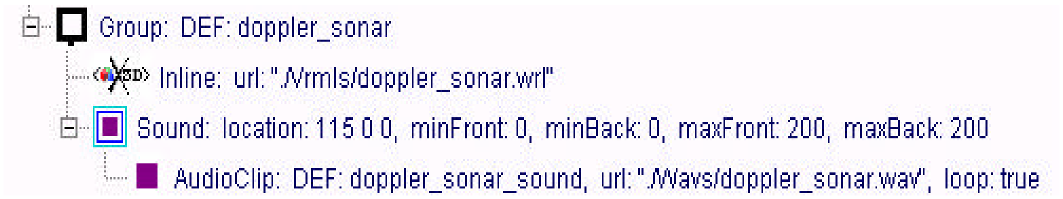


Figure 21. Applying the Textures.

d. The Two Levels of Detail

Objects can have two definitions: one in high level of detail and the other in low level of detail (3.3.2 The two levels of detail). X3D-Edit allows user to define from which distance, between the object and the camera, the object appear in high or low level of detail.




```
Group: DEF: doppler_sonar
    Inline: url: "./Vrmls/doppler_sonar.wrl"
    Sound: location: 115 0 0, minFront: 0, minBack: 0, maxFront: 200, maxBack: 200
        AudioClip: DEF: doppler_sonar_sound, url: "./Wavs/doppler_sonar.wav", loop: true
```

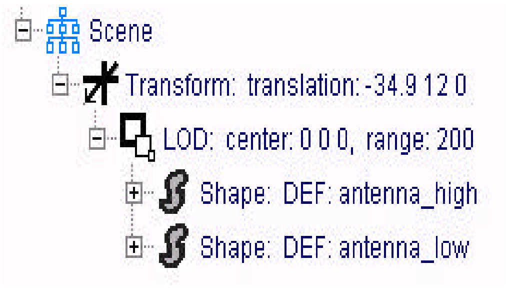

```
Scene
    Transform: translation: -34.9 12 0
        LOD: center: 0 0 0, range: 200
            Shape: DEF: antenna_high
            Shape: DEF: antenna_low
```

Figure 22. High and Low Level Of Detail (LOD)

e. Sounds

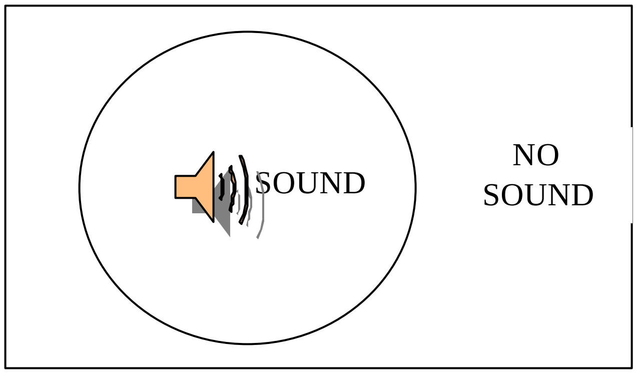
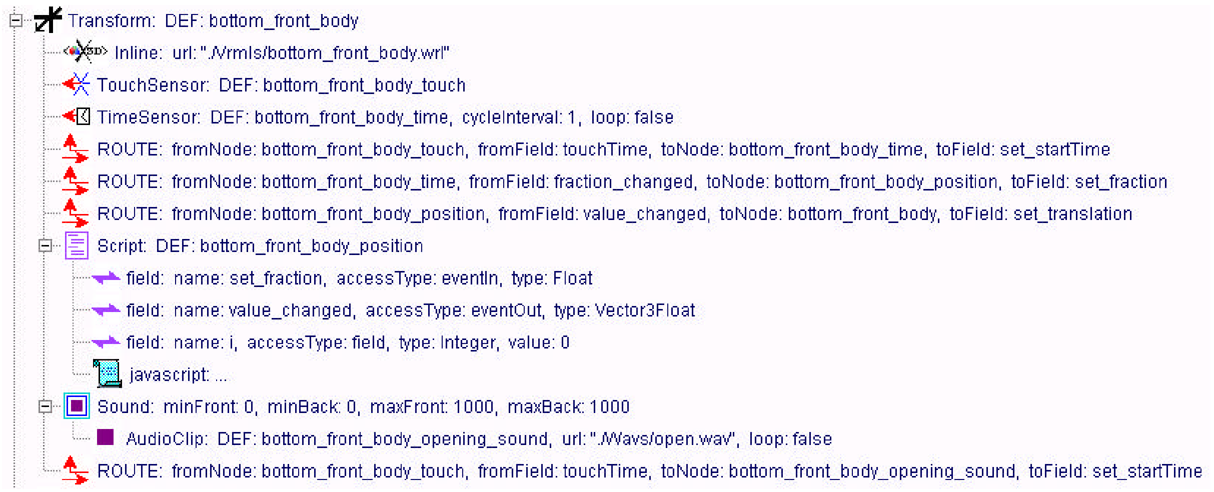


Figure 23. Sound Node is Dependent on Distance to Viewer Location

X3D allows users to place sound source in the scene. Users have to define the location and sound limit. The volume changes according to the position of the camera and makes stereo effects.

f. Animations



```
Transform: DEF: bottom_front_body
    Inline: url: "./Vrmls/bottom_front_body.wrl"
    TouchSensor: DEF: bottom_front_body_touch
    TimeSensor: DEF: bottom_front_body_time, cycleInterval: 1, loop: false
    ROUTE: fromNode: bottom_front_body_touch, fromField: touchTime, toNode: bottom_front_body_time, toField: set_startTime
    ROUTE: fromNode: bottom_front_body_time, fromField: fraction_changed, toNode: bottom_front_body_position, toField: set_fraction
    ROUTE: fromNode: bottom_front_body_position, fromField: value_changed, toNode: bottom_front_body, toField: set_translation
    Script: DEF: bottom_front_body_position
        field: name: set_fraction, accessType: eventIn, type: Float
        field: name: value_changed, accessType: eventOut, type: Vector3Float
        field: name: i, accessType: field, type: Integer, value: 0
        javascript: ...
    Sound: minFront: 0, minBack: 0, maxFront: 1000, maxBack: 1000
        AudioClip: DEF: bottom_front_body_opening_sound, url: "./Wavs/open.wav", loop: false
    ROUTE: fromNode: bottom_front_body_touch, fromField: touchTime, toNode: bottom_front_body_opening_sound, toField: set_startTime
```

Figure 24. Animations

To animate (translate, rotate, etc.) 3D objects, several nodes need to be included in the X3D file:

- TouchSensor: detects mouse interactions with a shape and sends events.

- TimeSensor: specify a timer for the animation.
- ROUTE: define the steps of the animation.
- Script: allows creation of EcmaScript (also known as JavaScript) or Java to make advanced animations.

IV. UNDERWATER VEHICLE HYDRODYNAMICS USING JAVA

A. INTRODUCTION

This chapter is provides an overview of the Software in charge of realized the Hydrodynamics part for an underwater vehicle and in the second time it will explain the documentation of this programs realized using JavaDoc.

B. DYNAMICS ALGORITHM DERIVATION AND IMPLEMENTATION

1. Aim

As explained in Chapter II, the Real-time Modeling of the AUV environment is provided by three main parts named: Execution, Dynamics and 3D Visualization. This chapter describes efforts in hydrodynamics modeling.

The Dynamics program is a substitute for the natural environment's effect on the robot submarine, and provides an estimation of the AUV's behavior in the water. This is a very important and difficult part in the Real-time Simulation in a Virtual World.

Many of the effects of the surrounding environment on a robot vehicle are unique to the underwater domain. Thus, understanding these forces is a key requirement in the development and control of vehicle behavior. This work originally appeared in A Virtual World for an Autono mous Underwater Vehicle [Brutzman, 1992].

2. The Working of the Dynamics Model

In the scope of prior related research many of the hydrodynamics models were investigated for underwater vehicle development, but no single, general vehicle hydrodynamics model was available that proved to be computationally suitable for predicting real-time underwater robot dynamics behavior in a virtual world.

The current hydrodynamics model is based on physical laws and sufficiently accurate to allow the study and development of robust control laws that work under a wide range of potential vehicle motion. Figure 25 explains how the dynamics interact with the others programs. The Execution Program transmits to the dynamics a variety of programs different data composed as the Telemetry vector element shown in Figure 26.

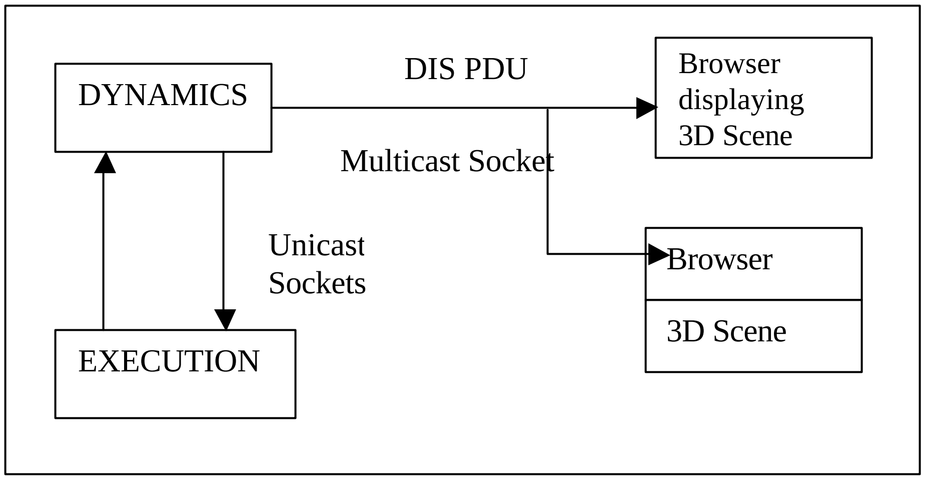


Figure 25. Main Models in the Combined AUV Underwater Virtual World.

Time

X	Y	Z	PHI	THETA	PSI
U	V	W	P	Q	R
X_dot	Y_dot	Z_dot	PHI_dot	PHI_dot	PSI_dot

delta_rudder	delta_planes
propeller_port_rpm	propeller_stbd_rpm
thruster_bow_vertical	thrusters_stern_vertical
thrusters_bow_lateral	thrusters_stern_lateral

ST1000_range	ST1000_Bearing	ST1000_Strength
ST725_range	ST725_Bearing	ST725_Strength

Figure 26. Telemetry State Vector Elements.

3. Hydrodynamics Model Class Hierarchy

The dynamics program was designed to incorporate the principles of object-oriented programming so that it can be easily adapted to other underwater vehicles. As shown in Figure 27, several classes have been created to compute the 3D posture, which is common to all vehicles and can be represented by Euler angles. A class rigid body is subject to Kinematics equations of motion, which combine velocities with postures and update the posture by integrating the velocities.

There also exists a class for DIS-networked rigid body, which communicates with other entities via DIS port communication and using the Protocol Data Unit (PDU). This dynamics program creating for a real-time networked virtual world combines the Dynamics equation of motion and the network.

The other class as UUV Model is composed by the different hydrodynamics coefficients used for the simulation (see 4-2-5).

The boxes in Figure 27 are composed of 4 different compartment boxes. The first compartment is class name. The second compartment indicates member data fields, the third compartment indicates object methods which provide an immediate response, and the fourth compartment includes methods which are time-consuming. This diagramming approach simplifies the presentation but clarifies the hierarchy design.

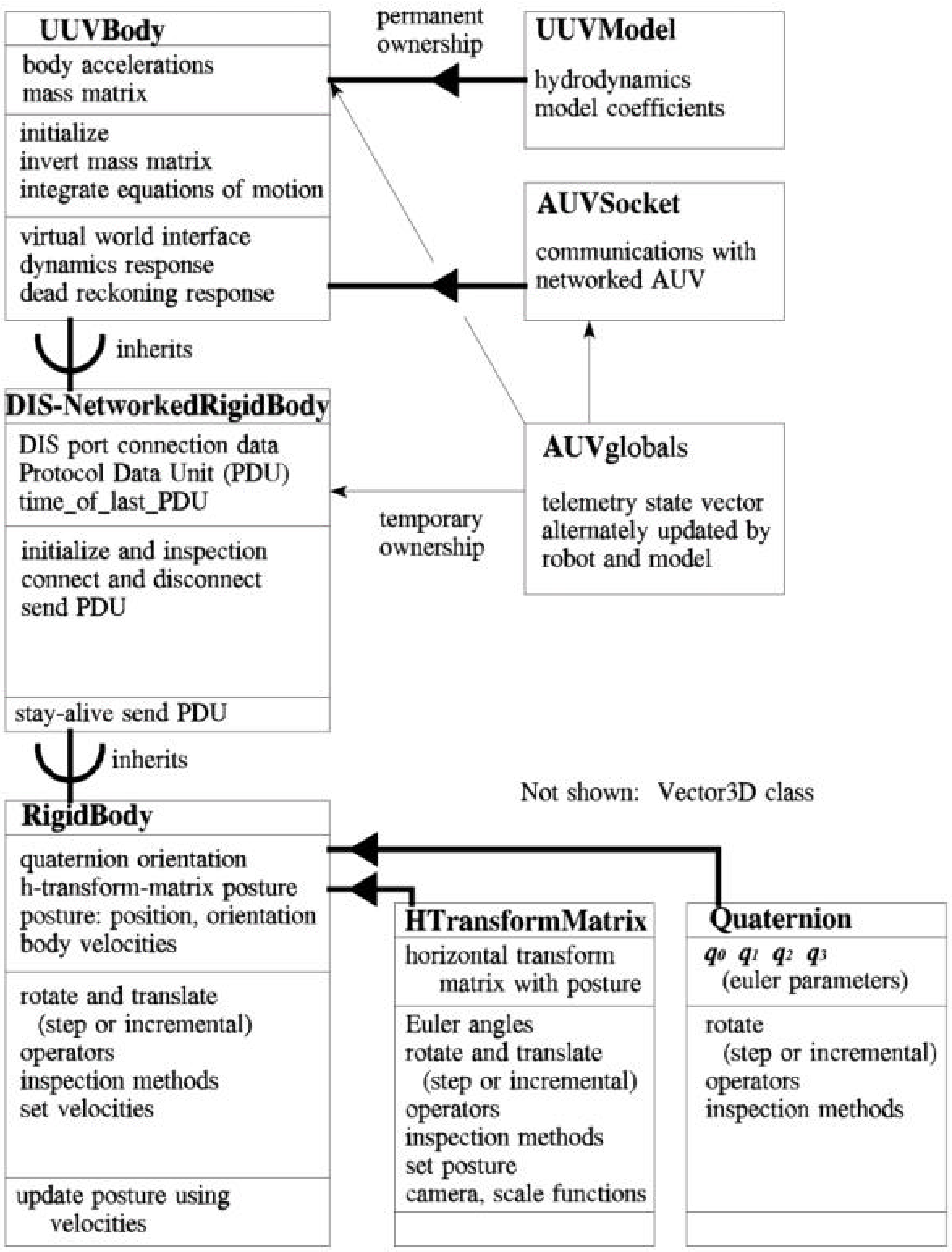


Figure 27. Hierarchy Class for AUV Hydrodynamic Response.

C. EQUATIONS OF MOTION

1. World Coordinates and Body Coordinate

In order to realize the simulation of the submarine in an underwater environment, the equations of motion must first be defined in the body coordinate system as shown in Figure 28.

$$[V\ ]_{Body} = \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} \qquad [V\ ]_{World} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{f} \\ \dot{q} \\ \dot{y} \end{bmatrix} \qquad \textbf{(4.1,4.2)}$$

$$[V]_{Body} = \left[\begin{array}{c|c} [R]^T & 0 \\ \hline 0 & [T]^{-1} \end{array}\right] \qquad \text{[R] and [T] = World Rotation Matrix} \qquad \textbf{(4.3)}$$

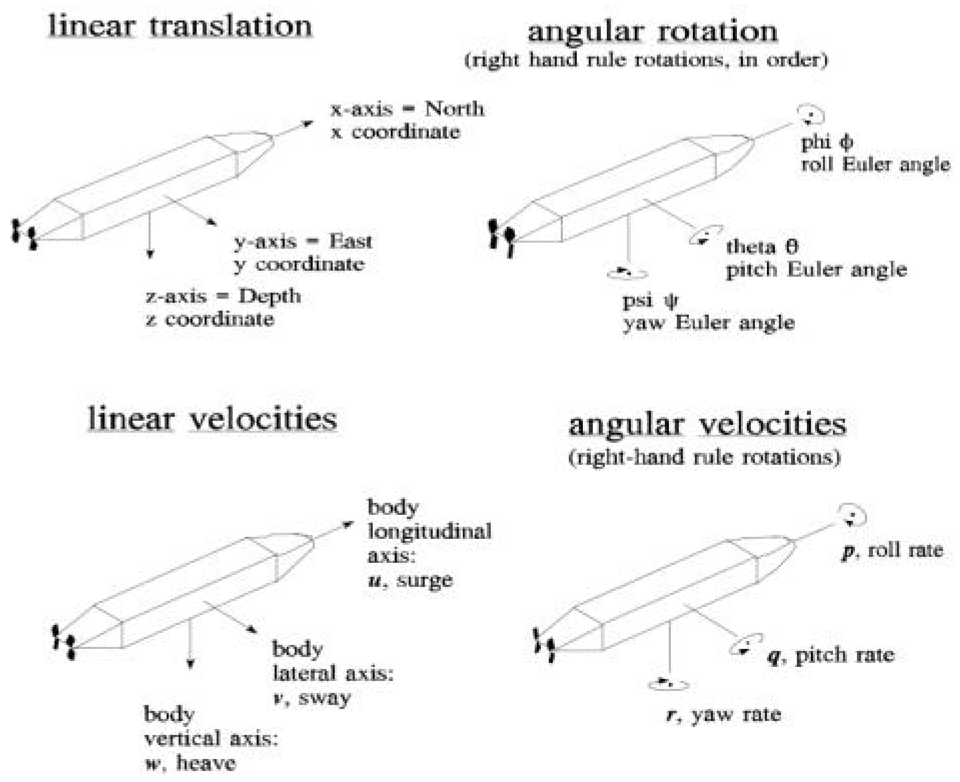


Figure 28. Coordinate System.

2. Force Moments and Acceleration

Force and accelerations for the six state variables of posture can be grouped together in the matrix form of Newton's Second Law:

$$[F] = \frac{d}{dt}[MV] \qquad \textbf{(4.4)}$$

Translational forces are applied at the CG and Moments are similarly applied about the CG origin of the vehicle body. In the aim of finding the mathematical relation between the unknowns of the vehicle state vector and the dynamics equations of motion, this law will be use in the body reference.

$$[F]_{body} = [M]([\dot{V}]_{body} + w \times [V]_{Body}) \qquad \textbf{(4.5)}$$

Within the body coordinate frame the mass matrix [M] is unchanging. Differentiation of the velocity matrix [V] reveals effects that are due to the body coordinate frame rotating with angular velocity w with respect to the world coordinate frame. By multiplied, the equation above, by the matrix $[M]^{-1}$ we have:

$$[\dot{V}]_{Body} = [M]^{-1} \times [F]_{Body} - w \times [V]_{Body} \qquad \textbf{(4.6)}$$

Then we have all the accelerations grouped together on the left-hand side and all the terms on the right-hand sides of the dynamics equations are known at each time step.

3. Velocities and Posture

Knowing the value of the body acceleration allows prediction of the new body velocities through integration.

$$[V]_{body(t0+dt)} = \int_{t0}^{to+dt} [\dot{V}]_{body(t)} dt + [V]_{body(t0)} \qquad \textbf{(4.7)}$$

Integration of the new body velocities to determine posture is preceded by a transformation from the body-fixed coordinate frame to the world coordinate frame.

$$\int_{t0}^{t0+dt} [V]_{world(t0+dt)} dt = \int_{t0}^{t0+dt} \left([V]_{body(t)}\right)_{body \rightarrow world} dt \qquad \textbf{(4.8)}$$

The final integration to determine posture is therefore:

$$[Posture]_{world(t0+dt)} = \int_{t0}^{t0+dt} [V]_{world(t)} dt + \int_{t0}^{t0+dt} [Ocean \cdot currents]_{world(t)} dt + [Posture]_{world(t0)} \qquad \textbf{(4.9)}$$

4. The Form of the Equation of Motion

In many other references, the equations of motion for a submerged vehicle are not usually written in the form suggested above. These other derivations have been presented and structured in such a way that similar time-dependent acceleration-related terms are presented on both sides of the dynamics equations of motion. Because related body acceleration terms are not grouped together, direct time integration of both sides of the equation is not mathematically valid.

This critical point explains why the equations used in the real-time hydrodynamics model have mass-related, inertia-related and acceleration body terms on the left hand side and all the forces like lift, drag, buoyancy, weight, propulsion, etc on the right-hand side. Separation of variables is an essential prerequisite for restructuring the equation of motion.

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}_{(t0)} = \left[M^{-1}\right] \begin{bmatrix} Dynamics \\ equations \\ of \\ motion \end{bmatrix}_{(t0)} \qquad \textbf{(4.10)}$$

Given that the origin of the body-fixed coordinate system is located at the center of buoyancy, the equation of motion for a rigid body in six degrees of freedom defined in terms of body fixed coordinates:

$$m[\dot{u} - vr + wq - x_g(q^2 + r^2) + y_g(pq - \dot{r}) + z_g(pr + \dot{q})] = \sum X_{ext} \quad \textbf{(4.11)}$$

This equation represents translational motion on the x axis. The term on the right side explain the external forces apply on the model on the x axis which are equal at:

$$\sum F_{ext} = F_{hydrostatic} + F_{lift} + F_{drag} + F_{control} \quad \textbf{(4.12)}$$

So for the X axis:

$$\sum X_{ext} = X_{HS} + X_{u|u|}u|u| + X_{\dot{u}}\dot{u} + X_{wq}wq + X_{qq}qq + X_{vr}vr + X_{rr}rr + X_{prop} \quad \textbf{(4.13)}$$

After replacing the right member of the first equation and simplify it to have the acceleration on the left side and the other variables on the right side, the equation of motion is written as follows:

$$\begin{aligned}(m - X_{\dot{u}})\dot{u} + mz_g\dot{q} - my_g\dot{r} &= X_{HS} + X_{u|u|}u|u| + (X_{wq} - m)wq + (X_{qq} - m)wq + (X_{qq} + mx_g)q^2 \\ &+ (X_{vr} + m)vr + (X_{rr} + mx_g)r^2 - my_g pq - mz_g pr + X_{prop} \quad \textbf{(4.14)}\end{aligned}$$

Numerous different coefficients are used to predict all the external Force and moments (see Chapter 4-2-5). Finally, the body frame velocity matrix can now be updated by numerical integration, such as Euler Methods.

$$[V]_{(t0+dt)} = dt \bullet [\dot{V}]_{(to)} + [V]_{(to)} \quad \textbf{(4.15)}$$

5. Variables and Coefficients

With the variety of different hydrodynamics, models, studies have produced numerous huge sets of several coefficient libraries. This is a serious problem for newcomers to hydrodynamics literature, since both names and definitions of key terms may vary. So it’s important to describe coefficients using a well-defined nomenclature, which in the AUV case corresponds to the standard reference work on ship control (Lewis 88).

For example

$$\sum X_{ext} = X_{HS} + X_{u|u|}u|u| + X_{\dot{u}}\dot{u} + X_{wq}wq + X_{qq}qq + X_{vr}vr + X_{rr}rr + X_{prop} \quad \textbf{(4.16)}$$

- XHS explain the combined between weight and buoyancy.
- Xuu explain the Cross-flow Drag
- $X\dot{u}$ explain the Added Mass
- X_{wq} , X_{qq}, X_{vr} , X_{rr} explain the added Mass Cross-term
- X_{prop} the propeller Thrust

6. Hydrodynamics Model Algorithm

Now that the different parts of the general underwater vehicle real-time hydrodynamics model have been presented, the following section explains the algorithm used:

- Estimate and invert mass matrix [M]
- Initialize hydrodynamics model variables for posture [P], velocities [V] and time rates of change of velocities.

Loop until robot is done:

- Receive updated state vector from robot, including ordered effectors values for rudders, planes, propeller, thrusters and elapsed time.
- Calculate new values for time rate of change of body velocities, using the current vehicle state vector and equation of motion.
- Update velocity [V].
- Perform transformation to [V] coordinates.
- Update posture [P]using newly-calculated velocities [V] world ocean current estimate and previous posture.
- Return newly-calculated hydrodynamics values to robot via telemetry update of the robot state vector. Most calculated velocities and accelerations correspond to real-world values provided by inertial, flow and pressure sensor.
- Wait for next updated state robot vector,
- shutdown when model is no longer required by robot.

7. JavaDoc

a. Introduction

The Java (SDK) comes with a number of development tools, including:

- A compiler, which translates source code into Java byte code.
- The JVM (Java Virtual Machine), which runs Java Programs once they have been compiled.
- A tool called JavaDoc, which reads Java source code and creates HTML files that document the corresponding source files.

This section explains how JavaDoc works and the production of JavaDoc for dynamics program.

b. JavaDoc Commenting Convention

When the Java compiler translates source code (java files) into binary byte code (class files), it ignores all comments. In other words, any text in the source code that is delimited by a "/**" at the beginning and a "*/" at the end, or lines beginning with "//" are interpreted as as a JavaDoc comment.

Comments in Java can be categorized as either programmer's comments or JavaDoc comments. The former comment on internal matters of implementation and are designed to aid other programmers who might, for example, be maintaining the code in the future. There are two kinds of programmer's comments:

- single-line comments that are delimited by "//"
- multiline comments that are delimited by "/*" and "*/".

For example:

```
/**
  * File : AUVglobal.java is a JavaDoc comment
  */
public class AUVglobals
{
// not a JavaDoc comment since there is no
asterix //followed by a slant
```

Figure 29. Comments in JavaDoc.

JavaDoc extends this rule comparably by enabling special documentation comments to appear within text sections bracketed by “/**” and “*/”.

JavaDoc comments document the functionality of a class or its API and are converted into HTML files to be read by people who do not necessarily need to read the source code of the HTML elements. Besides the delimiters, JavaDoc comments can employ HTML tags to identify different components that need to be organized within the documentation.

For example, the result of

Figure 29 will appear in JavaDoc HTML file as in Figure 30:

CLASS AUVGLOBALS

```
      java.lang.Object
      |
      +--AUVglobals
```

---

public class **AUVglobals**
extends java.lang.Object

File: AUVglobals.java is a JavaDoc comment

Figure 30. Javadoc Results.

c. Other JavaDoc Conventions

Additional commenting conventions can be used by JavaDoc; such as “@return”, “@author”, “@version” which generate documentation appropriate for the comments. For example, in the code listing in

Figure 31, the “@param” tag is used to document the parameter `amount` which is part of the withdraw method’s signature. The HTML generated by this JavaDoc is shown in Figure 32.

```
/**
  * Subtracts the specified amount from the
  * customer's account balance.
  * @param amount the amount to withdraw
  */

public void withdraw(Dollar amount) {
// code left out in this example
```

Figure 31. Javadoc Markup for Tags.

```
<A NAME="withdraw(Dollar)"></A>
<H3>withdraw</H3>
<PRE>
public void <B>withdraw</B>(<A
HREF="Dollar.html">Dollar</A>&nbsp;amount)
</PRE>
<DL>
<DD>Subtracts the specified amount from the
customer's account balance.
<DD>
<DL>
<DT><B>Parameters:</B><DD><CODE>amount</CODE>
- the amount to withdraw</DL>
</DD>
</DL>
<HR>
```

Figure 32. HTML Version.

Another possibility for generating JavaDoc is to write directly in the HTML format within the JavaDoc comments, will appear directly in the output.

Author: Don Brutzman (*web.nps.navy.mil/~brutzman*)
Revised: 6 March 1977 - converted to Java by Kevin Byrne and Jeff Schmidt 20 February 1998 - Updated by Kevin Byrne

AUV telemetry state vector Note these are global for direct access by any world model. Refer to individual world models for details. Data hiding within a private object is not necessary since all values are transient and superseded by actual state when it occurs. Additionally, half of the state variables are provided only by the AUV microprocessor socket, and the other half are provided by respective world models. Thus global variables in this design are not vulnerable to corruption and side effects, making data hiding unnecessary.

Source Code: AUVglobals.java

Figure 33. HTML Codes in JavaDoc Source.

```
/**
*<dt> Author: Don Brutzman (<A
HREF="http://web.nps.navy.mil/~brutzman"><i>web.nps.navy.mi
l/~brutzman</i></A>)
*<p>
*<dt>Revised:         6 March 1977 - converted to Java by
Kevin Byrne and Jeff Schmidt
*                20 February 1998 - Updated by Kevin Byrne
*<p>
*<dt> AUV telemetry state vector
*           <dd>Note these are global for direct access by
any world model.  Refer to
*           <dd>individual world models for details.  Data
hiding within a private object
*           <dd>are not vulnerable to corruption and side
effects, making data hiding
*          <dd>unnecessary.
*<dt><b>Source Code:</b>
*<dd><a href="AUVglobals.java">AUVglobals.java</a>
*/
```

Figure 34. JavaDoc Results.

d. Why JavaDoc?

A software vendor does not usually sell or include the source code with the executable binary files. The JavaDoc comments document the functionality of a class or its members. Once the source code has been compiled, all that remains that is humanly readable is the API and the functionality which the API provides access to. The binary code is packaged in a JAR file and is sold to the customer along with the API documentation.

For example, a software vendor may design and implement a class library that may be used by a third party to develop customer relationship management software (CRM). There would be classes that represent customers, contact information, service tickets, and so on. To use these classes to develop a CRM application, each of the classes would be documented using JavaDoc comments. After the source code was processed by JavaDoc, HTML documents would be generated and shipped with the binary code.

Technical writers who are going to document APIs write JavaDoc comments in the source code and then use the JavaDoc tool to turn those comments into HTML files. The source code that implements the functionality of a class and the documentation that explains that functionality exist in one place: the Java source file. This has become so popular, that JavaDoc-like tools have been developed for C++ APIs, also.

Practice has shown that inclusion of JavaDoc comments is fundamentally important for long-term maintenance of a source base.

D. CONCLUSIONS

Comments have been added to the different programs of the 'real-time dynamics model' using the tool JavaDoc. These comments allow a better interpretation and information for the future user by describing the different part of the programs, the functions, and the variables and coefficients interacting inside the Dynamics model.

The other utility of the comments is to inform the user, like a standard documentation, about different subject as the date, the author, the version and also the aim of the programs. By giving this information the modification and comprehension of the 'real-time dynamics model' will be easier.

V. AUV MISSION VISUALIZATION WORKBENCH

A. INTRODUCTION

This chapter describes how the AUV simulation works, and the new Java integrated development environment for simulation and development of the AUV source code

B. REQUIREMENTS

This graphic explains the different steps of the simulation and the different windows (Text editor, DOS windows and Netscape Navigator) used to calculate and display the simulation.

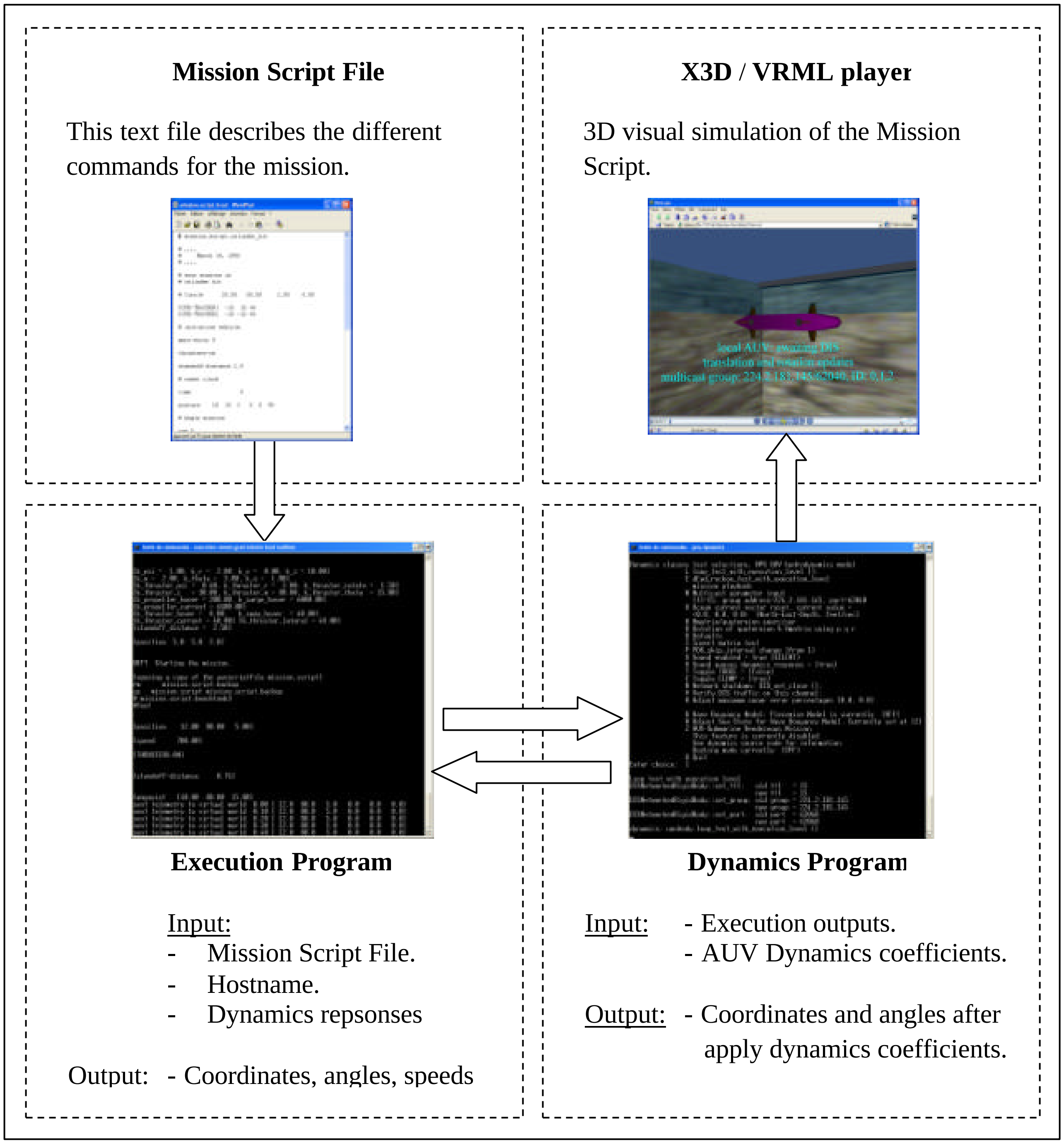

Mission Script File
This text file describes the different commands for the mission.
X3D / VRML player
3D visual simulation of the Mission Script.
local AUV: awaiting DIS translation and rotation updates multicast group: 224.8.185.145:62040, ID: 0,1,2
Execution Program
Input:
- Mission Script File.
- Hostname.
- Dynamics repsonses
Output: - Coordinates, angles, speeds
Dynamics Program
Input: - Execution outputs.
- AUV Dynamics coefficients.
Output: - Coordinates and angles after apply dynamics coefficients.

Figure 35. Interrelationships in the AUV Workbench.

1. The New Interface for the Simulation

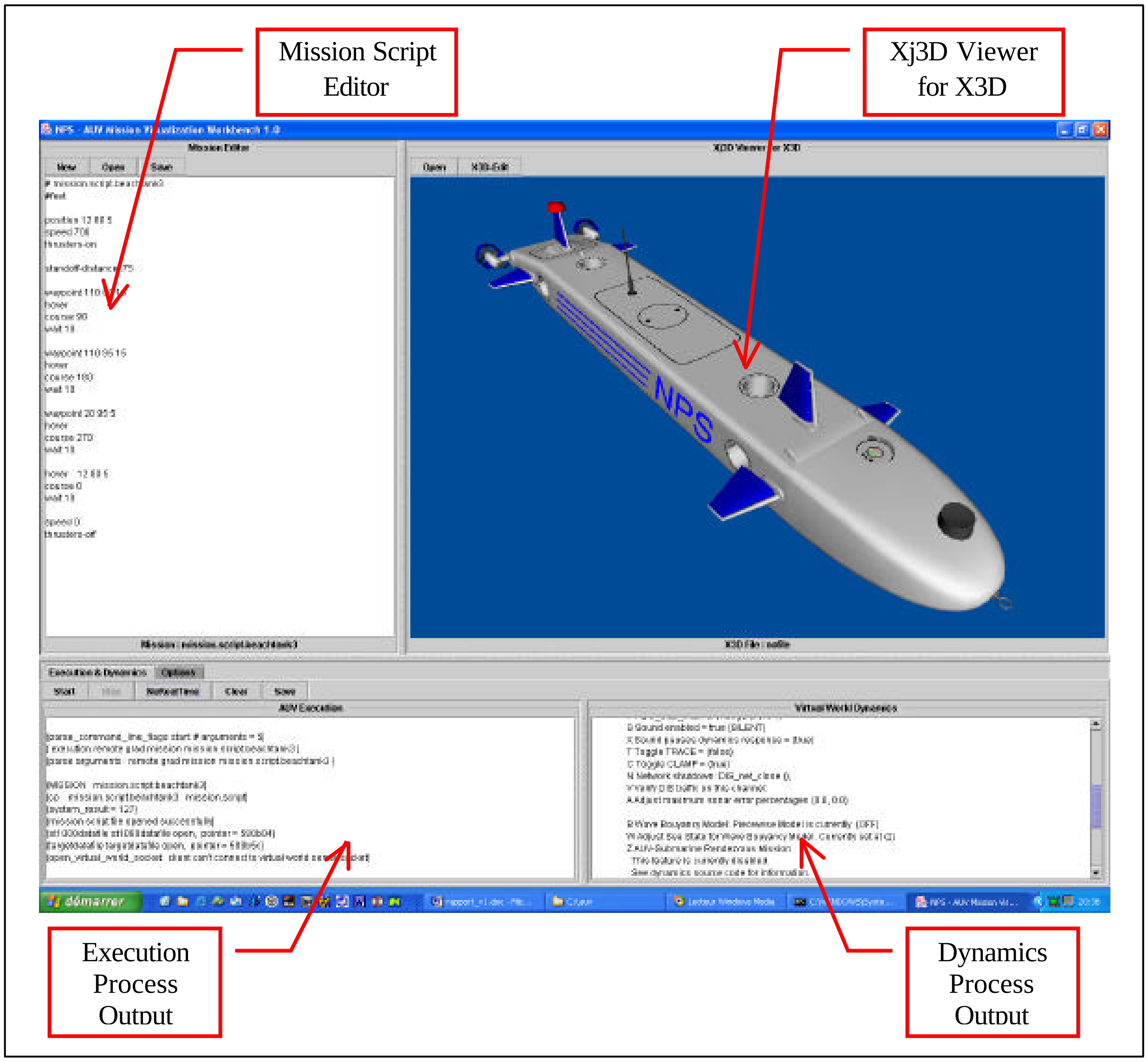
a. Why a New Interface?

This software will be use by the scripts developers and by the thesis students to test and edit their simulations. It will also allow for the pre-visualization of in-water missions. The new interface was designed to:

- Simplify and make more easily the utilization of the simulation.
- Have all the windows in one main window.
- Allow scripts developers to edit and test their scripts more quickly.
- Allow AUV software developers to evaluate execution and dynamics improvements.

b. Presentation of the New Interface

Figure 36 shows the newly produced integrated development interface.



```
# mission.script.beachtank3
#test

position 12 88 5
speed 708
thrusters-on

standoff-distance 75

waypoint 110 88 16
hover
course 90
wait 18

waypoint 110 95 16
hover
course 180
wait 18

waypoint 20 95 5
hover
course 270
wait 18

hover 12 88 5
course 0
wait 18

speed 0
thrusters-off
```

```
[parse_command_line_flags start # arguments = 3]
[ execution remote gradmission mission.script.beachtank3 ]
[parse arguments: remote gradmission mission.script.beachtank3 ]

[MISSION mission.script.beachtank3]
[cp mission.script.beachtank3 mission.script]
[system_result = 127]
[mission script file opened successfully]
[auv000.data file auv000.data file open, pointer = 500b04]
[targetdata file target data file open, pointer = 500b5c]
[open_virtual_world_socket client can't connect to virtual world server socket]
```

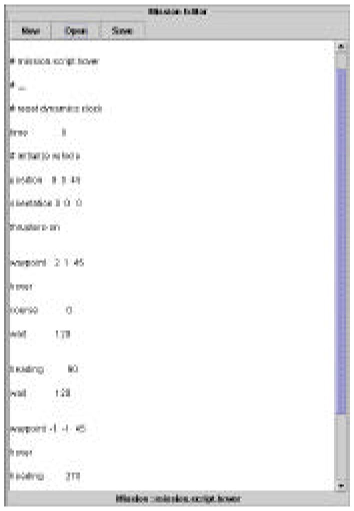
```
B Sound enabled = true (SILENT)
X Sound pauses dynamics response = (true)
T Toggle TRACE = (false)
C Toggle CLAMP = (true)
N Network shutdown: DIS_net_close ();
V Verify DIS traffic on this channel
A Adjust maximum sonar error percentages (0.0, 0.0)

B Wave Buoyancy Model. Recurrence Model is currently (OFF)
W Adjust Sea State for Wave Buoyancy Model. Currently set at (2)
Z AUV-Submarine Rendezvous Mission
  This feature is currently disabled.
  See dynamics source code for information.
```

Figure 36. Interface of the AUV Mission Visualization Workbench.

2. Details of the Different Parts of the Software

a. The Mission Script Editor



```
# mission script hover
# ...
# reset dynamics clock
time        0
# [illegible]
position   0 0 45
orientation 0 0 0
thrusters on

waypoint   2 1 45
hover
course       0
wait        120

heading      90
wait        120

waypoint -1 -1 45
hover
heading     270
```

Figure 37. The Mission Script Editor Panel.

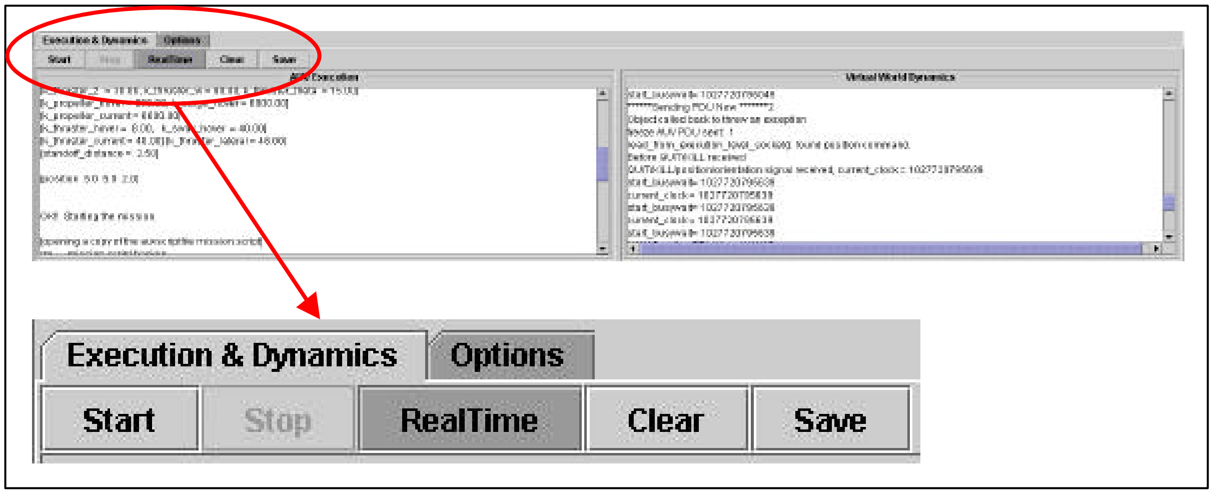
The Mission Script Editor is a text editor with which users can create, open or save missions.

When a mission is opened, the software automatically creates a backup of this file (with the name of the mission and the date). So, the user can reopen this file in case of a modification error.

The mission opened will be the mission used for the simulation.

b. Execution and Dynamics Process

(1) Execution & Dynamics Panel



```
[illegible] = 18.00, [illegible] = 18.00, [illegible] = 15.00]
[k_propeller_[illegible] = [illegible] = 8800.00]
[k_propeller_current = 8800.00]
[k_thruster_hover = 8.00, k_[illegible]_hover = 40.00]
[k_thruster_current = 48.00][k_thruster_lateral = 48.00]
[standoff_distance = 2.50]

[position 5.0 5.0 2.0]

[OK: Starting the mission

[opening a copy of the [illegible] of the mission script]
[illegible]
```

```
start_buswait = 1027720795048
*****Sending PDU New *******2
Object called back to throw an exception
freeze AUV PDU sent: 1
read_from_execution_level_socket() found position command.
Before QUITKILL received
QUITKILL/position/orientation signal received, current_clock = 1027720795628
start_buswait = 1027720795628
current_clock = 1027720795638
start_buswait = 1027720795628
current_clock = 1027720795638
start_buswait = 1027720795638
```

Figure 38. Execution and Dynamics Panel.

- **Start & Stop**: launch or stop the simulation of the Mission File in the Mission Editor.
- **RealTime**: this toggle button allows users to display the simulation in real-time or not.
- **Clear**: this button allows users to clear the execution and dynamics text area.
- **Save**: this button allows users to save the execution and dynamics text area in two different files:
    - MissionName_Execution_Date
    - MissionName_Dynamics_Date

(2) Options Panel

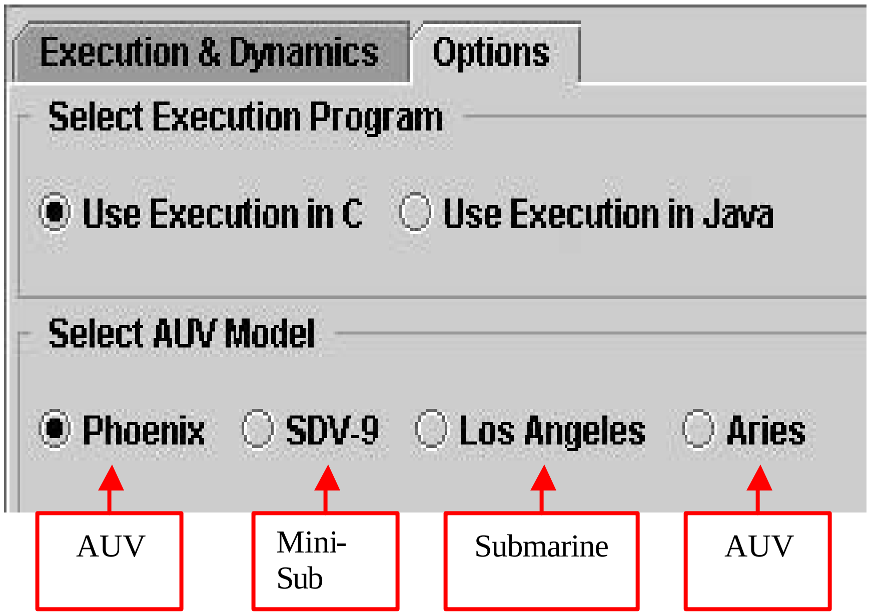


Figure 39. Options Panel.

- **Select Execution Program**: there are two different programs for the execution level (one in C and the other in Java), so the users can select which one to use.
- **Select AUV Model**: there are four different AUVs which each have different dynamics coefficients, so the users can select which coefficients they want to use for the simulation.

(3) How Does the Simulation Process Work?

The Java language allows developers to execute others programs from a Java program. Here, there are two different Threads (one for Execution and one for Dynamics) which have launch the programs, catch the output streams, and print them in the two text areas in the main window.

c. *The Xj3D Viewer for X3D*

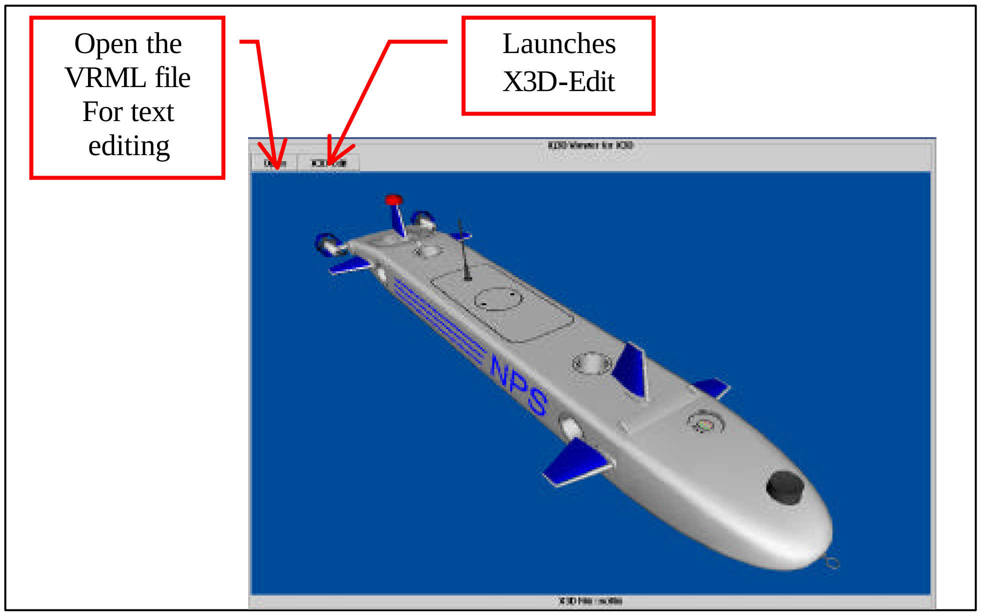


Figure 40. Xj3D Viewer for X3D.

Xj3D uses all the specification of X3D to be able to display VRML file in a program using Java3D.

However, Xj3D is currently under development, so not all of the X3D nodes are integrated in Xj3D (Billboard for example); thus, it is necessary for the users to download and install the latest version of the Xj3D package to update the Java classes which are used by the program. With time, Xj3D is expected to become fully stable, since it is an open-source project.

C. HOW TO INSTALL AND RUN THE PROGRAM

To use this software, users need to setup their computer and install some additional Java programs and packages used by the program:

- Download and install Java Runtime on the computer: http://java.sun.com/j2se
- Download and install Xj3D-full-Mx.exe which will install Java3D and Xj3D Java Class on your computer: http://www.web3d.org/TaskGroups/source/Xj3D.html

Then unzip AMVW_1.0.zip from the report CD into the c:\auv directory. You need to have all these files and folders:

- c:\auv\dynamics\
- c:\auv\execution\
- c:\auv\bin\
- c:\auv\AMVW_1.0.bat

Also recommended are X3D-Edit, the X3D examples, and the SAVAGE model archive.

To execute the program, double click on AMVW_1.0.bat.

D. CONCLUSIONS AND FUTURE WORK

This project has produced a useful tool for AUV mission visualization and software development.

To make the interface more convivial and customizable by different users, future work should consider other layouts, make them savable and selectable like the example is Figure 41.

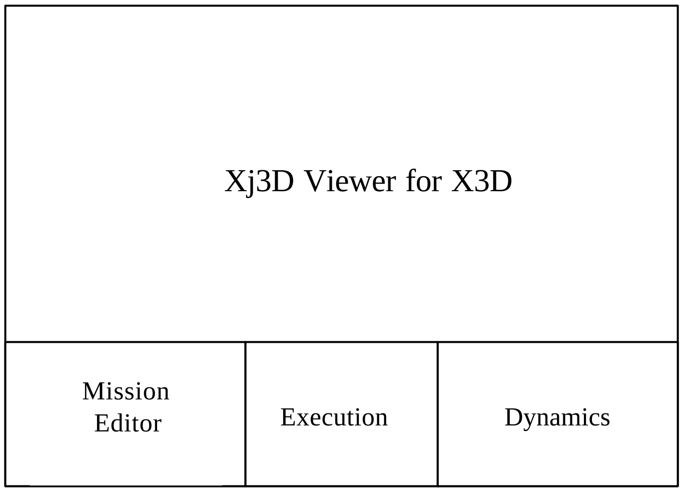


Figure 41. Future Work: Example of a Customized Layout.

VI. AUV DYNAMICS CONTROL WORKBENCH ON MATLAB

This section describes how the Dynamics Model was realized using MATLAB and the SimuLink tool. It further describes the realization of an interface using the tool of Java and MATLAB Guide. It builds on significant work originally produced by Olivier Doucy [October 2000].

A. THE MATLAB DYNAMICS MODEL

1. Overview

This part of the report focuses on the simulation tool for an AUV such as Phoenix (Figure 42) that is easily used to assess vehicle systems design and characteristic. This product is based on traditional standard tools of AUV developers control-system interfaced with a real-time 3D virtual reality representation.

To provide a simulation environment for AUV design and assessment the software has been created using MATLAB and its temporal domain simulation environment SimuLink. This is a common software development environment for mechanical engineering.

The 3D representation has been realized under VRML 97 (Virtual Reality Modeling language) (Brutzman) as a networked viewer. The use of open standards for networking (DIS) and 3D representation (VRML) always allows for running on different platforms.

This section explains the existing SimuLink model, interfacing the virtual world, additional options, and the graphical interface.

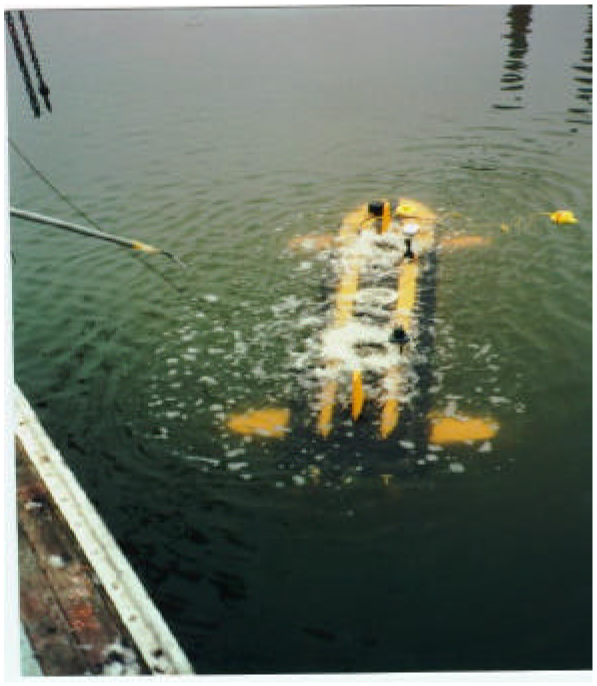


Figure 42. ***AUV Phoenix.***

2. The SimuLink Model

In order to realize a high-resolution dynamics simulation of the AUV with real-time response, the physical behavior of the AUV had to be based on the NPS dynamic model (Healey, see 4-2).

This model can be easily generalized to any AUV when the hydrodynamic coefficients which appear in the equations of motion have been modified.

The architecture of the actual dynamic model of the vehicle has been designed using a modular approach allowing easy addition and replacement of functionalities (Figure 43). This simplification has been achieved through six main modules.

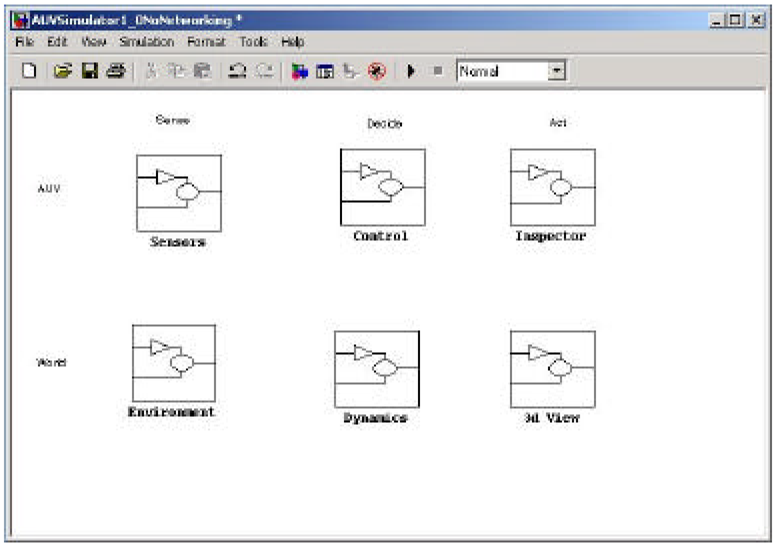


Figure 43. SimuLink Model.

Dynamics: This system embeds the dynamic behavior of the AUV. The model is based on the model developed at NPS by Anthony Healey and modified by Jeffrey Riedel for station keeping in waves. The Dynamics model includes the simulation of the forces applied based on environment simulation and control inputs. The output of the dynamic model is the complete vehicle dynamic state. Included in the vehicle state are the actuators related states. Inputs include actuator commands provided by the control function and relevant output of Environment.

Environment: This function embeds the description of the environment. Its output is contained in Goto Blocks that are needed as inputs by the dynamic model function, the measurement function and control function.

Sensors: Contains the embedded sensors and different level of model can be introduced.

Control: This function includes the simulation of Control scheme (including estimation) applied to the vehicle. For Input the user may use directly the simulated dynamic state or the simulated measurement. The choice will depend on the design stage and the design methods used. The output will mainly be actuators command or direct efforts if needed by the user.

Inspector: this function includes 2D viewers, dynamic displays of the simulation relevant variables and simulation results Storage. As a basis the most important variables, based on the existence of To/From blocks have their display. The user is free to add needed displays. Viewing functions have to be kept in Inspector to avoid loosing clarity of the model in general.

3D View: Those blocks provide the interface to the 3D real-time virtual world viewer. The behavior of the vehicle can be seeing in the 3D virtual world.

3. Integration of the Equations and Variables

The Dynamics blocks are the main block of the simulation, in fact all the equation of motion are integrate in this part inside function call S-Function. An S-Function is a MATLAB tool easy to integrate in the SimuLink model and in this case created in the C language. So different sub-programs as sfun_AUV_Hydrodynamics.dll or sfun_BuoyancyEfforts.dll have been created and dialogue with the SimuLink model thought different block parameters. All these functions are independent, can be use in another model, created on visual-C++ and compile with MATLAB as a MEX-File.

The value variables are type on an M-file as 'DynamicsParameters.m' or 'EnvironmentParameters.m' and then integrate in the Workspace when the program is lunch.

4. Test Results

Results are shown in the inspector block. Different information about the commands of the fins and thrusters, the buoyancy and thruster effort and also the vector of the position and velocity of the submarine are shown on different "scope" output plots.

The other option in this model is the visualization of the submarine in the 3D virtual world. This visualization result from a dialogue between the output of the SimuLink model sending the entity vector through the m-file 'sfun_to_ESPDU' and the "WaveAUV.wrl" file which receives the data and then translates the AUV model in the virtual world.

5. Interfacing to the Virtual World

A 3D-virtual real-time representation is available for the analysis of the AUV behavior and is realized by the networking module. SimuLink integrates the Java version of the DIS protocol and sends the ESPDU (Entity State PDU) which includes linear and rotational values for posture, velocity and acceleration. So the DIS details the message sent over the network by distributed components which describe the dynamic behavior of entities present in the virtual world. The model is then "driven" around in the VRML scene.

6. The Graphical Interface

a. GUI Development Environment

The creation of a interface with MATLAB can be realized by different possible approaches. In fact MATLAB is able to import a library (or even class objects) coming from Java or C. MATLAB also has its own environment to create a interface: **GUIDE**. This is the preferred approach for building native MATLAB interfaces.

GUIDE, MATLAB's Graphical User Interface Development Environment provides a set of tools for laying out a Graphical Interface. MATLAB includes a Layout Editor which is the control panel for GUIDE. The Figure 44 shows the layout Editor.

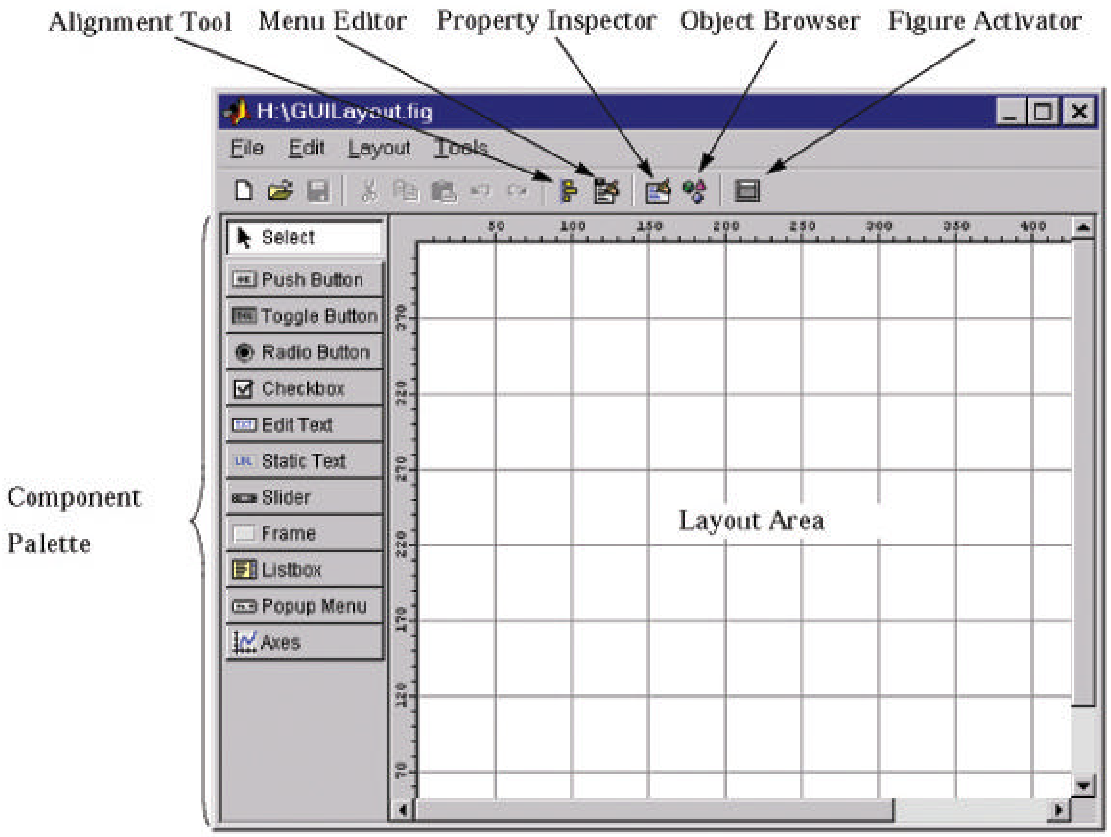


Figure 44. Models Layout Editor.

Each GUI display object must be programmed to perform the intended action when activated by the user of the GUIDE.

GUIDE generates a set of layout tools as well as an M-File that contains code to handle the initialization and launching of the GUI. This M-File provides a framework for the implementation of the function callbacks, which execute the functions that the user activates in the GUI.

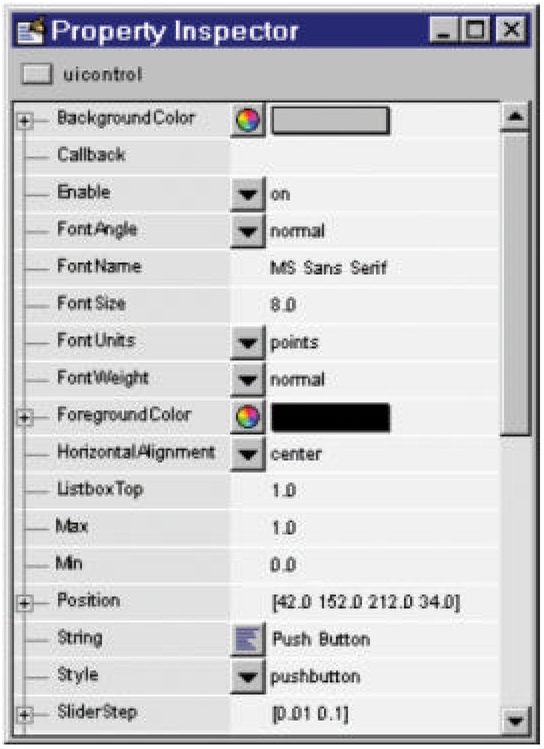
There are two file that save and launch the GUI:

- A FIG: file contains a complete description of the GUI figure and of all its children, as well as the values of all object properties.
- An M-file: contain the functions that launch and control the GUI and the callbacks.

b. MATLAB Tools

The tools are similar to other GUI interface tool, for example, in Java Swing. The codes generated are different even if the concept is the same.

Each component, like the Push Button and Checkboxes or Edit Text, has properties that you can set with the property inspector Figure 45.



BackgroundColor	⬜
Callback	
Enable	on
FontAngle	normal
FontName	MS Sans Serif
FontSize	8.0
FontUnits	points
FontWeight	normal
ForegroundColor	⬛
HorizontalAlignment	center
ListboxTop	1.0
Max	1.0
Min	0.0
Position	[42.0 152.0 212.0 34.0]
String	Push Button
Style	pushbutton
SliderStep	[0.01 0.1]

Figure 45. Property Inspector.

Moreover each component has automatically a Callback sub function in the application M-file created. This sub function created will house all the codes that will be activate when the component is:

The syntax of the Callback Function is:

function varargout = *objectTag*_Callback(h,eventdata,handles,varargin)

The arguments are listed in the following table:

Callback Function Arguments	
	The handle of the object whose callback is executing.
`eventdata`	Empty, reserved for future use.
`handles`	A structure containing the handles of all components in the GUI whose fieldnames are defined by the object's `Tag` property. Can also be used to pass data to other callback functions or the main program.
`varargin`	A variable-length list of arguments that you want to pass to the callback function.

Figure 46. Callback Function Arguments

This is the main tool used for the realization of a graphical interface on MATLAB.

c. The AUV Dynamics Control Workbench Functionality

The AUV Dynamics Control Workbench has been created to allow an easier utilization of the MATLAB AUV Dynamics model. This interface has been realized with MATLAB and Java. The future customer of this interface will be the students and research workers who design and evaluate control code and hydrodynamics models.

Students gain an overview and introduction to AUV dynamics modeling and can experiment with the different parameters and coefficients that interact in the dynamics models.

The research workers will be able to test the Dynamics model, to change it, and also to modify the different parameters, evaluate effectiveness and store the results.

d. *Description of the AUV Dynamics Control Workbench*

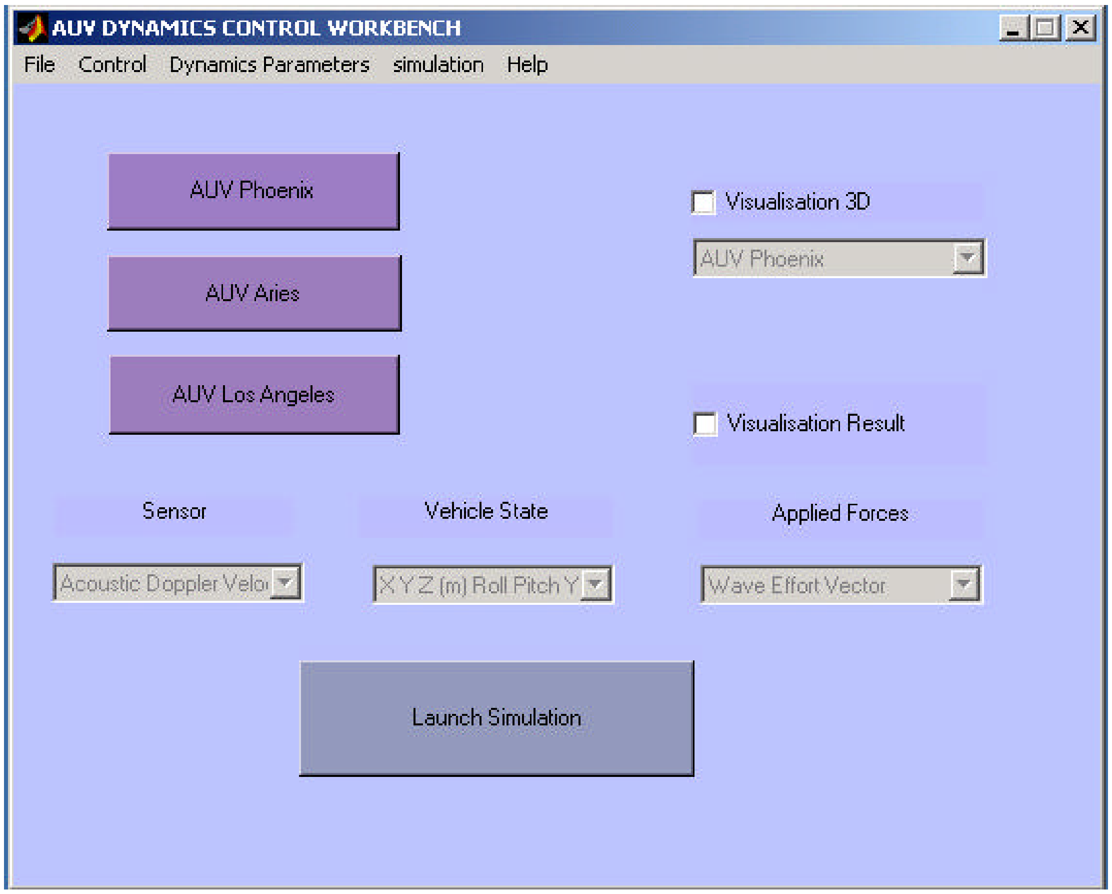


Figure 47. Main Windows.

The main window shown in Figure 47 contains 5 parts: Menu bar, 3D Visualization, Models, Results and Launch Simulation.

- **Menu Bar:** The Menu bar is composed by 5 Menus items the first one call File allow to save data resulting from the simulation, to open new data store in the pass and to close the programs. The sub-menu control allow the modification of the control parameters, The other menu dynamics can be use for modified and read all the coefficient interacting in the simulation, the simulation menu is just a short cut to the two SimuLink model (if 3d visualization or not) and the help menu is an access to an help about the AUV dynamics SimuLink model and MATLAB.
- **Models:** Models is composed of three push buttons which load the dynamics coefficients of the submersible choose in the workspace and then set the SimuLink model for the right AUV.

- **3D Visualization**: this option is use to see the evolution of the submersible on 3D VRML. The model can be choosing using the popup menu.
- **Results:** by using the different popup menu you can see the different results of the simulation like the Vehicle States of the submersible (velocity, position, etc.) the forces applied on it (wave effort, buoyancy efforts, etc.)
- **Launch Simulation:** starts the simulation after having choosen all the different options or change the different parameters to lunch the simulation.

Other Windows accessing from the main windows have been created to allow an easier modification, entrance, interpretation of this SimuLink models.

7. Modifications

a. AUV ARIES

The AUV ARIES is the newest submarine in action presented in the introduction, it geometry is clothe to the AUV Phoenix which was the one before. The coefficients of this submersible have been integrating in the MATLAB dynamics model and the equations of motion are the same that used for the AUV Phoenix. The model used for the 3D Visualisation has also been added to the existent AUV PHOENIX 3D Visualisation.

b. REMUS

The REMUS (Remote Environmental Monitoring Unit) is a low-cost, modular vehicle with applications in autonomous docking, long-range oceanographic survey, and shallow-waters mine reconnaissance (Figure 48). [WHOI]

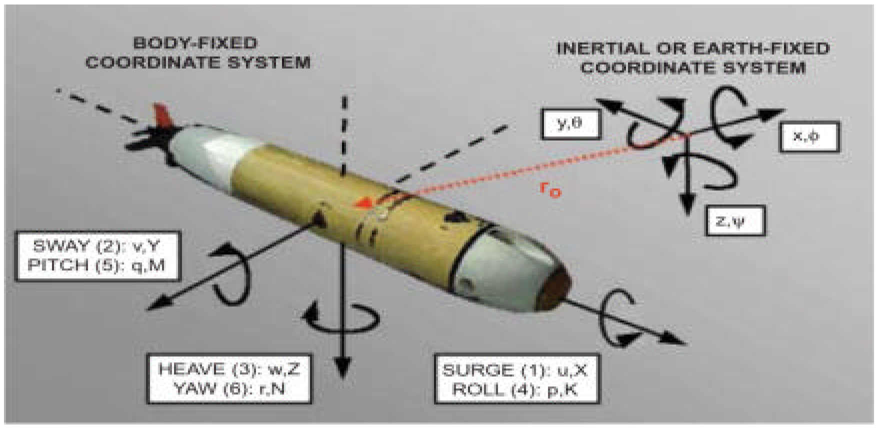


Figure 48. Remus AUV [WHOI].

The REMUS is different from the ARIES Submersible, the attitude of the REMUS vehicle is controlled by two horizontal fins, or stern planes, and two vertical fins or rudder, the propulsion is realise by only one thrusters and the geometry is different from the other Submersible. The integration of this underwater vehicle in the existing SimuLink model have been realised using the thesis of Timothy Presto (Verification of a six-degree of freedom Simulation Model for the REMUS Autonomous Underwater Vehicle) which store the different coefficients and profile that needs to be use for this model.

After testing the model in simulation, it seems that the behaviour of the submersible is not correct. Because of time restraints, the resolution of this problem must be deferred as future work. Either invalid coefficients or mismatched nomenclature are the likely causes of this failure.

B. CONCLUSIONS AND FUTURE WORK

This work was done with the goal of improving the existing model by allowing easier utilisation and interpretation of AUV control-law response and hydrodynamics modelling. The tool was further extended to include REMUS hydrodynamics. Unfortunately, the REMUS model coefficients are unstable, or else the hydrodynamics algorithm has a previously undiscovered flaw. Further work is needed to complete this important functionality.

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Many of the major challenges in operating an Autonomous Underwater Vehicle are solved. Robust, capable systems are being deployed in both experimental and operational contexts. But critical failures in control, software and systems still plague even the best AUVs. The thrust of the work presented here has been to make a useable, virtual world complete with hydrodynamics modelling, actual robotics software and realistic models to demonstrate behaviour indicated by the simulation. These pieces and parts have existed separately for many years, relative to the age of the technology in general. Coalescing them into a single tool has finally been accomplished.

By no means is this work finished. There are exciting possibilities in every direction: improved dynamics simulations, user reconfiguration of the GUI on demand, documentation, full integration with MATLAB, and integration of other simulations and models.

B. RECOMMENDATIONS FOR FUTURE WORK

As with all technical endeavours, documentation is the key to passing on knowledge and continuing the work. The JavaDoc, while valuable, is only a partial solution. A concerted effort should be made to complete the documentation to include: User’s Manual: Software Version Description Document that details updates, bug fixes and enhancements; Software Developer’s Manual; etc.

It will be beneficial to compare other dynamics models and simulations. Just one hydrodynamics model, albeit a good one, has been employed here. Other models would allow users to configure the tool to suit a variety of control algorithms and response characteristics.

The Inspector function of the MATLAB dynamics model allows a view of the different forces apply on the submersible. It also demonstrates the behaviour under different specific missions and controls, more precisely targeting the different parameters interacting in the models. Integrating this MATLAB function into the workbench would have benefit.

This work stands on the shoulders of giants in the field of autonomous underwater robots. It raises the bar of available functionality, so that exciting and interesting work continues.

LIST OF REFERENCES

Access from Different Existing Work from the Home Page of Don Brutzman: [http://web.nps.navy.mil/~brutzman/]

Brutzman, Donald P., “A Virtual World For An Autonomous Underwater Vehicle”, December 1994, Dissertation, Naval Postgraduate School.

Doucy ,Olivier, “Manoeuvring and Station-Keeping for an Autonomous Underwater Vehicle”, October 2000, Dissertation, NATO Symposium.

General Documentation about the JavaDoc Documentation Available on Java Tool Home Page: [http://java.sun.com/j2se/javadoc]

Healy, Anthony J., “Dynamics and Control for AUV’s”, May 2002, Presentation, Naval Postgraduate School.

Help for Creating a General Graphical User Interface on MATLAB (GUIDE): [http://www.mathworks.com/access/helpdesk/help/techdoc/creating_guis/creating_guis.shtml]

Help for Using and Calling Java from MATLAB: [http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_external/ch_java.shtml]

Marco, David B, and Healy, Anthony J., “Current Developments in Underwater Vehicle Control and Navigation: The NPS ARIES AUV”, 2001, Dissertation, Naval Postgraduate School.

Prestero, Timothy, “Verification of a Six Degree of Freedom Simulation Model for the Remus Autonomous Underwater Vehicle”, September 2001, Thesis, University of California at Davis.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Thierry Vidal
   Ecole Nationale d'Ingénieurs de Tarbes, France
   thierry@enit.fr

4. Didier Leandri
   University of Toulon, France
   leandri@univ-tln.fr

5. Cristina Russo Dos Santos
   University of Toulon, France
   cristina.russo@eurecom.fr

6. Tom Swean
   Office of Naval Research
   SWEANT@onr.navy.mil

7. Jean-Pierre LeGoff
   Société d’Ingénierie, de Recherches et d’Etudes en Hydrodynamique Navale, Nantes, France
   mailto:sirenha@sirenha.ec-nantes.fr

8. Olivier Doucy
   Société d’Ingénierie, de Recherches et d’Etudes en Hydrodynamique Navale, Nantes, France
   Olivier.Doucy@sirehna.ec-nantes.fr

9. Eric Chaum
   Naval Undersea Warfare Center
   ChaumE@npt.nuwc.navy.mil

10. Ernie Drew
    Sonalyst, Incorporated
    ewd@sonalysts.com

11. Margaret Bailey
Sonalyst, Incorporated
bailey_m@sonalysts.com

12. Dr. Don Brutzman
Naval Postgraduate School
brutzman@nps.navy.mil

13. Dr. Tony Healey
Naval Postgraduate School
healey@nps.navy.mil

14. CDR Bill Marr, USN
Naval Postgraduate School
wjmarr@nps.navy.mil

15. Doug Horner
Naval Postgraduate School
dphorner@nps.navy.mil

16. Jeffrey Weekley
Naval Postgraduate School
jdweekle@monterey.nps.navy.mil

17. Alastair Cormack
Seabyte, LLC
alastair.cormack@seebyte.com