Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology, Durham, NH, August 2005

THE AUTONOMOUS UNMANNED VEHICLE WORKBENCH: MISSION PLANNING, MISSION REHEARSAL, AND MISSION REPLAY TOOL FOR PHYSICS-BASED X3D VISUALIZATION

Duane T. Davis, Naval Postgraduate School
1 University Circle
Watkins Annex Room 265
Monterey, CA 93943
(831) 656-3733
dtdavis@nps.navy.mil

Don Brutzman, Naval Postgraduate School
1 University Circle, Code USW/Br
Watkins Annex Room 270
Monterey, CA 93940
brutzman@nps.navy.mil

ABSTRACT

In recent years, numerous military and civilian applications for autonomous underwater vehicles have been identified or proposed and a number of production and research vehicles have been developed to address many of these. However, the use of vehicle-specific data formats and mission planning systems hampers cooperative research efforts by fostering the implementation of stove-pipe systems. Additionally, the lack of physics-based playback and tightly coupled two- and three-dimensional (2D/3D) visualization environments typically precludes realistic mission rehearsal or high-fidelity playback.

The Naval Postgraduate School (NPS) Autonomous Unmanned Vehicle Workbench (AUVW) bridges the gap between dissimilar vehicles by providing a mission planning, rehearsal, and replay tool for arbitrary unmanned vehicles (UV). The AUVW is designed around the capabilities of the Autonomous Vehicle Control Language (AVCL), an Extensible Markup Language (XML) vocabulary for task-level mission specification, vehicle telemetry, control orders and sensor data. Also integral to the AUVW is physical simulation of planned missions in a virtual environment (VE) using a six degree of freedom (6-DOF) vehicle-response model that is parameterized for a variety of vehicles.

The vehicle-independent nature of AVCL, enables the AUVW user to design and test missions for arbitrary vehicles using a single format. Once satisfactory performance is obtained in the virtual environment (VE), the AVCL mission is transformed to vehicle-specific format using the Extensible Stylesheet Language for Transformations (XSLT) and loaded into the vehicle. In similar fashion, vehicle-specific data can be parsed using a context free grammar (CFG) and transformed into AVCL for editing and testing with the AUVW.

3D visualization utilizes a variety of archived and autogenerated scenes authored in Extensible 3D (X3D) graphics, the ISO-approved standard for 3D graphics on the web and the Xj3D open-source toolkit for X3D-compliant application development.

1. INTRODUCTION

Numerous military and civilian uses for autonomous vehicles (UV) have been identified or proposed in recent years. Not surprisingly, a number of UV systems have become available and research is ongoing in a number of areas that will significantly advance the state of the art in UV technology. A shortcoming exists, however, that will inhibit the implementation of systems that adequately address the majority of these potential applications. The fact of the matter is that vehicle-specific data formats and mission planning systems preclude effective coordination in multi-vehicle systems and hinder the design of such systems. Additionally, the nonavailability of a system for mission planning and monitoring for dissimilar UVs makes it difficult to develop and execute multi-vehicle plans for anything but homogeneous systems (i.e., multi-vehicle systems consisting of only one type of vehicle).

Current research at the Naval Postgraduate School (NPS) Center for Autonomous Underwater Vehicle (AUV) Research is addressing the paucity of vehicle-independent UV-planning systems through the development of the Autonomous Unmanned Vehicle Workbench (AUVW) [15]. The AUVW is a Java application for air, ground, surface and underwater UV mission planning, rehearsal and playback. Features include a geographically synchronized two-dimensional (2D) graphical user interface (GUI) for mission development and editing, physics in the loop mission rehearsal using six degree of freedom (6-DOF) models, 2D and three-dimensional (3D) visualization of mission progress, import and export of vehicle-specific data, and networked communication between the AUVW and vehicles before, during and after mission execution.

AUVW efforts utilize the Java Look + Feel (L+F) guidelines to achieve cross-platform compatibility. Menu tooltips, hotkeys and a consistent graphical user interface (GUI) permit a coherent approach to combined 2D and 3D displays. Additionally, the JavaHelp system has been utilized to provide extensive documentation, linked in a context-sensitive manner throughout the workbench toolbars and buttons.

The remainder of this paper will be cover specific details of the AUVW. Section 2 will briefly discuss the Autonomous Vehicle Control Language (AVCL), an Extensible Markup Language (XML) [22] tagset for UV mission definition, inter-vehicle communications and mission results data. Section 3 will provide a description of the mission planning functionality of the AUVW. Section 4 consists of an overview of the AUVW mission-rehearsal capability. Section topics include the implementation of physically based vehicle models and their use to model arbitrary vehicles in a common VE, and the use of Extensible 3D (X3D) graphics [14] and the Distributed Interactive Simulation (DIS) protocol [13] to support mission visualization in a VE. Section 5 will discuss AUVW facilities for support of specific vehicles. This section will cover the import of vehicle-specific data for editing and visualization in the AUVW, the production of vehicle-specific code from AUVW-generated AVCL files, and AUVW-vehicle communications supporting pre-, post-, and in-mission interaction with actual vehicles and operators.

2. AUTONOMOUS VEHICLE COMMAND LANGUAGE (AVCL)

Supporting the implementation of a planning system for arbitrary UVs is an ontology defined by AVCL—vocabulary, word meanings, assumptions and relationships sufficient to describe a subset of UV operations [6]. This common data format and corresponding utilities for the automatic conversion of data in this format to and from vehicle-specific formats serves as a bridge between dissimilar autonomous vehicles. AVCL facilitates coordinated operations between dissimilar vehicles and enables their human operators to interact with dissimilar vehicles during all operational phases—planning, execution, and post-mission analysis. As the data format utilized by the AUVW, AVCL enables the development of tasking for and interaction with numerous UVs regardless of their specific data formats.

For reasons described in [10], AVCL is being developed as a schema-governed XML vocabulary. XML has a number of advantages that make it well-suited to this role. An XML document can be easily understood and processed by both humans and computers. This simplifies the operator interface for generating and editing documents when compared with binary data formats and simplifies computer processing of documents when compared to non-XML, text-based formats.

More importantly, strict definition of allowable document content and structure using an XML schema [23][24] or data type document (DTD) [22] provides a number of advantages over alternative formats. First is programmatic verification of document correctness and validity. This enables the planning system to be implemented in such a way as to detect and possibly correct errors in loaded documents and preclude the generation of invalid documents. Another advantage of schema-governed XML is that the rigorously defined structure facilitates data mapping to and from vehicle-specific data formats. Ultimately, this directly supports automated conversions to and from vehicle-specific data formats using Extensible Stylesheet Language for Transformations (XSLT) and context free grammars (CFG) as described in section 5. Further, the XML schema serves as the basis for compression algorithms that make the transfer of XML data over noisy and bandwidth-limited communications paths feasible [19].

A final advantage to the use of schema-governed XML takes the form of XML data binding. The AUVW utilizes the Java Architecture for XML Data Binding (JAXB) [20] to automatically generate an application programmer's interface (API) specific to the AVCL schema. Since the API is specific to the type of document being processed, JAXB allows the manipulation of data-bound objects whose schema-compliance is programmatically maintained. On the other hand, while simpler parsers such as the Document Object Model (DOM) [25] and Simple API for XML (SAX) [16] can be used to process and manipulate XML documents they can be cumbersome and error prone because schema-compliance must be explicitly maintained by the programmer and cannot be assumed.

AVCL attempts to define an UV operational ontology consisting of three parts: mission definition, mission results, and communications. The mission results and communications portions of the schema are still being developed and are not yet utilized to a large degree by the AUVW (although the telemetry data included in the mission results portion of the ontology is used for playback of previously executed missions). Mission definition, on the other hand, is utilized extensively for mission planning in the AUVW and will therefore comprise the bulk of the AVCL-specific discussion of this paper.

AVCL provides two means of defining an UV mission. The first uses a sequence of script or task-level commands (task-level AUV commands are listed in Table 1) that are executed in order. Some commands are used to update vehicle control parameters such as commanded depth and closed-loop-control timestep and require only a single vehicle execution loop to complete. Others, such as waypoint commands, control vehicle execution until some commanded end state (e.g. geographic position) is achieved. In addition to the set of task-level commands, AVCL task-level scripts can contain meta commands. These will normally be used for vehicle-specific information or comments, and will not directly influence how specific vehicles might execute the specified mission. They may, however, effect how the AVCL script is translated to a vehicle-specific format. For example, an AVCL meta command can be used to capture the Hydroid Remote Environmental Measuring Units (REMUS) AUV navigation modes that are not directly representable with AVCL task-level commands. Conversion of a mission with such a meta command to a format for any vehicle other than REMUS will ignore the meta command, but conversion to the REMUS command

COMMAND	DESCRIPTION
CompositeWaypoint	Specifies a pattern consisting of multiple waypoints
GpsFix	Surface for a GPS fix while continuing the current control mode
Hover	Transit to a specified position and maintain a fixed position (requires hover-capable vehicle)
Loiter	Proceed to a specified position and remain in the vicinity
MakeAltitude	Maintain a specific altitude above the bottom while continuing the current control mode
MakeDepth	Maintain a specific depth while continuing the current control mode
MakeHeading	Set commanded vehicle heading
MakeSpeed	Set commanded forward speed
MissionScript	Load and execute a new script
MoveLateral	Slide laterally (requires a body thruster equipped vehicle)
MoveRotate	Rotate in place (requires a body thruster equipped vehicle
Position	Updates vehicle geographic position
Quit	Mission ended
ResetTime	Reset the vehicle time
SendMessage	Transmit a message
SetPlanes	Manually set vehicle control planes
SetPropeller	Manually set vehicle propellers
SetRudder	Manually set vehicle rudder
SetThruster	Manually set vehicle body thrusters
Standoff	Set the acceptable distance from waypoints
TakeStation	Hover relative to an external object (requires hover-capable vehicle)
Thrusters	Enable or disable body thrusters
TimeStep	Reset vehicle control closed loop timestep
Wait	Wait a specified time period before proceeding to next command
WaitUntilTime	Wait until a specified time before proceeding to next command
Waypoint	Transit to a specified position

Table 1: Available Autonomous Vehicle Control Language (AVCL) task-level commands for autonomous underwater vehicle (AUV) use.

language [12] will utilize it to set the correct navigation mode.

The second method of specifying an AVCL mission is as a declarative goal-based mission. Missions specified in this manner consist of a set of constraints (avoid areas and required ingress and egress routing) and a set of goals in the form of a finite state machine (FSM) where each goal to be accomplished (e.g. an area search or rendezvous with another UV). Individual goals are represented as states in the FSM with transitions executed upon success or failure of a goal. A declarative mission completes when the current goal succeeds (or fails) and there is no further applicable transition, such as would be the case if the "SampleEnvironment" goal depicted in Figures 1 and 2 failed. Upon reaching the end of the mission defined by the FSM, the egress routing specified by the mission constraints (or a direct routing back to the start point if no egress routing is specified) is executed prior to vehicle shutdown.

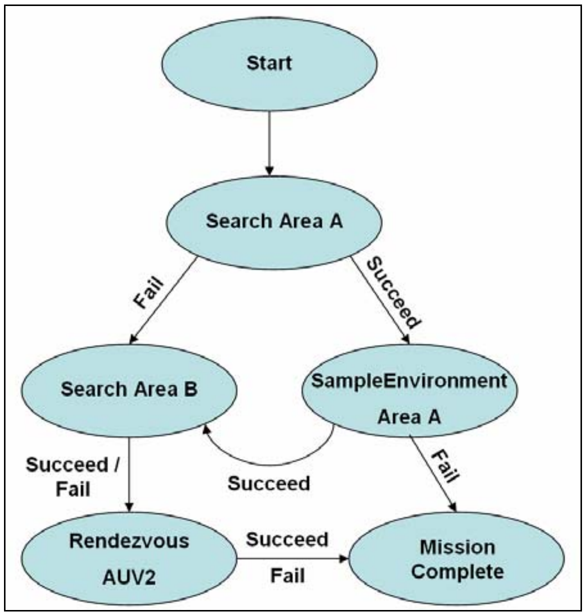


Figure 1: The finite state machine (FSM) portion of an example declarative goal-based AVCL mission specification.

3. MISSION PLANNING AND EDITING

Current mission-planning functionality of the AUVW utilizes the task-level mission definition portion of the AVCL schema. Declarative goal-based mission planning and rehearsal is currently under development. Multiple graphical user interface (GUI) displays are provided that allow for adding new task-level commands to a mission as well as for editing existing commands. Additionally, the mission editor supports simultaneous development and testing of multiple missions for all available air, surface, ground and underwater vehicles (Figure 3).

```
<GoalList>
    <Goal id="Search1" nextOnSucceed="Sample1"
                       nextOnFail="Search2">
        <Search datumType="point"
                requiredPD="0.8"/>
        <OperatingArea id="AreaA">
            <Circle>
                <Center>
                    <LatitudeLongitude
                        latitude="31.5"
                        longitude="-121.75"/>
                </Center>
                <Radius value="10000"/>
            </Circle>
        </OperatingArea>
    </Goal>
    <Goal id="Search2"
          nextOnSucceed="Rendezvous"
          nextOnFail="Rendezvous">
        <Search datumType="area"
                requiredPD="0.8"/>
        <OperatingArea id="AreaB">
            <Polygon>
               <LatitudeLongitude
                   latitude="33.0"
                   longitude="-122.0"/>
               <LatitudeLongitude
                   latitude="33.0"
                   longitude="-121.75"/>
               <LatitudeLongitude
                   latitude="32.75"
                   longitude="-121.75"/>
               <LatitudeLongitude
                   latitude="32.75"
                   longitude="-122.0"/>
            </Polygon>
        </OperatingArea>
    </Goal>
    <Goal id="Sample1" nextOnSucceed="Search2">
         <SampleEnvironment/>
         <OperatingArea idRef="AreaA"/>
    </Goal>
    <Goal id="Rendezvous">
        <Rendezvous contactID="AUV2"/>
        <OperatingArea idRef="AreaB"/>
    </Goal>
</GoalList>
```

Figure 2: AVCL defined declarative mission corresponding to the FSM depicted in Figure 1.

Five display formats are available for viewing AVCL task-level missions in the AUVW. Access to all task-level command editing functionality is available through the current mission's icon view (Figure 4). This view uses a list of icon/name pairs to graphically represent the task-level script of a mission. Individual task-level commands can be added to the end of the mission or inserted anywhere in the mission using either a pulldown or a popup menu. Dialog boxes similar to Figure 5 are used to edit individual commands. These are accessed via the same menus or by double-clicking the command in the icon list. Individual commands can be deleted, copied or moved to new locations in the script using pulldown or popup menus or user-specified hot keys.

Additionally, the popup and pulldown menus provide the capability to add general mission metadata set the mission's geographic origin upon which the mission's Cartesian coordinate system described later is anchored.

Although capable of accessing all AUVW task-level mission editing functions, the icon view does not depict the full AVCL document. Header information, metadata, command progress and mission results data elements, for instance, are not displayed. Two other display formats, however, can be utilized to view portions of the document not available in the icon view: the tree and text views, both of which display the entire AVCL document.

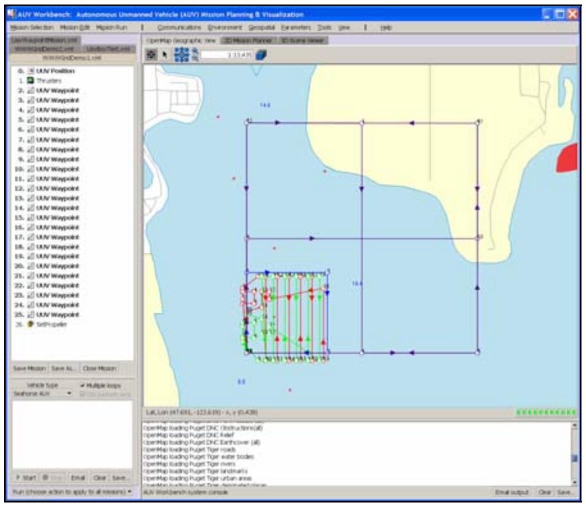
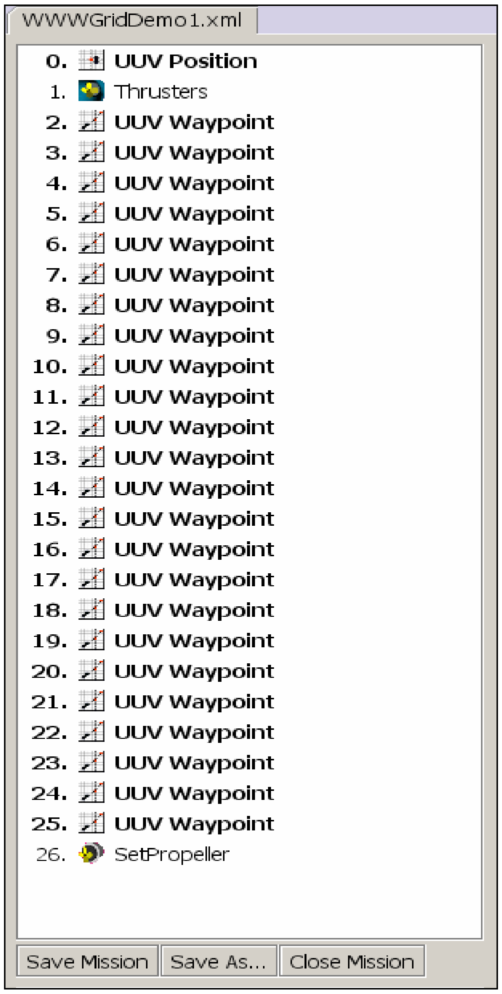


Figure 3: Screen snapshot of the Autonomous and Unmannned Vehicle Workbench being used to simultaneously edit autonomous air, surface and underwater vehicle missions.

The capability to view and edit all aspects of an AVCL task-level mission using the icon, tree and text views notwithstanding, most potential users will agree that a geographic interface is more intuitive in many instances for developing UV tasking. The AUVW, therefore, provides 2D interfaces that complement the functionality of the other display methods. The first displays the mission tracks of currently loaded missions on a Cartesian grid with the positive-X axis oriented true north and the positive-Y axis oriented true east (the resultant right-handed 3D system corresponds to the one utilized by AVCL and has the positive-Z axis oriented down). Locations can be entered using either Cartesian coordinates that can be plotted directly on the 2D display or latitude and longitude which are converted to Cartesian coordinates based on a user-defined geographic origin.

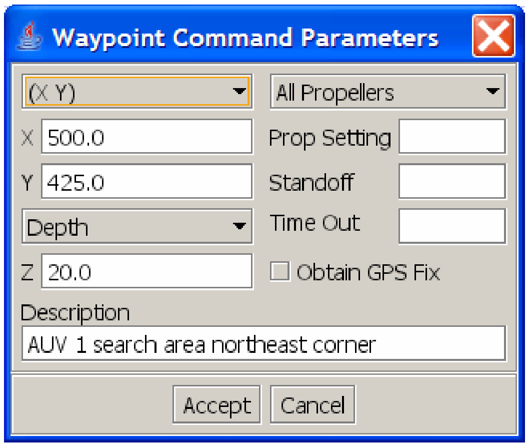


WWWGridDemo1.xml

0. UUV Position
1. Thrusters
2. UUV Waypoint
3. UUV Waypoint
4. UUV Waypoint
5. UUV Waypoint
6. UUV Waypoint
7. UUV Waypoint
8. UUV Waypoint
9. UUV Waypoint
10. UUV Waypoint
11. UUV Waypoint
12. UUV Waypoint
13. UUV Waypoint
14. UUV Waypoint
15. UUV Waypoint
16. UUV Waypoint
17. UUV Waypoint
18. UUV Waypoint
19. UUV Waypoint
20. UUV Waypoint
21. UUV Waypoint
22. UUV Waypoint
23. UUV Waypoint
24. UUV Waypoint
25. UUV Waypoint
26. SetPropeller

Figure 4: Autonomous Unmanned Vehicle Workbench icon view of an AVCL task-level AUV.

Unlike the icon, tree and text views, the 2D planner (Figure 6) does not display all AVCL task-level commands in the mission. Only commands having a geographic component such as waypoints and hoverpoints will be shown on the 2D display. Of the 26 task-level commands currently available for AUV use, only five (CompositeWaypoint, Hover, Loiter, Position and Waypoint) meet this criteria. Nevertheless, these are among the most common commands utilized in AUV missions, so most mission editing can be accomplished using the 2D editor. Drag and drop, snap to grid, click to highlight, and double-click to edit features support precise GUI modification of existing commands. The pulldown menu functionality to insert new commands (including those not visible in the 2D display), copy, move, edit or delete existing commands, add metadata or set the geographic origin provides full task-level mission editing capability using the AUVW 2D mission editor.



Waypoint Command Parameters

(X Y) | All Propellers
X 500.0 | Prop Setting
Y 425.0 | Standoff
Depth | Time Out
Z 20.0 | ☐ Obtain GPS Fix

Description
AUV 1 search area northeast corner

Accept | Cancel

Figure 5: Autonomous Unmanned Vehicle Workbench editing interface for an AUV waypoint command.

The final display available AUVW is the geographically-based OpenMap™ editor (Figure 7). OpenMap™ is an open-source Java API for handling geospatial data and digital map data [3]. The AUVW utilizes U.S. Census Bureau Census 2000 Tiger/Line data in shapefile format [9] and Digital Nautical Charts (DNC$^{®}$) to enable the user to plan missions for a specific geographic area. OpenMap™ allows the user to selectively enable and disable dataset layers, so the display can contain as much or as little geographic information as desired.

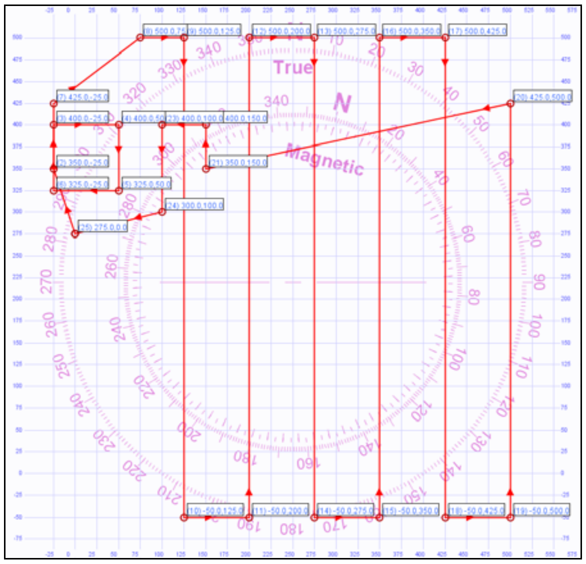


Figure 6: The 2D Cartesian coordinate-based editing interface for AVCL task-level script missions allowing mission authors to drag waypoints to reposition tracks and add/remove/inspect commands directly.

As with the 2D editor, only those task-level commands having a geographic component are overlayed in the OpenMap™ editor. While useable in its current state, the OpenMap™ editor is still under development. Thus mission editing capability is somewhat limited when compared to other edit modes of the AUVW. At present, physical manipulation of task-level AVCL missions using the OpenMap™ editor is limited to drag and drop repositioning of individual task-level command points, however full implementation of all 2D planner functionality is anticipated in the near future. Additionally, all previously discussed pulldown menu functionality can be accessed while using the OpenMap™ editor. Ultimately, it is envisioned that the this editor will closely mirror, and possibly replace, the current AUVW 2D planner.

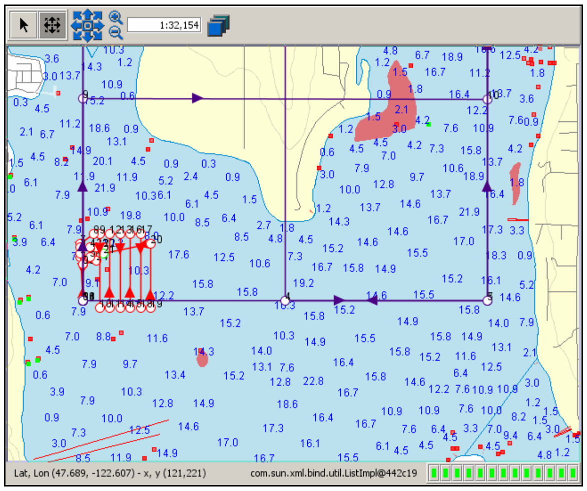

1:32,154
Lat, Lon (47.689, -122.607) - x, y (121,221)
com.sun.xml.bind.util.ListImpl@442c19

Figure 7: The OpenMap™ editing interface for AVCL task-level script missions.

4. MISSION REHEARSAL

Simulation in the AUVW

Among the most important aspects of the AUVW is the ability to realistically rehearse missions in a VE. Mission rehearsal in the AUVW utilizes physically based models that accurately represent the vehicles for which missions are being designed. All simultaneous simulated missions run in the same VE enabling the operator to effectively determine the effectiveness of multi-vehicle plans.

Simulation speed can be constrained to run in real time or accelerated to improve performance. During faster-than-real time simulations involving multiple vehicles, synchronization is maintained by consistently matching all vehicle speedup factors (e.g., 50 times real time). In this way, synchronization can be maintained not only among vehicle simulations running within a single AUVW session, but among those spawned by other, possibly distributed, AUVW instances as well.

3D visualization of mission progress utilizes X3D graphics and the Xj3D open source browser. 2D visualization is also provided via both the 2D planner and the OpenMap™ editor GUI windows.

Vehicle Physics Modeling

To date, physically based vehicle models have been implemented for autonomous underwater, surface and air vehicles. By nature, each is vehicle-nonspecific and can be parameterized to obtain accurate physical response for arbitrary vehicles of the given type. Appropriate parameters are loaded at run time from an XML configuration file corresponding to the vehicle being simulated. The AUVW provides a GUI menu for each mission being edited (immediately below the icon, tree or text view pane) for selection of the specific vehicle that is to be modeled. Additions and deletions from the set of available vehicles can be made by editing the master AUVW configuration XML file.

The AUV hydrodynamics model, fully described in [5], was first utilized to model the NPS Phoenix AUV. It is a 6-DOF rigid-body model capable of achieving accurate physical response for AUVs of various control configurations including those with cross-body thrusters. The model utilizes five environmental parameters and 28 vehicle-configuration parameters (mass, inertia matrix, control configuration, etc.). The six equations of motion utilize these parameters, 61 dimensionless force/moment coefficients, and a current state vector consisting of position, orientation, linear and angular velocity and all control settings to compute linear and angular accelerations for each closed-loop control cycle. Heun integration is utilized to update vehicle position, orientation and velocities based on the computed accelerations.

The AUVW unmanned air vehicle (UAV) model is significantly simpler than the AUV model. Nevertheless, it is robust enough to accurately model the physical response of all UAVs for which AUVW implementation is anticipated. Also a 6-DOF rigid-body model, the UAV model was first used at NPS to implement an A-4 Skyhawk aircraft in the NPSNET III distributed VE [5]. The model equations of motion rely on 14 parameters defining the vehicle configuration, 25 dimensionless aerodynamic coefficients and a state vector consisting of position, orientation, linear and angular velocity and control settings. Coefficients for the Predator UAV currently available in the AUVW were derived by matching the performance of an airfoil-specific model implemented as described in [4] with National Advisory Committee for Aeronautics (NACA) airfoil data from [1]. A simplifying assumption of the UAV model implementation is that atmospheric characteristics (e.g., air density, temperature at altitude, etc.) are computed based on a standard meteorological day (zero percent relative humidity, 288.15 degrees Kelvin and 29.92” Hg barometric pressure at sea level), however future improvements include the use of actual meteorological data obtained via web service at run time. As with the AUV model, Heun integration is utilized to update vehicle state variables for each timestep.

The simplest of the AUVW models implements the unmanned surface vehicle (USV) and is intended as a place holder until a more accurate physically based model is implemented. Unlike the AUV and UAV models, the USV model provides only two degrees of freedom to account for forward velocity and yaw rate. Further, the model is kinematic in nature and does not utilize forces and moments to compute vehicle accelerations. Rather, response is computed as a function of current velocity (linear or angular), the current control setting (rudder or propeller), and the maximum velocity possible with the current setting. Over time vehicle velocity will asymptotically approach the maximum possible control-setting-specific velocity. Simplicity notwithstanding, this kinematic model is capable of modeling reasonable response for envisioned USVs until a more robust model is implemented. As with the previous models, Heun integration is used to update vehicle state variables for each timestep.

Environmental Modeling

Proper modeling of environmental factors can produce major changes in sensor propagation, vehicle buoyancy, vehicle control and predicted power consumption. Therefore, multiple environmental datasets and services are being connected to the AUVW in order to maximize the real-world physics modeling capability for mission rehearsal and mission evaluation.

AUVW now includes the ability to read supercomputer-generated Network Common Data Form (NetCDF) [18] datasets which include 4D (x y z t) gridded time series of ocean parameters such as sound speed profile (SSP), local ocean current, wind speed, etc. Similar real-time oceanographic parameters are also available via XML-based mechanisms. Together with Fleet Numerical Meteorological Oceanographic Center (FNMOC) Monterey we have integrated XML-based Web Services queries using the Joint METOC Brokering

Language (JMBL) for query/response of real-time and projected ocean data. Additional environmental inputs (and outputs) are planned.

3D Visualization

The AUVW supports 3D visualization of mission progress during rehearsal and playback through the use of X3D—an International Organization for Standards (ISO) standardized format for web-capable 3D graphics. It utilizes an XML-enabled file format to facilitate the transfer of 3D data across networked applications. Significantly more robust than its web-capable 3D predecessor, the Virtual Reality Modeling Language (VRML), X3D includes implicit support for DIS networking and incorporates a rigorously defined Scene Access Interface (SAI) making it well-suited for use in the AUVW.

X3D-based 3D visualization is implemented in the AUVW with the Xj3D toolkit [11]. Xj3D is an open source API produced by Yumetech, Inc. for developing X3D-compliant applications. Implemented with the support of the Web3D Consortium as an exemplar X3D-compliant browser, Xj3D implements most aspects of the interchange, interactive and immersive X3D profiles [14] as well as a number of proposed extensions to the ISO standard. Figure 8 shows the AUVW Xj3D viewer being used to monitor the operations of multiple AUVs in a VE incorporating bathymetry and cartography near Panama City, Florida.

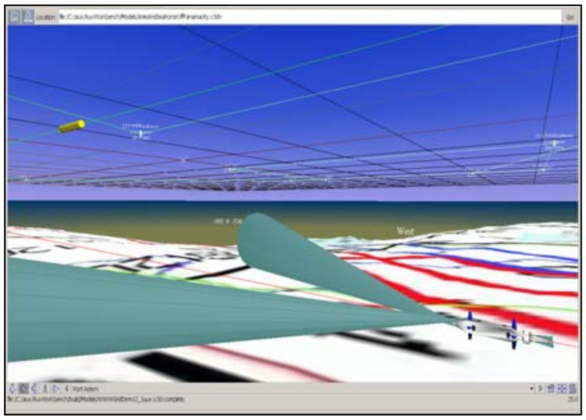


Figure 8: ARIES and Seahorse autonomous underwater vehicles operating in the same virtual environment as seen in the Autonomous Unmanned Vehicle Workbench Xj3D viewer.

A number of vehicle and VE models are included with the AUVW distribution. In addition, approximately 1000 models including vehicles, structures, sensors, terrain and even entire scenarios are available for unencumbered individual, government and corporate use in the Scenario Authoring and Visualization for Advanced Graphical Environments (SAVAGE) online archive (available at http://web.nps.navy.mil/~brutzman/Savage). A number of authoring tools are also available that facilitate the use of these and other models. Thus, the development of large VEs remains time consuming but is becoming a straight forward process. When coupled with the potential autogeneration of significant VE content, the rapid creation of realistic VEs to rehearse and visualize real-world operations of arbitrary UVs is becoming an achievable goal.

The X3D Scene Access Interface (SAI)

Among the most important Xj3D features is implementation of the X3D SAI—a portion of the X3D specification that provides for programmatic access to a loaded scene graph. Within the AUVW, the SAI enables dynamic generation of X3D content for addition to the existing VE as well as the manipulation of existing content.

The first implemented AUVW dynamic generation of X3D content using the SAI takes the form of mission-path trackline addition to the scene. Mission tracks are automatically created based on the content of the activated AVCL task-level script using X3D indexed line sets and billboards when the mission is loaded for rehearsal. This X3D corresponding to a mission's path is generated by applying an XSLT stylesheet to the AVCL document as described in the next section. The resultant X3D is then added to the current VE scene graph using the X3D SAI. If the mission is subsequently edited and rerun, the previously generated content is removed from the scene graph and replaced with updated content.

A second use of the SAI for manipulation of the AUVW VE is its use for sensor modeling. The VE contains all of the objects with which the UVs are intended to interact during mission rehearsal and playback. Enough information is therefore contained in the scene graph for vehicles operating in the VE to model various sensors through the use of collision detection and picking. [7] documented the use of C++ and the Open Inventor™ SoRayPickAction [17] to model mechanically steered the narrow beam active sonars installed on the NPS Phoenix AUV. Unfortunately, the X3D specification does not support general collision detection along the lines of that required for ray picking operations. Xj3D, however, implements a proposed formal extension to the X3D specification that supports various forms of picking suitable for sensor modeling [26]. Specifically, the AUVW uses the Xj3D PrimitivePicker node to obtain the same functionality provided by the Open Inventor™ SoRayPickAction. Individual nodes are created and added to the VE using the SAI upon the request of individual vehicle instances. Each vehicle can manipulate pickers via the SAI as required to model onboard sensors. Individual sensors can be modeled with single picker nodes (as depicted in Figure 9) or multiple nodes depending on sensor characteristics. Currently implemented vehicles use this functionality to model fathometers, sonar and radar altimeters and ranging sonars.

Distributed Interactive Simulation (DIS)

Vehicle position in the VE is maintained through the use of DIS updates with individual vehicles periodically transmitting multicast entity state protocol data units (PDU). Explicitly supported by X3D, entity state PDUs provide a means of simultaneously updating multiple views into a common VE. This inherently supports the use of multiple AUVW instances in a networked environment to provide for planning and rehearsal of multiple-vehicle missions from different locations by synchronizing the VE across the network.

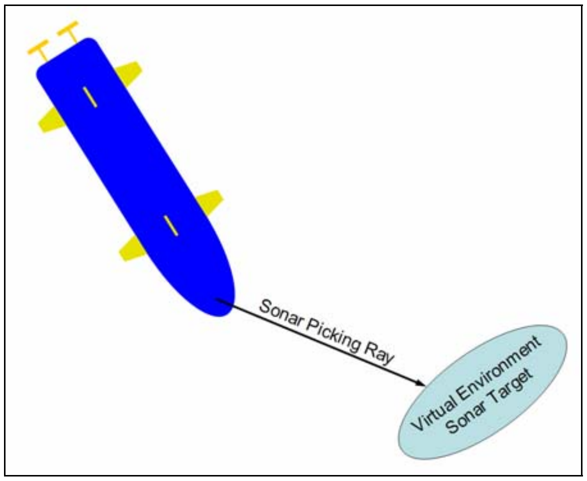


Figure 9: Sensor modeling using the X3D scene graph and Xj3D picking nodes. This approach allows modelin realistic sensor response in arbitrarily large VEs.

Potential DIS support highlights an additional planned use of the SAI: DIS entity monitoring to support the automatic addition of applicable vehicle models to the scene graph based on the characteristics of new entity state PDUs received specifically, the PDU's siteID, applicationID, entityID and marking. These four fields contain enough information to uniquely identify the type of vehicle and determine whether or not it is present in a given VE. With this information, an instance of the appropriate vehicle that will correctly respond to the entity state PDUs that its rehearsal instance is transmitting can be created dynamically and inserted into the VE scene graph.

5. VEHICLE SUPPORT

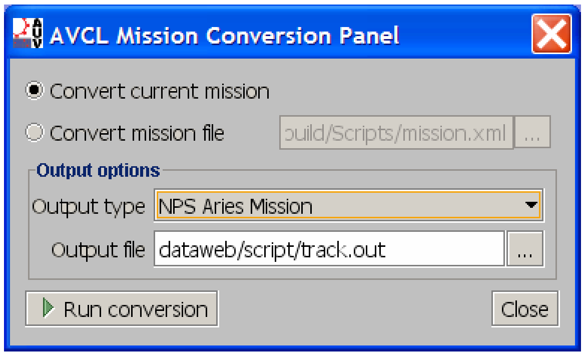
It is worth noting at this point that the common UV operational ontology defined by AVCL, along with the mission planning and rehearsal functionality of the AUVW provide no real-world value unless they support actual vehicles. In order to satisfy this requirement, the AUVW must be capable of importing and exporting vehicle-specific data for all supported vehicles. Fortunately, the use of XML in the AUVW enables automated translation to and from vehicle-specific data formats and facilitates compatibility with real-world vehicles. Additionally, communications capabilities are provided to support interaction with vehicles during pre-, in-, and post-mission operating phases.

Data Format Conversion

Enabling the translation of AVCL to vehicle-specific formats is the Extensible Stylesheet Language for Transformation (XSLT) [21]. XSLT is a pattern-matching, template-based language that is used to convert XML documents to other text formats. Although primarily intended as a tool for converting from one XML format to another, XSLT is more than capable of supporting conversion from XML to arbitrary text formats along the lines of those used by most UVs.

A number of APIs are available for the incorporation of XSLT transformations in applications written in a variety of programming languages. The AUVW provides XSLT stylesheets for each supported vehicle and uses the Xalan-Java API, an open source product of the Apache XML Project [2], to conduct transformations. The GUI panel of Figure 10 is used to initiate XSLT transformations to convert stored or loaded AVCL documents to vehicle-specific formats as required.

While the use of XSLT to convert AVCL documents to vehicle-specific data formats is fairly straightforward, the reverse transformation (i.e., vehicle-specific data to AVCL) is more problematic. Generally speaking, most vehicle-specific data is not maintained with XML, and no utility along the lines of XSLT is available to support such conversion to AVCL. Nevertheless, vehicle-specific data formats are highly structured and can be rigorously defined mathematically. Formally, any consistently defined vehicle-specific data format is a context-free language (CFL) that can be expressed using a context-free grammar (CFG).



● Convert current mission
○ Convert mission file: build/Scripts/mission.xml
Output type: NPS Aries Mission
Output file: dataweb/script/track.out

Figure 10: Support for automated conversion from AVCL to vehicle-specific data formats using the Extensible Stylesheet Language for Transformations (XSLT) stylesheets.

The AUVW uses a CFG definition for supported vehicle-specific data formats as the basis for conversion to AVCL and subsequent import into the AUVW mission editor. Development of a CFG for specific vehicle data formats enables automated parsing of instance documents. Once parsed, the resulting parse tree is traversed and translated to AVCL using templates in much the same way that XSLT traverses and converts an XML document. Described in detail in [8], this methodology has been utilized in the AUVW to enable the import of NPS ARIES AUV, Hydroid REMUS and Naval Oceanographic Office (NAVO) Seahorse AUV data.

Communications

All phases of UV operations generally require some level of communication between vehicles and operators. During the pre-mission phase, the operator must be able to initialize the vehicle and load and initiate missions. During mission execution, many vehicles are able to provide position and status reports or receive updated tasking. Following execution, mission results must be downloaded from the vehicle to offboard systems for analysis and archiving. The AUVW has a number of communications capabilities implemented or planned to support these requirements.

Communications involving UVs routinely utilize acoustic modems or other devices relying on serial communications. The AUVW implements user-configurable serial communications and Kermit protocol file transfer appropriate for point to point communications between the AUVW and a variety of devices. Also slated for implementation are File Transfer Protocol (FTP), Secure FTP (SFT), Terminal Emulation (TELNET) and Secure Shell (SSH) facilities that will improve the flexibility and efficiency of communications between the AUVW and controlled vehicles that use Transmission Control Protocol/Internet Protocol (TCP/IP) networking.

6. CONCLUSIONS

A platform-independent planning, rehearsal, execution and playback tool for UVs can make a significant contribution in the area of dissimilar vehicle interoperability. The NPS AUVW is just such a product. Built around an evolving common UV ontology, the AUVW provides a planning tool suitable for arbitrary UVs with a number of utilities available to facilitate operations with real-world vehicles including automated data format translations, rigorous physics-in-the-loop mission rehearsal, and AUVW-to-vehicle communications and data transfer support.

Already capable of supporting a variety of vehicles, a number of enhancements to the AUVW are anticipated that will further increase its utility in real-world operations. Among these are increased use of autogenerated X3D content in the 3D view, including the insertion of new, unannounced vehicles into the scene based on DIS packet characteristics. Also planned is the automated generation of scene bathymetry and topography based on run-time queries to networked web services, providing improved sensor modeling with real-time environmental data.

Increased networking based on web technologies and XML will continue to improve the functionality and usefulness of the AUVW. Therefore, key goals of a number of planned improvements are increased access to and utilization of network data bases, sensor models and 3D models using the web services. The use of web-based technologies will provide access to a significantly larger cache of information than is reasonable otherwise and leverage the capabilities of distributed systems in a net-centric environment. Further, it will enable AUVW users to accurately plan and rehearse missions for specific operating environments and foster collaboration among operators at distributed locations.

Availability and Development

Thanks to consistent use of Java and Java L+F, the AUVW has been successfully tested on Windows, MacOSX, Linux and Solaris. Autoinstall CDs are updated weekly online, and installation DVDs are available on request.

Source code is available under an open-source license that ensures unencumbered use by individuals, government projects and industry. All source code, configuration files and documentation are maintained under Concurrent Version System (CVS) control, allowing around-the-clock distributed development by qualified participants.

An archived mailing list is used to discuss design issues and problem resolution. The bugzilla tracking system is used to resolve all problems and precisely define new features.

Further participation is welcome. Additional information is available online at http://www.movesinstitute.org/xmsf/xmsf.html#Projects-AUV.

ACKNOWLEDGEMENT

The authors wish to acknowledge the financial support of the Navy Modeling and Simulation Office (NMSO), Naval Undersea Warfare Center (NUWC) CARUSO project, Navy Research Laboratory-Stennis Space Center, and the Singapore Defense Science and Technology Agency and Defense Science Organization National Laboratories.

REFERENCES

[1] Abbott, I. H. and Von Doenhoff, A. E., *Theory of Wing Sections*, Dover Publications, 1959.

[2] Apache Software Foundation, Xalan-Java 2.6.2 Online Documentation, 2004. Available at http://xml.apache.org/xalan-j/index.html

[3] BBN Technologies Solutions, LLC, *OpenMap Viewer Application User's Guide*, November 2001. Available at http://openmap.bbn.com/doc/user-guide.html

[4] Bourg, D. M., *Physics for Game Developers*, O'Reilly and Associates, 2002.

[5] Brutzman, D. P., *A Virtual World for an Autonomous Underwater Vehicle*, Ph.D. Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, California, December 1994. http://web.nps.navy.mil/~brutzman/dissertation

[5] Cooke, J. M., Zyda, M. J., Pratt, D. R. and McGhee, R. B., "NPSNET: Flight Simulation Dynamic Modeling Using Quaternions," *Presence*, v. 1, nbr. 4, pp. 404-420, Fall 1992.

[6] Daconta, M. C., Obrst, L. J. and Smith, K. T., *The Semantic Web, A Guide to the Future of XML, Web Services, and Knowledge Management*, Wiley Publishing, 2003.

[7] Davis, D. T., *Precision Maneuvering of the Phoenix Autonomous Underwater Vehicle for Entering a Recovery Tube*, Masters Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, California, September 1996.

[8] Davis, D. T., "Automated Parsing and Conversion of Vehicle-Specific Data into Autonomous Vehicle Control Language using Context Free Grammars and XML Data Binding," *Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology*, Durham, NH, August 2005.

[9] Environmental Systems Research Institute (ESRI) White Paper, "ESRI Shapefile Technical Description," July 1998. Available at http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[10] Hawkins, D. L. and Van Leuvan, B. C., *An XML-Based Mission Command Language for Autonomous Underwater Vehicles*, Masters Thesis, Department of Information Sciences, Naval Postgraduate School, Monterey, California, June 2003.

[11] Hudson, A. D., "An Introduction to the Xj3D Toolkit," *Proceedings of the 9th International Conference on 3D Web Technology*, Monterey, CA, April 2004.

[12] Hydroid, Inc., *Technical Manual Operations and Maintenance Instructions, REMUS Remote Environmental Measuring Units*, 2001.

[13] Institute of Electrical and Electronics Engineers (IEEE) Standard 1278.1-1995, *Standard for Distributed Interactive Simulation (DIS) Application Protocols*, 1995.

[14] International Organization for Standardization / International Electrotechnical Commission International Specification 19775:200x, *Extensible*

3D (X3D) International Specification, 2004. Available at http://www.web3d.org/x3d/specifications/#x3d

[15] Lee, C. S., *NPS AUV Workbench: Collaborative Environment for Autonomous Underwater Vehicles (AUV) Mission Planning and 3D Visualization*, Masters Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, California, March 2004.

[16] Means, W. S. and Bodie, M. A., *The Book of SAX, The Simple API for XML*, No Starch Press, 2002.

[17] Open Inventor Architecture Group (OIAG), *Open Inventor C++ Reference Manual*, Addison Wesley Publishing, 1994.

[18] Rew, R., Davis, G., Emmerson, S. and Davies, H. *The NetCDF Users' Guide, Data Model, Programming Interface, and Format for Self-Describing, Portable Data, NetCDF Version 3.6.1*, Unidata Program Center, May 2005.

[19] Serin, E., *Design and Test of the Cross Format Schema Protocol (XFSP) for Networked Virtual Environments*, Masters Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, California, March 2003.

[20] Sun Microsystems, Inc., *Java Architecture for XML Binding (JAXB*) Version 1.0, edited by Fialli, J. and Vajjhala, S., January 2003. Available at http://www.sun.com/xml/jaxb/jaxb-docs.pdf

[21] World Wide Web Consortium, XSL Transformations (XSLT) Version 1.0 Recommended Specification, edited by Clark, J., October 2001. Available at http://www.w3.org/TR/2001/REC-xsl-20011015

[22] World Wide Web Consortium (W3C), Extensible Markup Language (XML) 1.0 (Third Edition) Recommended Specification, edited by Bray, T., Paoli, J., Sperberg-McQueen C. M., Maler, E., and Yergeau, F., February 2004. Available at http://www.w3.org/TR/REC-xml

[23] World Wide Web Consortium, XML Schema Part 1: Structures Second Edition Recommended Specification, edited by Biron, Thompson, H. S., Beech, D., Maloney, M. and Mendelsohn, N., October 2004. Available at http://www.w3.org/TR/xmlschema-1

[24] World Wide Web Consortium, XML Schema Part 2: Datatypes Second Edition Recommended Specification, edited by Biron, P. V. and Malhotra, A., October 2004. Available at http://www.w3.org/TR/xmlschema-2

[25] World Wide Web Consortium, Document Object Model Level 3 Core Specification Recommendation, edited by Le Hors, A., Le Hegaret, P., Wood, L., Nicol, G., Robie, J., Champion, M. and Byrne, S., April 2004. Available at http://www.w3.org/DOM

[26] Yumetech, Inc., "Xj3D Picking Extensions," Proposed extension to the X3D specification, 2004. Available at http://www.xj3d.org/extensions/picking.html