1998

Modeling and simulation for the FAU AUVs: Ocean Explorer

Lin, Huaying

Modeling and Simulation for the FAU AUVS: Ocean Explorer

Huaying Lin[+], Dave Marco[++], Edgar An[+], Krish Ganesan[+], Sam Smith[+], Tony Healey[++]

Email correspondence: ean@oe.fau.edu

Abstract:

This paper describes the research progress made on modeling and simulation development for the Florida Atlantic University autonomous underwater vehicles (AUV). Recent addition of simulation components include kinematic effect of longitudinal waves, inertial and position sensor dynamics so that realistic scenarios can be better accommodated. In addition, the existing FAU communication protocol used for the onboard acoustic modem has been ported to the simulation platform, thereby enabling multiple vehicle operations and communication to be simulated. At this stage acoustic propagation for the modem is assumed to be ideal although a more realistic model for shallow water propagation will be developed in the near future. This research endeavor is supported by a 5-year ONR MURI project and is jointly carried out by FAU and Naval Postgraduate School.

I. Introduction

Autonomous underwater vehicles (AUV) are emerging and proven technology with which synoptic collection of four-dimensional oceanographic data and rapid environmental assessment of hostile environments can be quantified. This added utility value to the underwater community provides an unprecedented opportunity to validate existing ocean models for better now-casting and forecasting capabilities. To ascertain such added value the performance reliability of these vehicles and data throughput must be thoroughly evaluated. AUV is a highly complex system, and its development, integration and testing require the functionality of each and every system's components to be properly evaluated. This process is generally time-consuming and can be very costly if it is to be implemented at-sea. A more sensible and cost-effective approach is to preliminarily evaluate and troubleshoot the system-level development via modeling and simulation concept [1].

An important criterion associated with modeling and simulation facilities is that both the simulation and actual platforms must be as closely compatible as possible otherwise additional overhead is thus needed to port the tested components to the vehicle. Given a well-matched simulation platform, extensive analysis and visualization procedures can be carried out, providing valuable insights into pre-flight mission profiles. The platform can also be used to train non-AUV personnel to gain insights into routine at-sea operations and mission planning for specific applications. Considerable system development, verification and at-sea operations can also be carried out in parallel provided that the platform architecture is modular.

Our main goal in the modeling and simulation work is to support the development and testing of FAU AUVs. This paper is organized as follows: a brief overview of the existing software architecture will be given in Section II. It is then followed by a description of the simulation platform regarding its implementation and capability. Section IV presents a brief overview of the acoustic communication implementation that can be used for co-operative multiple vehicle operations. Section V summarizes the work that has been achieved and the remaining work targeted for the MURI project.

II. High-Level Software Architecture

Our current high-level software architecture is built upon a global shared memory which allows managers and daemons to communicate indirectly with each other [3]. Shared memory is commonly considered as the most efficient way to implement inter-process communication, provided that suitable semaphores are allocated for handling mutual exclusion and synchronization. To

[+] Department of Ocean Engineering, Florida Atlantic University
[++] Department of Mechanical Engineering, Naval Postgraduate School

automate the construction of the shared memory and minimize unnecessary typographical error, a filter program, which takes in a simple text, has been developed for the vehicle development and can be used to automatically generate the required C-header files and dedicated function calls for accessing the shared memory. The text file contains information of each of the variables, such as name, type, initial value, unit and sampling rate. Each variable can be a single element or a complicated structure. A "*one writer many readers*" policy is typically applied to each of the variables although in some cases where it might be desirable to obtain a semaphore if a group of element values within a structure are needed synchronously. The filter program also automatically generates a semaphore and a time-stamp for each of the variables.

Each of the managers shown in Figure 1 is implemented as a task with a pre-defined scheduling characteristic to accomplish its dedicated behavior. Most of the managers are synchronous, and run at fixed rates. For an example, the *Head Manager* executes a variety of controllers via function calls. Different heading objectives of these controllers include crabbing compensation, line tracking between way-points, and open-loop rudder control. An arbiter, a heading arbiter in this case, associated with each of the managers, is then used to resolve the final rudder angle based on its chosen selection criterion. Note that although the arbiter selection criteria are pre-defined before every mission, it can be dynamically changed during a mission, and this can greatly enhance the overall mission capability. Additional heading controller's features involve only adding extra inputs to the heading arbiter and calling appropriate functions in the *Head Manager* in sequence. Other managers can be explained based on the same principle. Thus the combination of shared memory and the arbiter-manager configuration allows the system to be scalable without involving much rewriting of codes. This is a very important attribute inherited in any complex system design with a bottom-top approach.

MonServer, being implemented as a daemon, is a TCP/IP socket server program. It is used to dynamically monitor or set the values of the shared memory variables demanded from the remote client. Meanwhile, a *Monitor* program, which is running on the remote computer, has been developed as a TCP/IP socket client program with X-Motif GUI to receive and display data from *MonServer*. Figure 2 shows the *Monitor* layout where the first column displays the current variable values and the second column the desired values. The display will get updated as soon as any shared memory variable changes its value. This toolbox can be very useful when testing and debugging errors on a real-time operating system. The selection of particular shared memory variables is easily controlled by an ASCII file, which contains a list of variables to be monitored.

Logger, being implemented as another daemon, can be used to record the entire record associated with each of the chosen variables in terms of its value, unit, precision and name at a given sampling frequency along with its update time. Similar to *Monitor*, the selection of shared memory variables to be logged is controlled by an ASCII file. Note that the logged file generated is completely self-contained, and can provide enough information to reproduce the entire mission during post-processing.

III. Simulation Implementation

The vehicle software is a multitasking program, and is run on a real time operating system VxWorks that support POSIX standard. Whereas the simulation is currently run on an SGI with IRIX version 5.3. Note that this version doesn't support POSIX standard, and IRIX6.3, which supports POSIX, will be used at the next design iteration. In order to maintain compatibility between the vehicle and simulation platform, several issues regarding the OS differences, described in the following have been taken into consideration:

- **Task creation** Threads are the first choice to generate tasks and will be implemented on IRIX 6.3. Although IRIX 5.3 doesn't support POSIX thread, SGI light weight sub-process generating functions sproc works fine to generate subtasks. As of the difference between Unix and VxWorks, task scheduling such as task delay and task suspension has to be tuned to make simulation work correctly.
- **Shared memory** Early simulation version was a multi-process programming and the Unix shared memory mechanism was used. It was found to be awkward and uneasy to implement. Current *sproc* or *pthread* task creating mechanism solves this problem because tasks shared the same address space. A C filter program, which automatically converts from the VxWorks codes to the Unix codes, has been written.
- **System symbol table** The VxWorks OS has a built-in system symbol table. This table contains pointers to all the loaded functions and can be called via the corresponding function names. The current SGI Unix does not have that capability and thus an AUV symbol table was implemented for compatibility reason. The source code used to create symbol table is also automatically generated by the UNIX filter program mentioned before.
- **Timers** Up to 20 real-time timers are required by the vehicle platform for multi-tasking control. On our Unix system, a group of timers have been implemented as a circular buffer. Once a new timer is created, it is registered into that buffer. When the

Unix timer is waken up, it checks all the expired timers, executes the related functions and finds the minimum waiting time to reset the Unix timer. To make a timer more accurate, timer related functions must be simple, such as just release a semaphore.

- **Semaphore** Currently IRIX 5.3 only supports a limited number of UNIX named semaphores. As the number of variables increases, the number of UNIX named semaphores will reach the system limitation. IRIX 6.3, which supports POSIX named and unnamed semaphore, can solve this problem. Semaphore related functions have been rewritten to mimic VxWorks system.

Besides the above-mentioned OS implementation issues, the simulation platform requires a high-fidelity vehicle, sensor and environment model before any realistic mission can be simulated. The vehicle model being used is a nonlinear 6 degree-of-freedom hydrodynamics model tailored specifically for the FAU Ocean Explorer AUV [5]. A short description of the Ocean Explorer is given in the following whereas its detailed description can be found in [7]. The OEX body length is 7.14 feet and maximum diameter 21 inches; its weight in air is 714.2 lbf and displaced weight is 716.7 lbf. To simulate the environment, a longitudinal surface wave model which is based on standard linear wave theory, has been developed. Motion and position sensor models with standard bias and variance parameters have been incorporated. In addition to the above features, the compass model also includes local magnetic field interference which was experimentally obtained using the onboard flux gate compass TCM2 [4]. Global Positioning System (GPS) sensor model includes noise levels and time constants for standard and differential signals. To estimate the vehicle position, an extended Kalman filter, which fuses the motion and position sensors, has been developed and tested.

Visualization of 3D-motion simulation is based on SGI Open Inventor GUI, and the data are sent via a multicast socket so that multiple vehicles can be displayed on the same screen simultaneously. Platform independent GUI, such as JAVA-VRML, will be soon integrated into the next-generation simulation design. Numerical visualization is based on the same Monitor layout with a modified MonServer interface. To run the simulation program, an X-Motif GUI, which allows daemons and managers to be individually spawned, has been written. A mission plan can be specified using an ASCII file in terms of way/set points. The mission plan construct is same as that used on the OEX.

IV. Acoustic modem

To simulate the acoustic communication capability between AUV and ship or AUV-AUV, a separate X-Motif GUI, which allows commands to be entered or statuses to be displayed, has been developed. This GUI can be run on any machine on the same network, and communicates with the simulation via UDP messaging. The existing acoustic modem protocol belongs to a peer-to-peer type that supports functions such as send mission commands, abort mission, receive AUV position and its state [6].

There are two kinds of acoustic data structures defined in shared memory, one for an incoming message, one for an outgoing message. Each acoustic message comes with a specified source and destination address, a binary data block that is currently limited to 5 bytes long, a message *id* and finally an error check bit. When a message is sent, the sender modem expects to receive an acknowledgement message. If the acknowledgement message is not received from the remote host for a pre-defined period of time, the sender modem will enter a re-transmit mode and re-send the same data. If the transmission keeps failing for several times, a failure message will be reported. To minimize the message size to within 5 bytes, a data conversion table and a group of function maps are designed. The 5 bytes are divided into six non-overlapping parts, three 4-bit-block and three 8-bit-block. See Table 1 for a sample of commands and data block definition.

For example, when a "*set way-point*" command is sent together with the 5 bytes data, *nib1* should be set to 5 which indicates the command "*SetWaypointNE*". *nib2* and *nib3* specify the sign of x and y-direction respectively (positive values indicate north or east). Because only 8 bits are used each to store the *y* and *x* position in *byte1* and *byte2* respectively, *nib4* specifies which element in the conversion table to be used. Up to 15 different conversion values are available for the 4-bit block. The latitude and longitude are thus computed by multiplying the value in *byte1* or *byte2* with the conversion number specified by *nib4*.

V. Remarks

Multi-task instead of multi-process implementation is an important improvement for the simulation, as this resulted in minimal code change. The limitation of number of tasks has been extended by resetting UNIX system parameter in order to match the need for the simulation.

Navigator, planner and guidance managers have been successfully ported to the simulation platform. Thus, complex commands like tracking way-point control can be executed as well as simple set-point commands. The description of these managers is beyond the scope of this paper.

Besides getting commands from either the mission plan file or Monitor, the simulation model can also accept commands from the simulated acoustic modem from ship or any other. This feature improves the flexibility of AUV.

For intricate mission profiles, sending high level control commands whenever possible is more efficient in terms of writing the mission plan files. Next generation mission script will incorporate the use of macros and repeat commands to define the high-level commands. Currently, a yo-yo controller, which performs either stair-case or glideslope maneuvering, is being implemented as part of the guidance manager.

Remaining tasks of the MURI project include the development of a shallow-water acoustic propagation model for characterizing acoustic modem communication and adaptive sampling and search performance. The existing protocol will be revised for supporting communication among multi-vehicle.

AUV hovering and docking capabilities will be implemented, and sonar, obstacle and bathymetry models will also be developed.

VI. References

[1] Chen, X., Marco, D., Smith, S., An. E., Ganesan K., Healey T. 6 DOF nonlinear AUV Simulation Toolbox Ocean'97, Oceans 97 conference, Halifax, Nova Scotia, October, 1997.

[2] Fossen T. I. Guidance and Control of Ocean Vehicles, Prentice Hall, 1997.

[3] Ganesan K., Smith S., White K., Flanigan T. A Programtic Software Architecuter for UUVs, UUST Conference, Durham, NH, September, 1995.

[4] Grenon G. TCM2: Compensation of Error on AUV, FAU internal report, Ocean Engineering Dept, FAU.

[5] Humphreys D. E. Vehicle Hydrodynamic & Maneuvering Model for the FAU Ocean Explorer Vehicle (OEX), V.C.T Technical Memorandum 96-05 (available upon request).

[6] Neel A., LeBLanc L.R., Park, J.C., Smith, S. Peer-to-Peer Communication Protocol, for Deep-, Shallow-Water Communication, Modem Protocol Enables Coordination Among Multiple Platforms – Reliably, Effectively. Sea Technology, 10-15, May, 1998.

[7] Smith S., Heeb K., Frolund N., Pantelakis T. The Ocean Explorer AUV: A Modular Platform for Coastal Oceanography, UUST 95, pp.67-73.

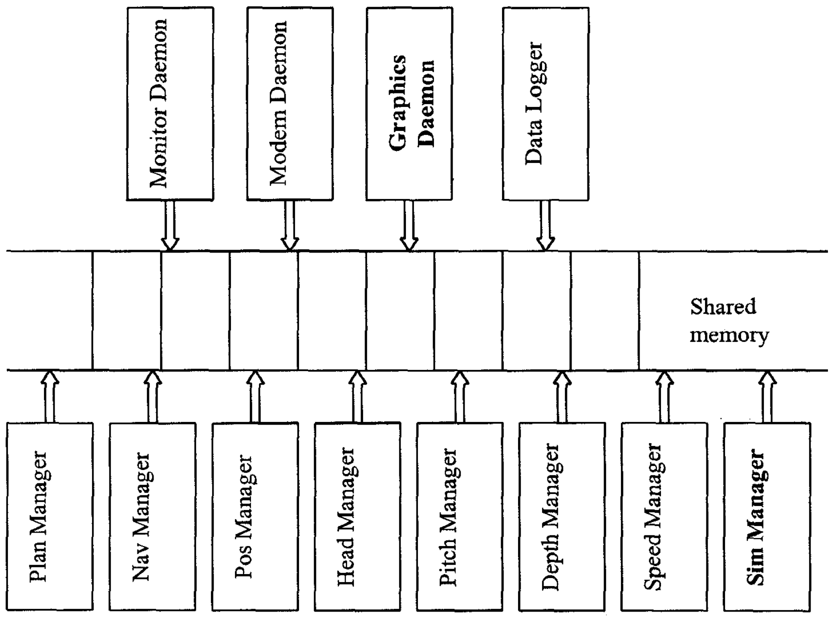
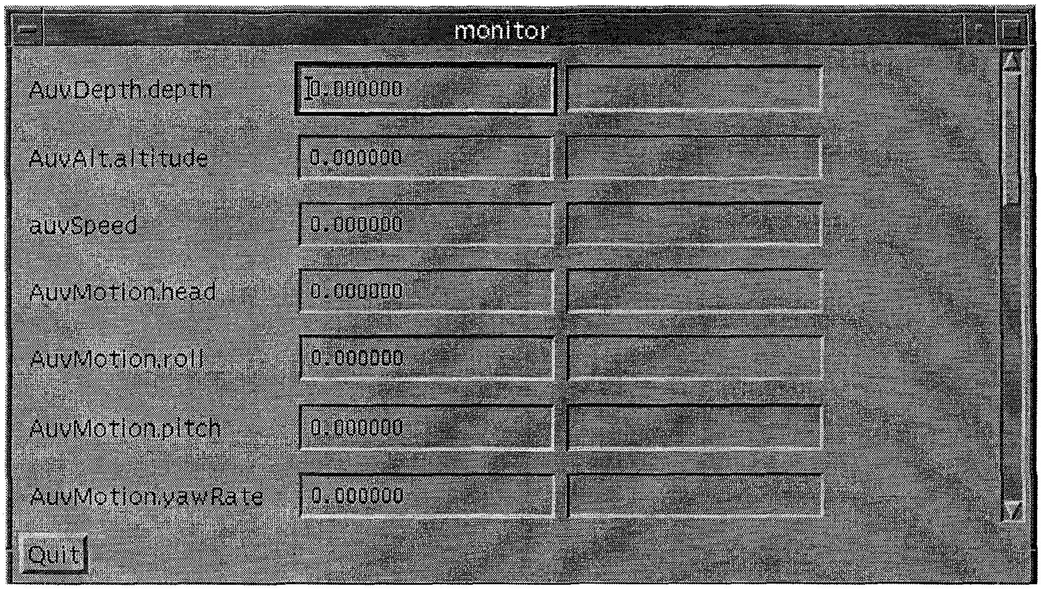


Figure 1 Simulation Architecture



AuvDepth.depth 0.000000
AuvAlt.altitude 0.000000
auvSpeed 0.000000
AuvMotion.head 0.000000
AuvMotion.roll 0.000000
AuvMotion.pitch 0.000000
AuvMotion.yawRate 0.000000

Figure 2 Monitor User Interface

Command Name	4 bits nib1	4 bits nib2	4 bits nib3	4 bits nib4	8 bits byte1	8 bits byte2	8 bits byte3
ChangeCurCmdMapTbl	0	CmdTblNum					
SendAuvPosData	1	xRel sign	yRel sign	CnvTblNum	X related pos.	Y related pos.	depth
SendAuvStateData	2	Heading CnvTblNum	speed CnvTblNum	Gps state + 1	heading	speed	weight
SetCmdNoCrt	3	sign	CnvTblNum	CmdNum	value		
SetCmdWithCrt	4	sign	CnvTblNum	CmdNum	value	Criteria Value	Criteria delta
CmdWaypointNE	5	y sign	x sign	CnvTlbNum	Y position	X position	plane
CmdGoHome	6						
CmdAbort	7						
CmdStartMission	8						
CmdEndMission	9						
CmdStartMotor	10						
CmdStopMotor	11						
SetFeature	12				feature		
TerminateCurrentCmd	13						

Table 1 Acoustic Command and Data Definition