2002-09

Execution level Java software and hardware for the NPS autonomous underwater vehicle

Ayala, Miguel Arnaldo

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

EXECUTION LEVEL JAVA SOFTWARE AND HARDWARE FOR THE NPS AUTONOMOUS UNDERWATER VEHICLE

by

Miguel Arnaldo Ayala

September 2002

Thesis Advisors: Don Brutzman
Man-Tak Shing

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

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 instruction, 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** September 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Execution Level Java Software and Hardware for the NPS Autonomous Underwater Vehicle			**5. FUNDING NUMBERS**
6. AUTHOR(S) Miguel Arnaldo Ayala			
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME AND ADDRESS N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public distribution; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT

Autonomous underwater vehicles (AUVs) have a great potential use for the United States Marine Corps and United States Navy. When performing amphibious operations, underwater mines present a danger for the forces going ashore. The use of underwater vehicles for the detection of this mines and signaling to the Amphibious Ready Group is very attractive. With advancements in hardware and object oriented language technology, more complicated and robust software can be developed. The Naval Postgraduate School Center for AUV Research has been designing, building, operating, and researching AUVs since 1987. Each generation of vehicles has provided substantially increased in operational capabilities and level of sophistication in the hardware and software respectively.

With the advancement in real-time computer languages support, object oriented technology, and cost efficient and high performance hardware, this thesis lays the foundations to develop a software system for the execution level using the Java language. We look into the Java Real-Time specifications and extension to familiarize with the capabilities of Java for real-time support, and study Java boards and its application for embedded real-time systems. We developed an object-oriented design for the execution level control software and implemented the design in Java. A testing phase is still under work.

14. SUBJECT TERMS Software Engineering, Autonomous Underwater Vehicles, AUV, Java, Java Board, Embedded Java, Real-time Software, Real-time Java, UML			**15. NUMBER OF PAGES** 280
			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 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EXECUTION LEVEL JAVA SOFTWARE AND HARDWARE FOR THE NPS AUTONOMOUS UNDERWATER VEHICLE

Miguel A. Ayala
Captain, United States Marine Corps
B.S.M.E., University of Puerto Rico, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2002

Author:

Miguel A. Ayala

Approved by:

Don Brutzman, Thesis Advisor

Man-Tak Shing, Thesis Co-Advisor

Chris Eagle, Chair
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Autonomous underwater vehicles (AUVs) have a great potential use for the United States Marine Corps and United States Navy. When performing amphibious operations, underwater mines present a danger for he forces going ashore. The use of underwater vehicles for the detection of this mines and signaling to the Amphibious Ready Group is very attractive. With advancements in hardware and object oriented language technology, more complicated and robust software can be developed. The Naval Postgraduate School Center for AUV Research has been designing, building, operating, and researching AUVs since 1987. Each generation of vehicles has provided substantially increased in operational capabilities and level of sophistication in the hardware and software respectively.

With the advancement in real-time computer languages support, object oriented technology, and cost efficient and high performance hardware, this thesis lays the foundations to develop a software system for the execution level using the Java language. We look into the Java Real-Time specifications and extension to familiarize with the capabilities of Java for real-time support, and study Java boards and its application for embedded real-time systems. We developed an object-oriented design for the execution level control software and implemented the design in Java. A testing phase is still under work.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my parents, for unconditional support in all my endeavors, to my wife Daisy and son Luis Miguel for their sacrifices, understanding, and support during the last two years.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. INTRODUCTION

Autonomous underwater vehicles have a great potential use for the United States Marine Corps and the Navy. When performing amphibious operations, underwater mines present a danger for the forces going ashore. The use of underwater vehicles for the detection of this mines and signaling to the Amphibious Ready Group is very attractive.

Research on AUV has been of interest for quite some time and is the focus of effort for many groups in many universities, private institutions and the government (Figures 1, 2 and 3).

Figure 1. Bluefin Robotics AUV recovered during Fleet Battle Exercise Juliet, Camp Pendleton, CA, July 2002.

The existing AUVs at NPS (Figure 3) are robots for student research in shallow water sensing and control. The AUV is controlled by three layered software architecture known as The Rational Behavior Model (RBM) [Byrnes 96]; execution, tactical and strategic layers respectively. The execution layer, studied in the scope of this thesis,

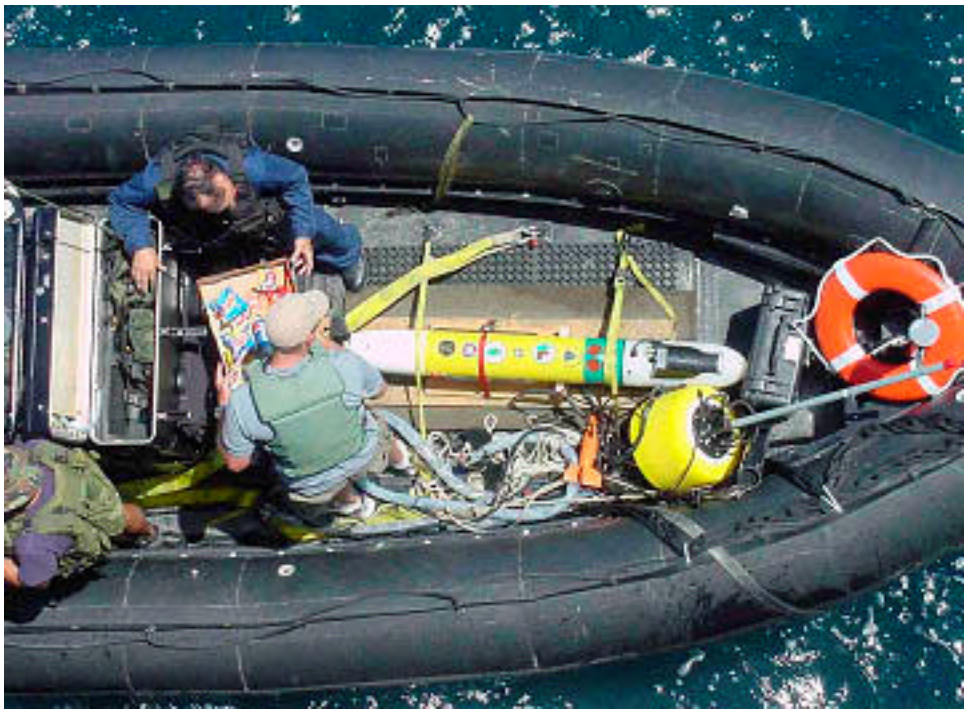

Figure 2. Remus AUV ready to be launched during Fleet Battle Exercise Juliet, Camp Pendleton, CA, July 2002.

corresponds to hard real-time reactive control. This layer provides maneuvering control of plane surfaces and propellers. Interface with the different sensors aboard the vehicle is also provided through the layer. The execution level integration includes control of physical devices, sense-decide-act, reactive behaviors, communications connectivity, a mission script language, and stand-alone robustness in case of loss of higher software levels.

The existing NPS AUV control software has been written in C [Marco 96]. The purpose of the execution level software is to provide control commands to the different actuators onboard the vehicle. The execution level code is the heart of the vehicle, in which provides the mechanism to guide the vehicle and execute missions successfully.

B. MOTIVATION

Virtual environments provide a realistic arena for the testing and development of future vehicle technologies [Byrne 98]. Amphibious operations conducted by the Navy/Marine Corps team faces numerous dangers in littoral waters. The presence of underwater mines in shallow waters in the event of an amphibious operation constitutes a real danger for the forces going ashore. Numerous research efforts have been directed towards the area of detecting, identifying and neutralizing underwater mines. The use of autonomous underwater vehicles has been one of the research areas that interest the United State Navy and Marine Corps. The Naval Postgraduate School Center for Autonomous Underwater Vehicle (AUV) Research have developed a series of AUVs with this purpose and command and control between vehicles (Figure 3).

Figure 3. Naval Postgraduate School ARIES AUV ready to be launched off the coast of The Azores, Portugal, Summer2001.

With the advent of the Internet, the concept of collaborative planning, execution of missions using the Internet started to take shape. With the introduction of Java as a programming language, those concepts became reality as Java took over as one of the World Wide Web programming languages. The characteristics of Java as highly portable and platform independent program environment offered good advantages for the transition of the NPS AUV software into Java. The visualization in 3D interface of the AUV using Virtual Reality Modeling Language was the prime candidate to proceed in this approach with the follow-on of the entire transition of the execution level code.

With the advancement in real-time computer language support, object-oriented technology and cost efficient and high performance hardware, a different approach for a new generation of AUV hardware and software can be attempted. A new underwater vehicle with Java-based architecture to improve reliability and verifiability can be studied. Java hardware for in-water hardware will provide fast performance, low power consumption and improved endurance. Accessibility to both low-level and advanced Java Applications Programming Interfaces (APIs) will make the software robust and easy to maintain, contributing to low maintenance cost. Perhaps most importantly to NPS, students will be able to test and develop AUV software compatible on any other computer, at home or in the lab.

C. OBJECTIVES

The objectives of this thesis are to provide sound software engineering practices in the transition, design and development of the execution level code for the NPS AUV. These objectives can be achieve by:

- Providing a good set of software requirements specification using UML and other software engineering practices.
- Building a Java based execution level that is efficiently capable of being interchangeable with virtual world without any effect, in other words, that the AUV software could not notice the difference if it is operating in the water or the virtual world.
- Preparing execution level software for Real-time Java migration and Java Board hardware integration solution.

D. SCOPE

The scope of the thesis is focused on the execution level code of the NPS AUV specifically: the study, understanding and therefore the translation of the C code into Java. Since the execution code is very complicated and involves numerous interfaces with hardware, the virtual world has been selected as the first transition bridge. From there the code will be tested for correct operation. In future work real world code will be incorporated. That interfaces the Java board with the actual AUV hardware components.

E. THESIS ORGANIZATION

Chapter II examines previous work on the ARIES AUV, specifically in the virtual world. Execution level code and the dynamics code are also described.

Chapter III presents the problem statement, goals for the thesis, and overview of Java as a computer language and why Java is useful for an autonomous underwater vehicle.

Chapter IV presents the Software Engineering requirements for the thesis, problem domain, constraints, issues and the software architecture for the execution level implementation. This is supported by Unified Modeling Language (UML) diagrams and Use Cases.

Chapter V discusses the Java board as a candidate hardware solution. The implementation of Java boards and Real-time Java for embedded systems are explored in details.

Chapter VI provides a close look to the source code and an explanation of the classes.

Chapter VII presents design methodology for testing of the source code and the results of those tests.

A summary, conclusions and future work recommendations are provided in Chapter VIII.

Appendices provide download information and source code summary.

THIS PAGE INTENTIONALLY LEFT BLANK

II. RELATED WORK

A. INTRODUCTION

Research on autonomous underwater vehicles has been growing during the last decade. With advancements in real-time computer language support, object-oriented technology and cost-efficient high-performance hardware, development of more sophisticated vehicle software is now possible. This chapter summarizes pertinent previous work on the current NPS AUV and its virtual world software.

B. NAVAL POSTGRADUATE SCHOOL ARIES AUV

1. Description

The existing AUVs at NPS are robots for student research in shallow-water sensing and control. The AUV is controlled by three-layered software architecture known as The Rational Behavior Model (RBM) [Byrnes 96], comprising execution, tactical and strategic layers respectively.

The Acoustic Radio Interactive Exploratory Server (ARIES) AUV is approximately two meters long weighing approximately 500 pounds. It was designed to operate neutrally buoyant. It is designed for communications server and Command and Control research. The vehicle is also used to develop low-cost underwater navigation capabilities using Differential Global Positioning System (DGPS) when surfaced. It has a top speed of approximately 4 knots, an operating depth of approximately 50 meters, and an endurance of up to 4 hours. It has bottom following, track following, station keeping and bottom sitting capability. It communicates through acoustic modem up to a range of 700 meters. Navigation means are Acoustic Ground Locked Doppler, Inertial Measurement Unit (IMU), Compass, Dead Reckoning, and Global Positioning System (GPS) correction when surfaced. Thus ARIES vehicle can operates as a shallow-water communications server vehicle with a DGPS and a doppler aided IMU / Compass navigation suite.

2. ARIES Software

The current ARIES software architecture is designed to operate using a single computer processor or two, independent, cooperating processors linked through a

network interface. Splitting the processing between two computers can significantly improve computational load balancing and software segregation. Each processor assumes different tasks for mission operation. Both computers run the QNX real-time operating system using synchronous socket sender and receiver network processes for data sharing between the two. In water ARIES software is written in C and C++ languages.

All vehicle sensors are interrogated by separate, independently controlled concurrent processes, and there is no restriction on whether the processes operate synchronously or asynchronously. Since various sensors gather data at different rates, each process may be tailored to operate at the acquisition speed of the respective sensor. Each process contains a unique shared memory data structure that is updated at the specific rate of each sensor. All sensor data are accessible to a synchronous navigation process through shared memory that is a main feature of the software architecture [Marco 96].

A virtual world has been developed as an aid to visualization of vehicle behavior as well as for real-time hardware in the loop, control code testing and evaluation [Brutzman 94].

B. VIRTUAL REALITY MODELING LANGUAGE (VRML) AND EXTANSIBLE 3D (X3D) GRAPHICS FOR ARIES

Underwater vehicles robots are unique. They operate in harsh environments for extended period of time. Mission planning, rehearsal and playback in a controlled physics-based environment is needed to preview an expected behavior of the vehicle in the water.

Virtual Reality Modeling Language (VRML) provided the necessary tools to develop a virtual AUV [Brutzman 94]. With the creation of a networked virtual AUV (Figure 4), visualization of mission planning, rehearsal, and mission analysis can be performed.

The development of the virtual AUV was done using X3D-Edit authoring tool developed by Don Brutzman, Naval Postgraduate School (NPS), Monterey, CA (Figure 5). The software expressed the geometry and behavior capabilities of the Virtual Reality Modeling Language (VRML 97) using the Extensible Markup Language (XML). 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

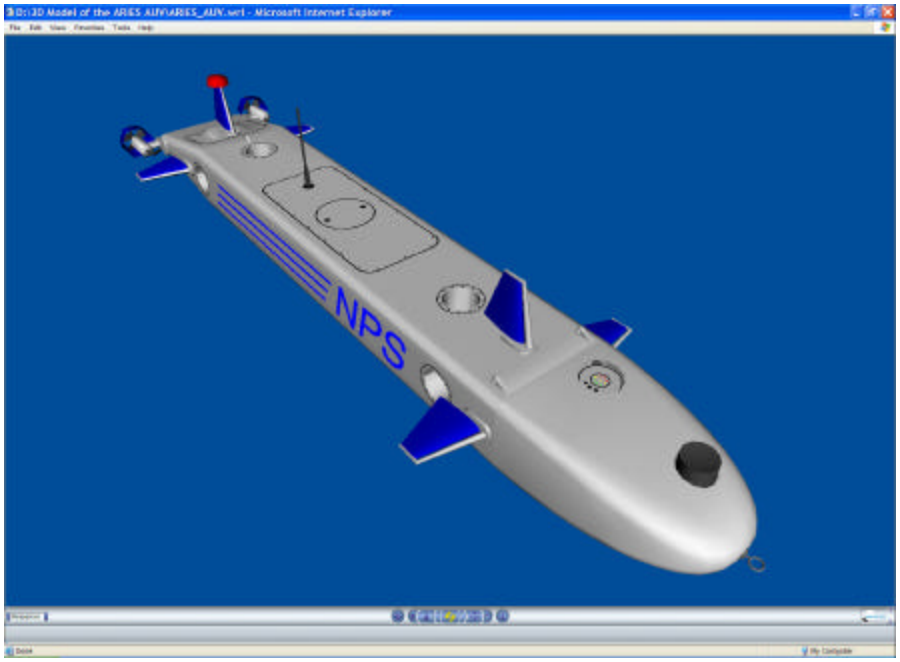

Figure 4. Virtual AUV model of ARIES [Gruneisen 2002]

application and an editor profile configuration file. X3D-Edit enables simple error-free editing, authoring and validation of X3D or VRML scene-graph files.

a. Execution Code

The primary objective of the execution code is to perform real-time data acquisition and control of the vehicle. The execution level software is the heart of the autonomous underwater vehicle. The vehicle execution software is designed to operate both in the virtual world and in the real world. While sensing in the virtual world, distributed hydrodynamics and sonar models fill in the respective telemetry vector values. While sensing in the real world, actual sensors and their corresponding interfaces fill in those telemetry vector values. In either case, the rest of the execution code, which deals with command parsing, dynamics control, sensor interpretation, etc. is unaffected [Brutzman 94]. The interaction of the software with the different actuators and hardware

aboard the vehicle makes the execution level software a very important one. With a hard real-time deadline of 0.1 seconds per sense-decide-act loop, the execution level code

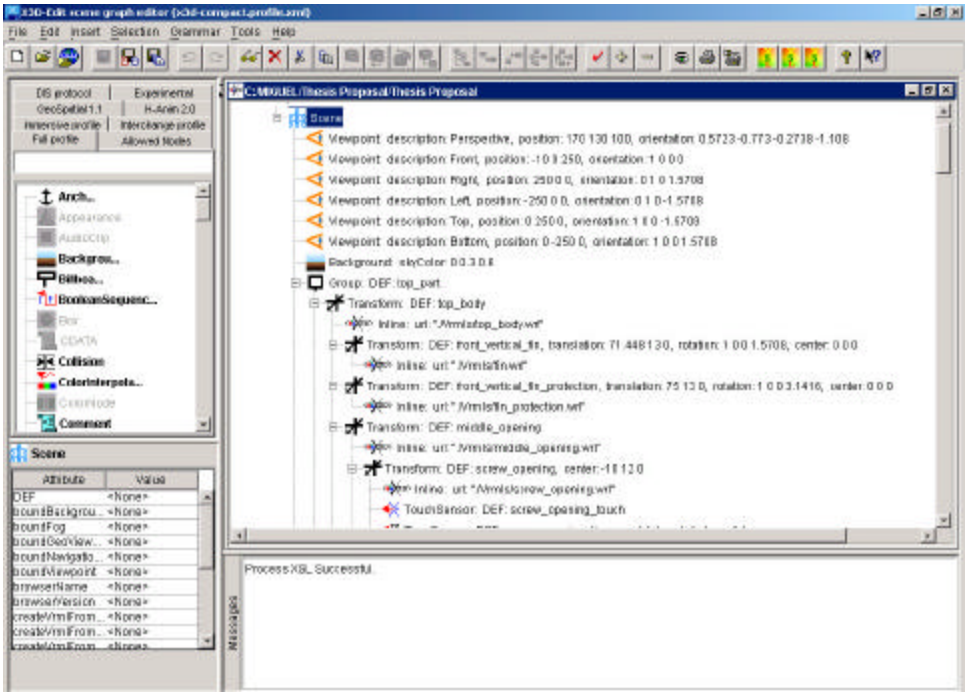

Figure 5. X3D-Edit Graphical User Interface for developing 3D objects and scenes using VRML

reads commands, make decisions and act accordingly sending the appropriate instructions to the different actuators while monitoring the control. It 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. The execution code corresponds to hard real-time reactive control. This layer provides maneuvering control of plane surfaces and propellers. Interface with the different sensors aboard the vehicle is provided through the layer. The execution level integration includes the physical devices control, sense-decide-act, reactive behaviors, connectivity, a mission

script language, and a stand-alone robustness in case of loss of higher levels. Figure 6 shows the existing execution level software/hardware structure.

b. Dynamics Code

The effects of the surrounding environment on a robot vehicle are unique to underwater domain. Understanding these forces is a key requirement in the development and control of the vehicle behavior.

The dynamics program Java source code is designed to substitute for the natural environment effects on the AUV. It also provides an estimate of the AUV behavior in the water by performing a series of calculations using physical laws. By communicating with the execution code via a network socket, the telemetry data or state variables of the vehicle are collected. Dynamics apply several equations of motions, forces, and accelerations to the hydrodynamics model and the data received from the execution code. The data produced by dynamics is then sent back to execution, where it is analyzed and appropriate action commands are then given to the respective actuators based on that data. This is a very important and difficult part in the real-time simulation in a virtual world.

The 3D visualization algorithms in the dynamics code allow the update of 3D scenes developed using X3D-Edit. These scenes are viewed through an internet browser using a plug-in VRML viewer. Many VRML plug-ins are available as 3D browsers, including:

Xj3D Open Source Browser: http://www.web3d.org/TaskGroups/source/xj3d.html

Nexternet: Pivoron player: http://www.nexternet.com

Cosmosoftware: Cosmoplayer: http://ca.com/cosmo

Parallel Graphics: Cortona player: http://www.parallelgraphics.com/cortona

Blaxxun: Contact browser: http://www.blaxxun.com

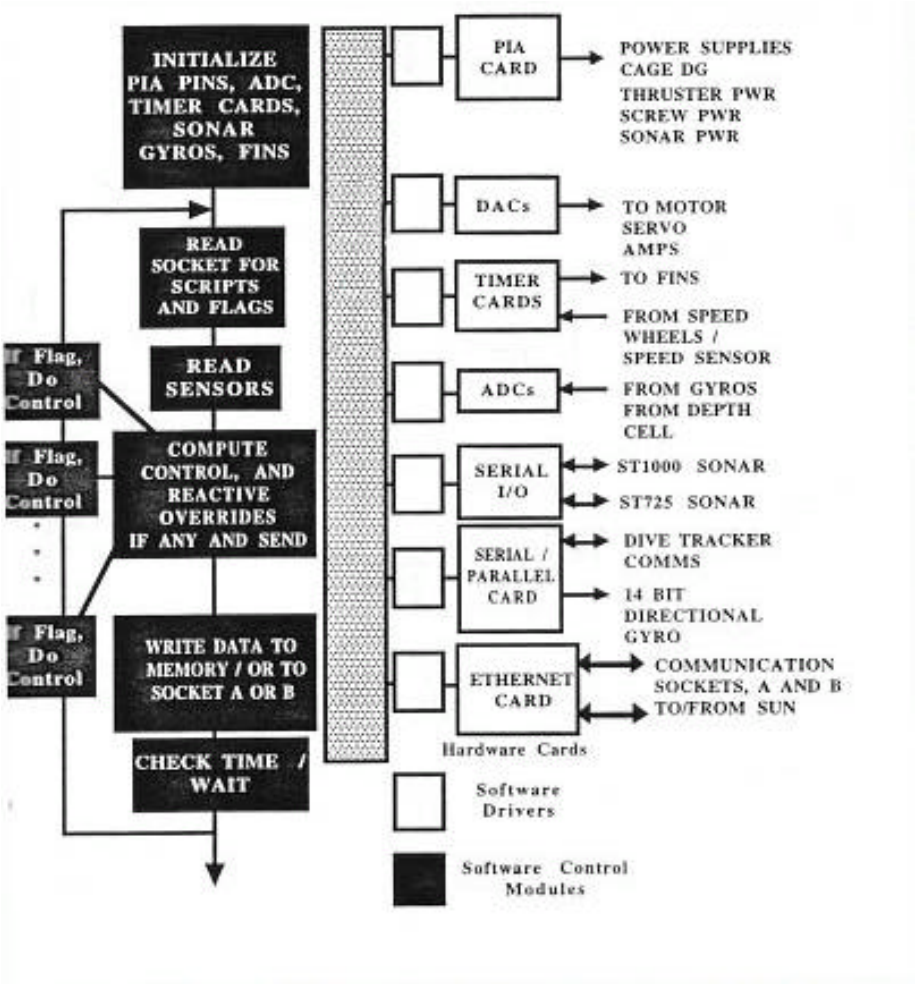

Figure 6. Structure of the Execution Level Software/Hardware [Marco 96]

C. SUMMARY

The development of autonomous underwater vehicles encompasses a numerous challenges. A lot of effort has been dedicated to developing tools that might make the

process of research, developing and testing of AUV a lot faster, safest, and economical. As presented in this chapter, the Naval Postgraduate School ARIES AUV is a product of years of research and development. With the creation of the virtual environment for the vehicle, mission planning, visualization, rehearsal, playback, and analysis can be achieved more quickly. The capabilities for using the virtual world interface with the physical world information provide an excellent combination for post-mission analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PROBLEM STATEMENT

A. INTRODUCTION

Autonomous underwater vehicles have a great potential use for the United States Marine Corps and the Navy. When performing amphibious operations, underwater mines present a danger for the forces going ashore. The use of underwater vehicles for the detection of mines and signaling to the Amphibious Ready Group is very attractive. With advancements in hardware and object-oriented language technology, more sophisticated and robust software can be developed. In this chapter, the problem statement is addressed to provide insights about why Java and Java boards are desirable for an autonomous underwater vehicle.

B. PROBLEM STATEMENT

The Naval Postgraduate School Center for AUV Research, Monterey CA, has been designing, building, operating and researching AUVs since 1987. The NPS AUV series is in its fourth generation. Great advancements have been made since the introduction of the first AUV, but the control software of the vehicle is still written in C. For the next generation of AUV, it is desirable to have a reliable, fast platform-independent source base that can apply modern software engineering techniques, be developed by students on personal computers, and capable of been accessed via the World Wide Web. Since the environment where the AUV operates is a very unforgiving one, the code that operates in the vehicle must be the same as the one of the one operating in the virtual world. In other words, the vehicle must not notice the difference between virtual world operations or in water operations.

C. WHY JAVA AND JAVA BOARD?

Each generation of NPS vehicles has provided substantially increased operational capabilities and sophistication in hardware and software. With the advancement in real-time computer languages support, object-oriented technology and cost-efficient and high-performance hardware, this works proposes for a fifth generation of AUV software. A new underwater vehicle with Java-based architecture can greatly improve reliability,

endurance, and verifiability. Java hardware for in-water components will provide fast performance and low power consumption in which improves endurance. Accessibility to both low-level and advance Java Applications Programming Interfaces (APIs) will make the software robust and easy to maintain, contributing to low maintenance costs.

Over the past several years the computing community has been coming to widely accept the Java platform, a technology triad comprising a relatively simple Object Oriented Language, an extensive and continually growing set of standard libraries, a virtual machine architecture and a portable bytecode class file format that provides portability at the binary code level. Java's promises of "write once, run anywhere" has increased the platform's application domain. Java is a fully object oriented programming language with strong support for proper software engineering techniques like the building block approach to creating programs and reuse of already created. Java is more secure than C and simpler than C++. Unlike C and C++, Java has a built-in model for concurrency (threads) with a low level "building blocks" for mutual exclusion and communication that offer flexibility in the design of multi-threaded programs. Parts of Java (APIs) address some real-time application areas like javax.com for manipulating serial and parallel devices and SDK 1.3 for the timed event classes. The integration with a native Java hardware board reduce the dependence on external components and provides high data transfer. Java byte code will be directly executed on the board, so there is no need of Java Virtual Machine, providing a small footprint, and with the real-time environment offered by the processor and development tools offered with the package, a complete development solution is provided. The Java board will provide cost

effective embedded applications by improving performance and reducing power consumption, a key parameter of performance in robotics development.

D. GOALS

The area of concern of this thesis is the execution level code. The primary goals for this work are:

- To provide a good set of software requirements specifications documents enabling a programmer to quickly start programming according to specifications without major problems.
- To build a Java-based execution level capable of interacting with virtual world and real world without any noticeable difference, in other words, that the AUV software could not notice the difference if it is operating in the water or the virtual world.
- To prepare AUV software for Real Time Java migration and Java board hardware integrated solution in the near future.

E. SUMMARY

With the advancements in software developing languages technology and the increasing speed of current processors, the opportunity to create more sophisticated processes is available. Complicated software like the one found on autonomous underwater vehicles can take advantage of these technological improvements. In this chapter we presented the problem statement developed based on the arguments discussed in this chapter. We also provided an overview of Java and Java.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SOFTWARE ENGINEERING

A. INTRODUCTION

Many new computer scientists, programmers and even project managers find themselves as new arrivals to a software development team. Usually they are not fully aware that a software product is just one major component of what is usually a complex system of hardware, people, software and procedures. In the software development process probably the two most significant steps are the determination of (1) precisely what the software product must do and (2) how, specifically, the software product will accomplish its task. The results in these two steps are manifested in two critically important specifications: the software requirements specifications and the software design specifications.

Software development currently suffers from three major deficiencies: (1) software engineering principles and practices, which should be the backbone of the software development life cycle, are not fully accepted and followed; (2) straightforward, well established, and universally accepted design standards are lacking for the software development process and for representation of both process and product; and (3) the software development process is empirical in nature and not yet predicted by easily quantified or confirmed mathematical models.

The effect of these and other deficiencies has been amplified by the rapid growth in the volume of software being produced and the almost exponential increase in the complexity of the problems now being solved with software. In spite of the need for a strongly structured approach to software development, in many organizations software development is still basically a freestyle event.

In this chapter an overview of these processes and their application to the software development for the NPS AUV Execution Java code will be discussed.

B. SOFTWARE DEVELOPMENT CYCLE

A software development life cycle is a framework composed of a sequence of distinct steps or phases in the development of the software. It attempts to create an ordered structure to the software development process. Each phase consists of a set of

related activities usually culminating in a product that could be a document or a review in which contributes to the completion of the software product. There are several software development models like the Waterfall model and the Spiral model. A generic software development model will typically go through these phases:

- Analysis
- Design
- Coding
- System Integration
- Testing

1. The Waterfall Model

As shown in Figure 7, the software activities proceed logically through a sequence of steps. Each step bases its work on the products from the previous step. Design logically follows requirements, coding follows design, and so on. The waterfall model has been widely used over the past two decades and has served successfully as a process model for a variety of medium-scale to large-scale software projects.

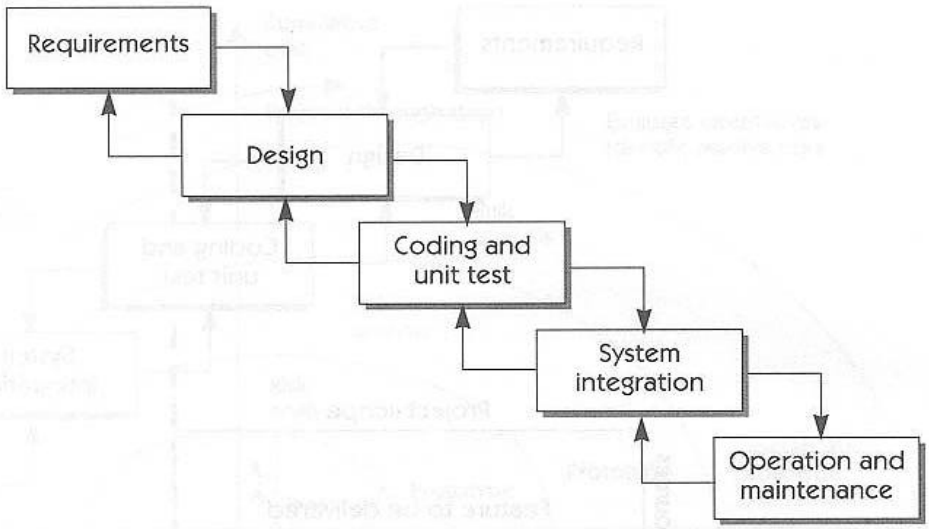

Figure 7. The Waterfall model for software development [Leffingwell & Widrig]

2. The Spiral Model

In the spiral model, shown in Figure 8, development is initially driven by a series of risk-driven prototypes; then a series of waterfall interactions are used to produce the final product. It provides a good structure that helps to address some of the requirements challenges. Specifically, the spiral model starts with requirements planning and concept validation, followed by one or more prototypes to assist in early confirmation of the understanding of the requirements for the software system. The main advantage of this process is the ability of multiple feedback opportunities with the user and customers.

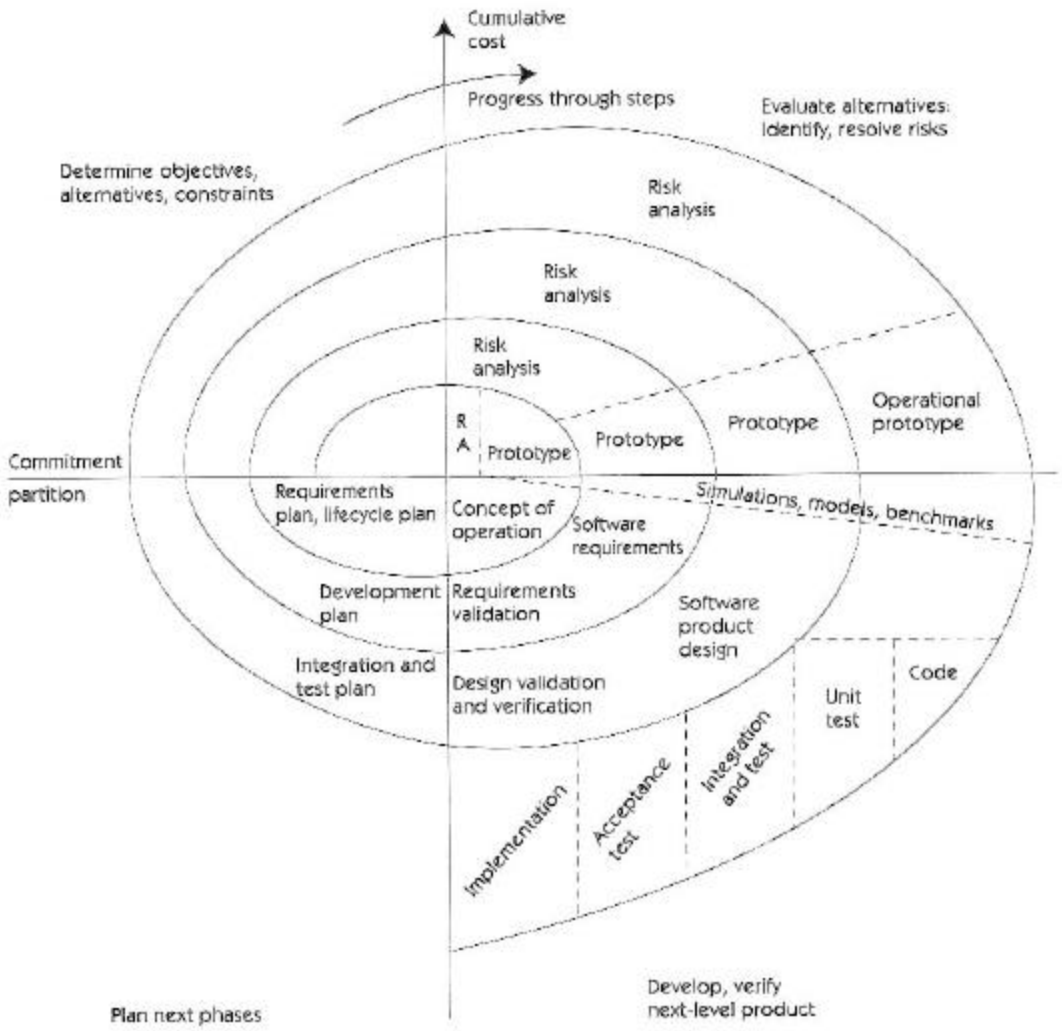

Figure 8. The Spiral model for software development [Leffingwell & Widrig]

C. ANALYSIS AND SPECIFICATIONS

1. Requirement Elicitation and Analysis

Performing a requirements or problem analysis takes time. The goal is to gain a better understanding of the problem being solved before development begins. After defining the problem, a clear problem statement was stated. Part of the analysis is to identify the stakeholders and users of the system in development. At this point the AUV is a vehicle for student research. The demographic background of the users is very diverse. Generally United States Navy officers constitute the great majority of the stakeholders. Officers with a variety of academics backgrounds but with the commonality of coming straight from the Fleet provide the military expertise needed for the vision of AUV use in military applications. These officers are from different academic backgrounds. Faculty stakeholders will be the permanent ones. Participants from the faculty generally come from the Mechanical Engineering Department, Computer Science Department, Electrical Engineering Department or Undersea Warfare Department. As discussed above, the development of the software must encompass this variety and constant rotation of users. Ease of use and maintainability are important approaches to follow for the development of the software system.

Since the AUV is a complex system, composed of various software components, it was very important to define a solution system boundary. Once the problem statement was identified and agreed, a system boundary was traced in order to successfully manage the project. For this project the execution level code was going to be converted from C to Java with the first phase to be the virtual world code.

As part of the requirements analysis, a requirements elicitation process was conducted. It is a simple, direct technique that can be used in essentially any design situation. The purpose of the process for this project is to gain better understanding of the execution level code. The process was conducted during the early stages of the project. Interview with two faculty stakeholders was done. Insights about the functionality of the existing AUV and desire features for the next generation were given. A set of questions was prepared prior to the interview process. A good time was spent in literature review trying to get an understanding of the system and its complexity. The

initial interviews provided a good and broad information and produced more questions about the system. A second round of interviews were done in order to clarify and answer new questions that surfaced while analyzing the information produced during the first interviews. The second interviews also provided the opportunity to go to preliminary use cases and a storyboard in order to get feedback from the stakeholder.

2. Software Requirements Specifications

Every engineered and manufactured product must be specified in some fashion. As a product becomes more complicated, it will require more detail specification. A given product may be specified in terms of its behavior and performance. Such specifications are called operational specifications. A product may also be described by its effect on its environment and by its properties. Such specifications are called descriptive specifications. Descriptive specifications tend to define a product in terms of its output whereas operational specifications tend to describe a product by its operational performance.

The specifications for the NPS AUV are a mixture of operational and descriptive specifications. The most important specification for the execution level code is:

- Sense-decide-act control loop with a hard real-time of 0.1 seconds or 10Hz

Any software requirements specifications document must be done without ambiguity. Several methods can be used in order to accomplish the requirements specifications document.

Formal methods can provide tools used in the specification of software requirements. Formal specification uses a language with a mathematically defined syntax and semantics, usually predicate logic. The kinds of system properties might include functional behavior, timing behavior, performance characteristics, or internal structure.

So far, such specification has been most successful for behavioral properties. One current trend is to integrate different specification languages, each able to handle a different aspect of a system. Another is to handle non-functional aspects of a system such as its performance, real-time constraints, security policies, and architectural design. Some known formal methods are Z [Spivey] and SPECS [Luqi].

Recently, with the introduction of the Unified Modeling Language (UML) as a standard graphical language for visualization, specifying, construction, and documentation of software systems, the specification of software requirements is becoming easy. UML provides a standard way to write a systems' blueprints, covering conceptual issues, such as business process and a system functions, as well as concrete programming tasks, such as classes written in a specific programming language, and reusable software components.

D. SOFTWARE DESIGN

1. Design Steps

Ideally, it is desirable to have a generic software design process that is independent of any particular programming language or specific design tool, which applies to the full range of software products. The software design process is the sequence of steps that systematically transform the function and performance requirements (requirements specifications documents) to pseudocode, UML diagrams, or any form of Programming Design Language (PDL) suited for coding.

The existing AUV execution level C code is very complicated [Burns 96], [Marco 96]. The lack of comments and the use of global variables makes it even more complicated. Several months were spent analyzing the code in terms of functionality and traceability. Generally, several guidelines were taken to derive a good design.

a. Function Decomposition

Basically the partitioning of the software product into components or design entities. Some criteria and/or constraints for partitioning followed are:

- Unit testing improvement
- Modularity
- Information hiding
- Data dependency
- Function separation

With the modularization, the cohesiveness and the coupling of the modules were improved, basically by trying to achieve highly cohesive modules while low coupling. Modules that have clean interfaces, and that have all the necessary means to perform their task, without undue influence from other modules, are most reliable.

b. High Cohesion and Low Coupling

Well-designed modules must have clearly defined, precisely named and carefully typed interfaces. In terms of object-oriented design, coupling is a measurement of how strongly one class is connected to, has knowledge of, or relies upon other classes. Low coupling means not dependent on too many other classes. A module with high coupling relies upon many other modules. Such modules are undesirable since changes in related modules force local changes and furthermore such modules are harder to reuse because its use requires the additional presence the modules it is dependent upon.

Cohesion is a measure of how strongly related and focused the responsibilities of a class are. A class with highly related responsibilities, and which does

not do a tremendously diverse amount of work, has high cohesion. Highly cohesive modules are desirable. A module with low cohesion does many unrelated things or attempts to do too much work. Low cohesive modules problems include hard to comprehend, hard to reuse, and hard to maintain.

The existing C code was designed with very low cohesion and very high coupling. Thus a lot of redesign work is needed to achieve high cohesion and low coupling.

c. Data Definition

The selection of the appropriate data types, data structures, files definition and network communication types is fundamentally important. For the AUV a simple queue was developed to hold all the commands being read from the mission script. The decision to use a queue instead of keeping a file open was the simplicity, fast access to the commands and minimized dependence of the hard drive for maximum independent reliability.

2. Use Cases

A use case is a narrative document that describes the sequence of events for an actor (an external agent) using a system to complete a process [Jacobsen 92]. A use case thus describes a sequence of performed actions that yields a result of value to a particular actor. They are not exactly requirements of functional specifications, but they do illustrate and imply requirements in the story they tell. A use case is roughly the same as a function point, i.e. a cohesive piece of functionality of the system that is visible from outside. Use cases are a form of functional, rather than object, decomposition.

a. Sense

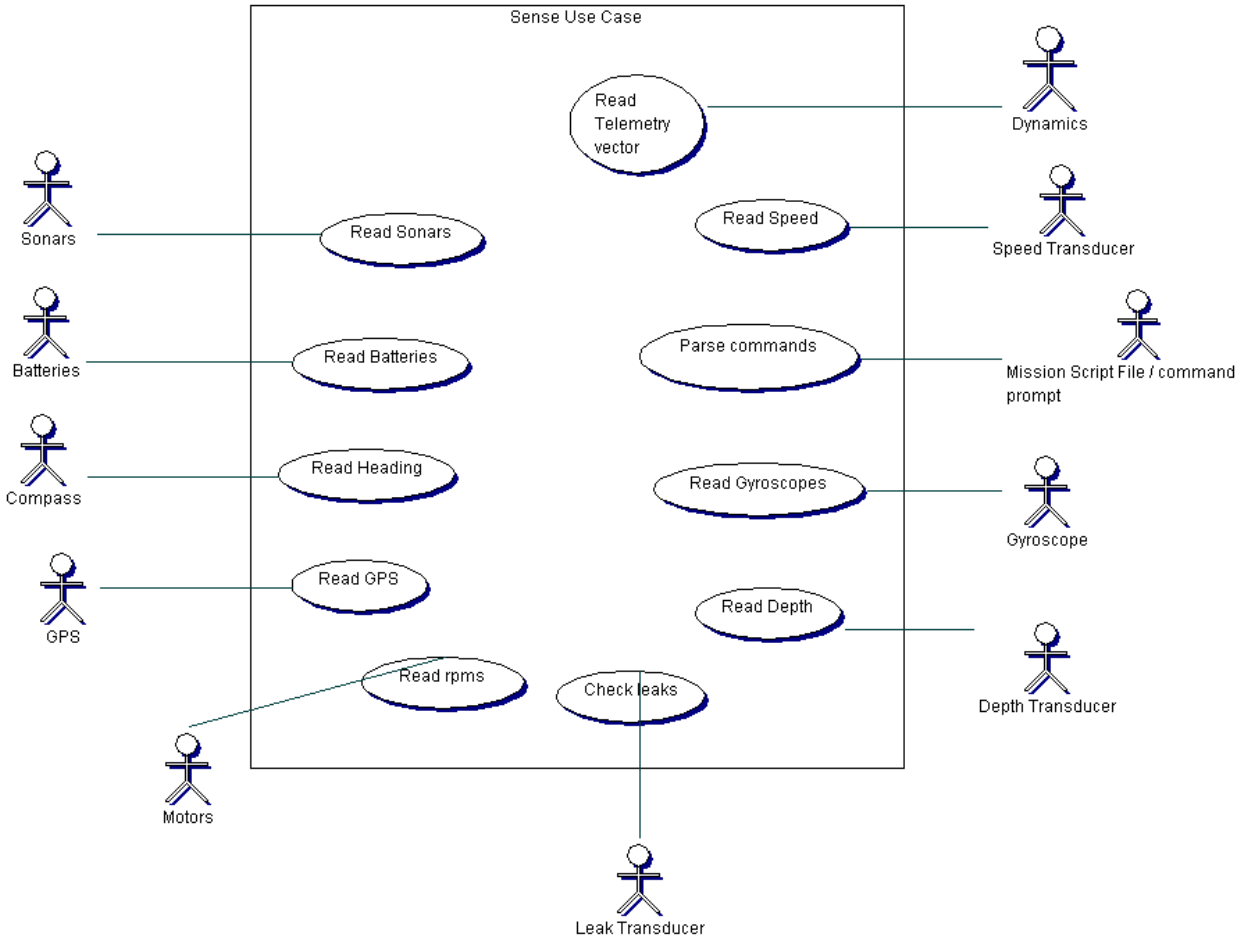

Figure 9. Sense Use Case as UML Diagram

Use Case:	**Sense**
Actors:	Control loop class, sonars, batteries, leak sensor, speed sensor, depth transducer, GPS, Gyroscopes, motors, fins, compass, mission script, command prompt, Dynamics.
Purpose:	Collect information about the environment through the different sensors aboard the vehicle. These sensors are the eyes and ears of the vehicle.
Overview:	The autonomous underwater vehicle control loop class located at the execution level is responsible for the process of polling information.

Type: Primary and essential

Typical Course of Events

Actor Action	System Response
1. Parameters from the command prompt are provided.	2. Parameters are read and matched against a set of predetermined flags hardwired in the software. If a match occurs, then the appropriate flag is set.
3. Control loop class calls the method to check the voltage in the batteries. A circuit board provides the voltages of the batteries.	4. If batteries (motors and computer) < 20.0 volts then call the safe shut down method.
5. Method to check for leaks in the AUV is called. The leak transducer provides a voltage signal.	6. The voltage is compared against parameter established. If the result is outside the parameters then leaks are found inside the AUV and proceed to call the safe shut down method.
7. Method to check for the depth is called. The pressure transducer provides a voltage signal through an analog to digital card. That voltage represents the depth of the vehicle. The transducer must be properly calibrated.	8. If depth is > 6.0 meters then calls the safe shut down method.
9. Control loop method calls the read sonars methods. The sonars return any contacts made. Two classes of sonars, for contacts and the other one for surfaces sweep.	10. If a contact with an object is made, the information is saved like location and distance of the object. If an object is found at a distance < 3.0 meters from the vehicle then all stop. Collision avoidance procedure is executed.
11. The speed sensor reads the speed of the AUV by mean of an acoustic Doppler. The velocity is already in meters/sec. The velocity is read as a vector form, in its three components: u, v, and w. The information coming from the Doppler comes out at a rate of 2 Hz or one reading every 0.5 seconds.	12. Take the speed attribute of the AUV and save it as part of the telemetry information.

13. Rpm sensors on the starboard, port and thrusters motors provide the rpm of each motor.	14. Calculation of the mean rpm from the starboard and port motors. The result is saved as part of the telemetry information. Rpms from thrusters are taken individually and saved in the telemetry information.
15. The method that read the gyroscopes is invoked. Mechanical gyros provide output in the form of voltage signal. The information coming from the gyros are available every 100 Hz or 0.1 seconds.	16. Conversion from analog to digital is done through an analog/digital (A/D) conversion card. The information is perceived as rad/sec. Roll, Yaw and Pitch rates recorded.
17. Proceed to read GPS information. The GPS information is available every 1 second or 1 Hz.	18. The information is processed and saved as part of the telemetry data. The position is then use the calculate future location, distance to travel and time based on current speed. The information obtained is true Latitude and Longitude. It is compared to a GPS zero (origin of the mission) in terms of meters North and East. A computation of distance travel in meters from the origin is performed.
19. The Compass interacts with the method responsible for reading the information about the heading. This information is available every 10 seconds.	20. Record the information as part of the telemetry data. Compass heading is compared with the integration values of the numbers coming from the gyros and dead reckoning procedures.
21. Commands from the mission script are read from mission script in the hard drive.	22. Commands from the mission script are parsed. A matched command is given to the Act class for action.

Table 1. Sense Use Case Typical Course of Action

b. Decide

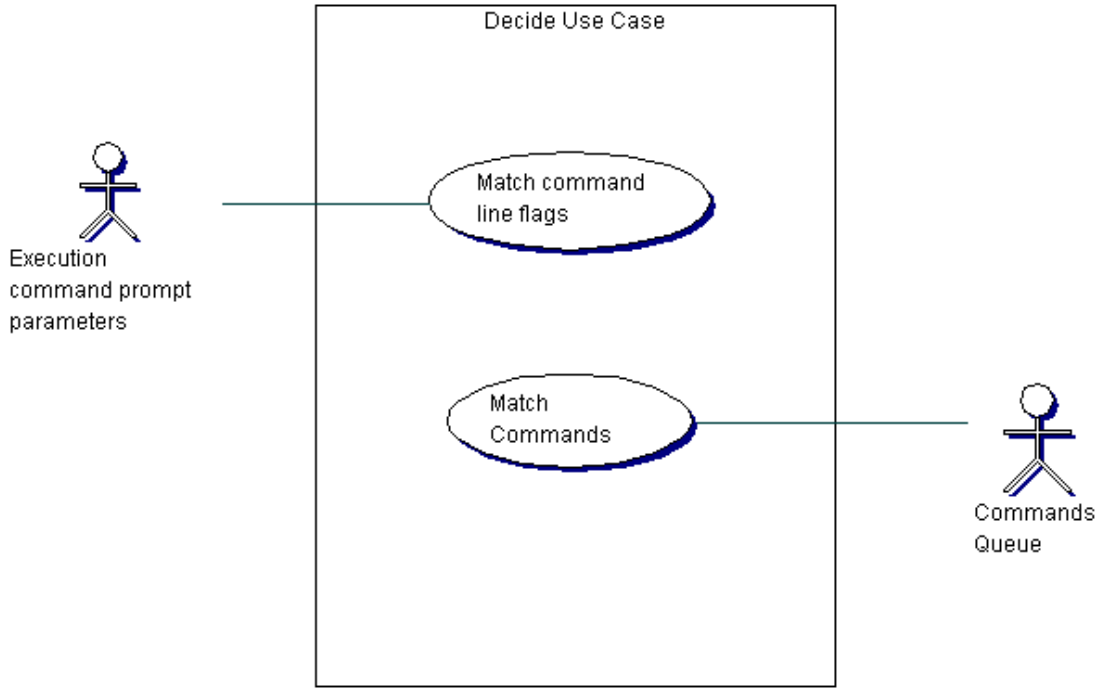

Figure 10. Decide Use Case as UML diagram

Use Case:	**Decide**
Actors:	Execution command prompt, Commands queue
Purpose:	To compute the necessary control commands required to successfully navigate the vehicle.
Overview:	New commands are received or execution continues on prior commands. Based on the information polled from the sensors, a series of computations and comparison are performed. Decisions are made with respect to the appropriate commands to the respective control surface in order to achieve the desire navigation path.
Type:	Primary and essential

Typical Course of Events

Actor Action	System Response
1. Provides the parameters string containing the flags to be set.	2. Read the flags string, parse it and match the flags. Once a flag is match then the respective value of is set to true or false.
3. Provides a command string.	4. Read the command string, parse it and match the command with its respective parameters. Based on the command read a corresponding action is performed.
5. Provides the voltages read from the computer battery.	6. If computer battery voltage < 20.0 then the safe shut down method is called and the mission is stopped.
7. Provides the voltage read from the motors.	8. If motors battery voltage < 20.0 then the safe shut down method is called and the mission is stopped.
9. Provides voltage read from the leak check transducer.	10. If the voltage is > 1.0 volts then a leak has been detected therefore proceed to call safe shut down method.
11. Provides the range in meters of objects pinned by the sonar	12. If the range is < 3.0 meters then shut-down the motors. Perform shutdown procedure.
13. Rpms from the starboard and port motors are provided.	14. Compute motor control commands based on the values provided.
15. A mission command is given from the Sense class	16. Compute navigation controls based on the values obtained from the command. This controls include: • Hover control • Docking control • Target control • Waypoint control • Lateral control • Rotation control • Recovery control • Lateral thrusters control

Table 2. Decide Use Case Typical Course of Action

c. Act

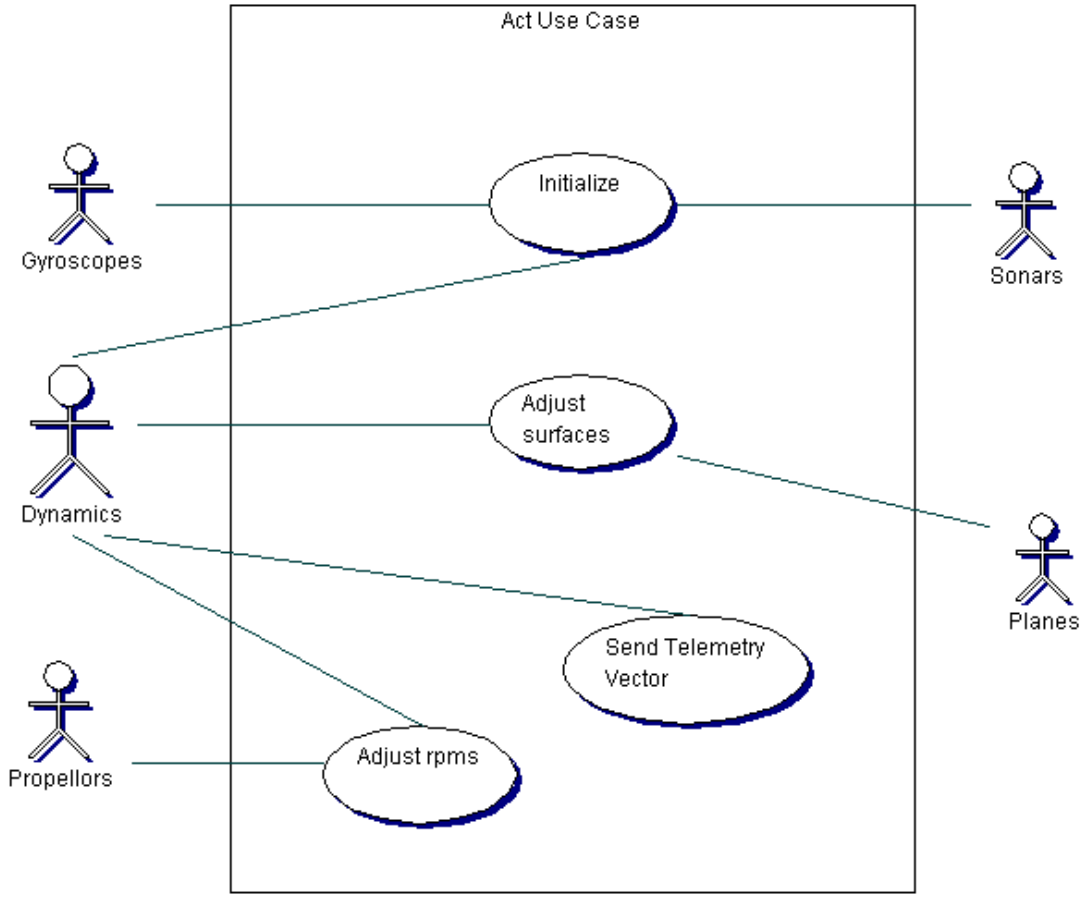

Figure 11. Act Use Case as UML diagram

Use Case:	**Act**
Actors:	Gyroscopes, propellers, Sonars, planes, Dynamics.
Purpose:	To control the navigation of the AUV.
Overview:	Commands are sent to the different actuators in order to correct a position or speed.
Type:	Primary and essential

Typical Course of Events

Actor Action	System Response
1. Provides the command call necessary to effectively and successfully control the vehicle.	2. If computer battery voltage < 20.0 then the shut-down method is called and the mission is stopped.
	3. If motors battery voltage < 20.0 then the shut-down method is called and the mission is stopped.
	4. If the voltage is > 1.0 volts then a leak has been detected therefore proceed to call shutdown method.
	5. If the range is < 3.0 meters then shut-down the motors. Perform shutdown procedure.
	6. If a successfully command is received from the parsing function, compute the necessary command in order to achieve the desire result.
	7. Communicate with dynamic by sending a telemetry string.

Table 3. Act Use Case Typical Course of Action

d. Control

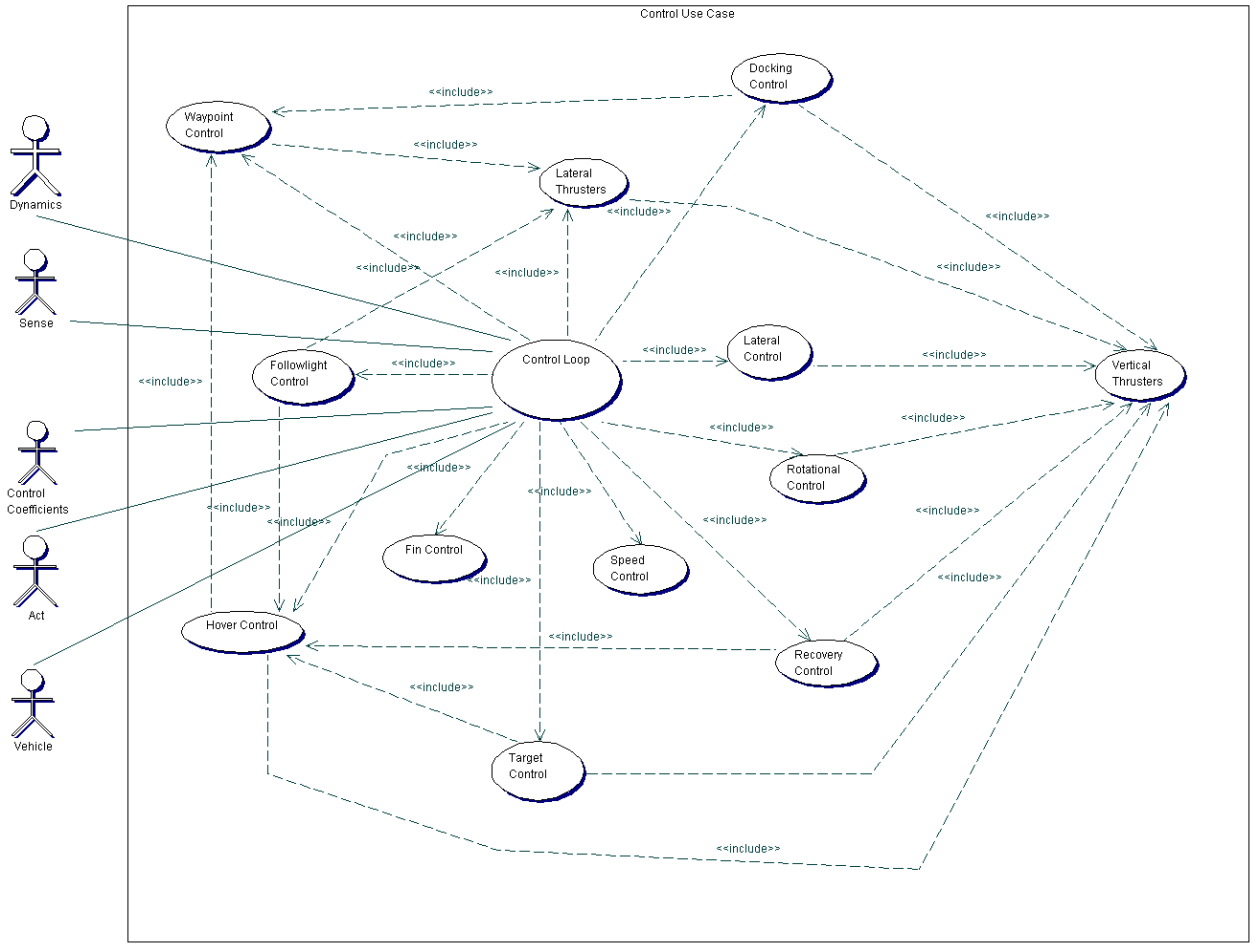

Figure 12. Control Use Case as UML diagram

Use Case:	**Control**
Actors:	Dynamics, Sensors, Control Coefficients, Act, Vehicle
Purpose:	To control the navigation of the AUV.
Overview:	All the corresponding control calculations are performed here. Computations of hovering, waypoint control, target, speed control, and fin control for example depends of some other calculations like lateral control and vertical thrusters control.
Type:	Primary and essential

Typical Course of Events

Actor Action	System Response
1. Provides with the corresponding flag that specify what type of control will be computed and performed.	2. If Hovering flag is set, compute hover control.
	3. If Followlight flag is set, compute followlight control.
	4. If Docking flag is set, compute docking control.
	5. If Target flag is set, compute target control.
	6. If Waypoint flag is set, compute waypoint control.
	7. If Lateral control flag is set, compute lateral control.
	8. If Rotate control flag is set, compute rotate control.
	9. If Recovery flag is set, compute recovery control.
	10. If Thruster control flag is set, compute thruster control.
11. Perform the necessary action to execute the control.	

Table 4. Control Use Case Typical Course of Action

3. UML Diagrams

a. Conceptual Model

Figure 13 shows the conceptual model for the AUV. A conceptual model illustrates the different categories of things in the domain. The vehicle class, which contains all the telemetry information like position, speed and mission elapsed time, uses the control class in order to navigate. The control class acts like a controller, which is the

heart of the vehicle. This Control class has all the methods to compute the required control parameters for updating the trajectory of the vehicle, speed and depth. This update happens every 0.1 seconds. An aggregation is shown between the Control class and the Control Coefficients class. This is used to model a kind of association in which the whole-part relationships between the two classes. The whole is the Control class (known as the composite) and the part or component is the Control Coefficient class. The Control class can have one or more Control Coefficient classes as shown in the multiplicity notation but only one Control class will be in existence.

Sensors and actuators are shown as abstract classes. The basic functionalities of this devices are the same, a sensor senses the environment and an actuator responds to a command to act into something. The differences are in the applications. Abstract classes are perfect to collect the common functionalities. An implementation of this class will guarantee those basic functionalities and will provide flexibility to extend and add functionalities as required for a more specific sensor or actuator. It is also shown that the actuators and sensors will have digital-to-analog cards for the actuators and analog-to-digital cards for the sensors. The same concept of the whole-part idea applies here.

The dashed lines represents the system boundaries of the Execution level code.

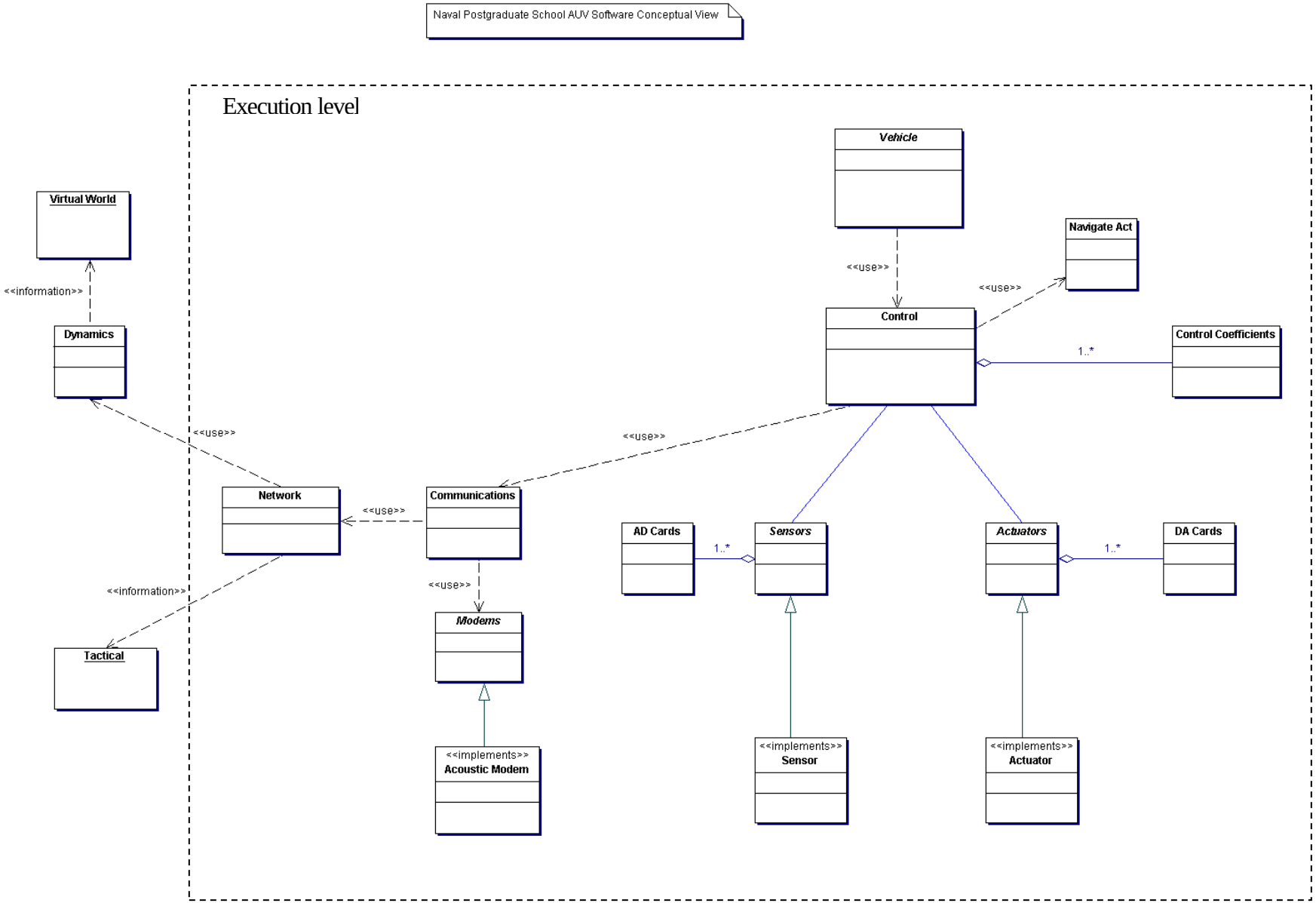

Figure 13. NPS AUV Conceptual Model

b. Sequence Diagram

The use cases suggest how actors interact. During this interaction and actor generates events to a system, requesting information or operations to be performed. It is desirable to isolate and illustrate the operations that an actor request of a system, because they are an important part of understanding system behavior. A sequence diagram is a representation of actor interaction and the operations initiated by them and the system responses.

Figure 14 shows a high-level sequence diagram of the overall interactions between the different actors on the AUV. It is not intended to cover every aspect of the complex operations but to summarize the operation.

Figure 15 shows a high-level sequence diagram of interactions between execution and dynamics.

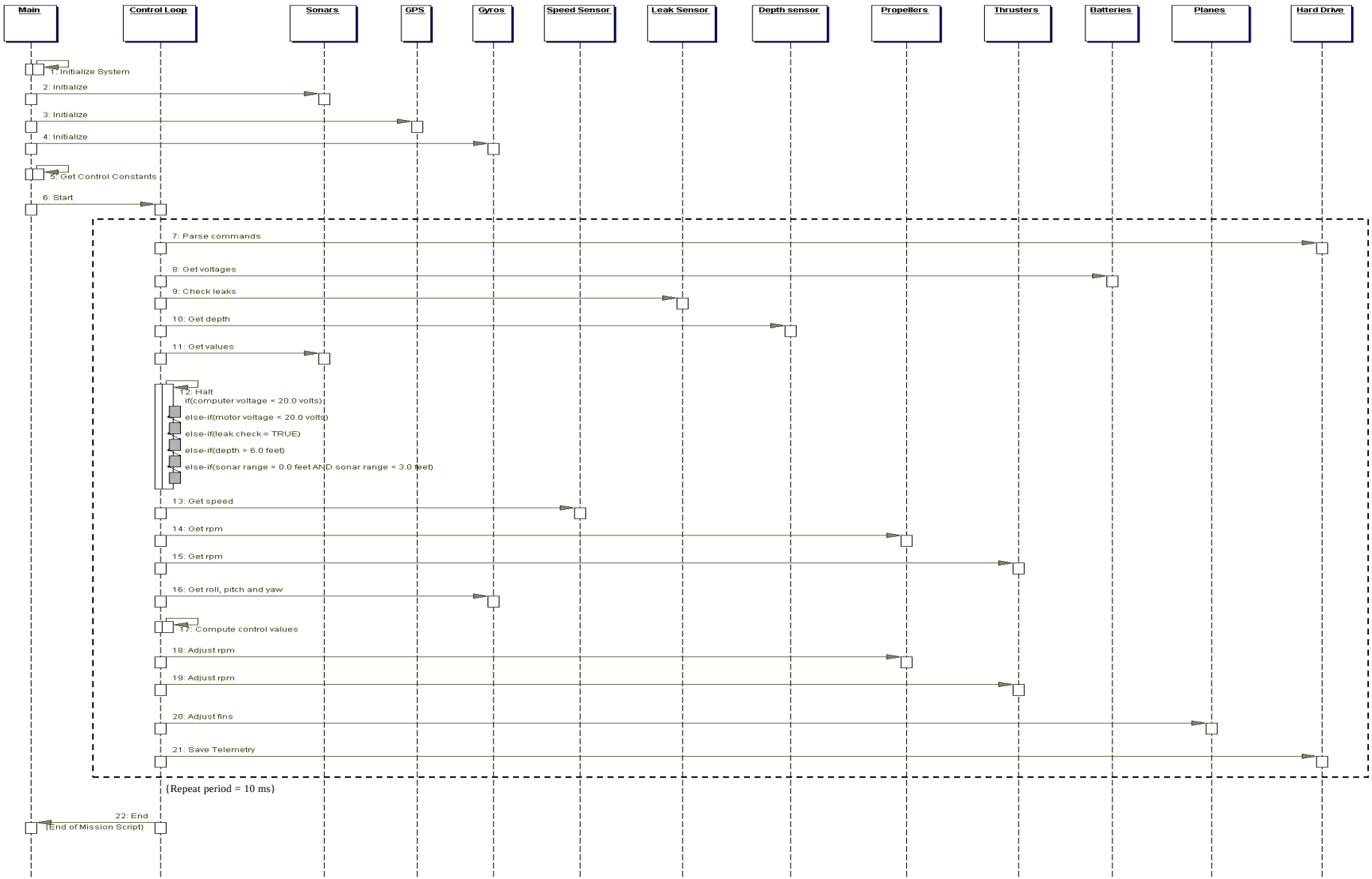

Figure 14. Sequence Diagram

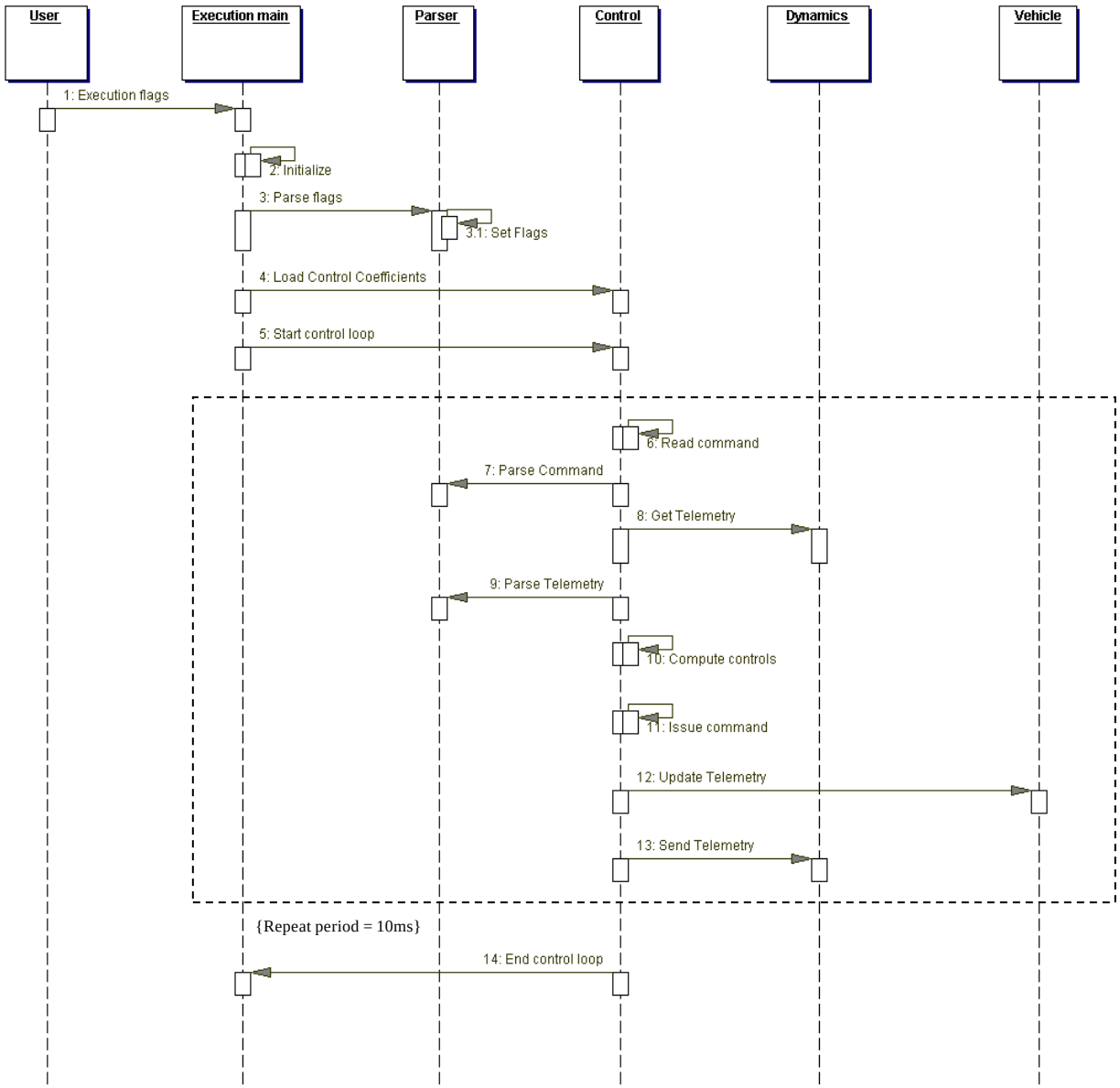

Figure 15. Execution-Dynamics Sequence Diagram

E. SOFTWARE ARCHITECTURE

As the size and complexity of the AUV software increases, the design and specifications of the overall system structure becomes a more significant issue than the choice of algorithms and data structures. The organization and composition of a system is affected by the composition of components, control structures, protocol of communications, synchronization, and data access. The assignment of functionality to design elements, the composition of design elements, the distribution of those elements, and the performance are some of the parameters taken in consideration at the time of developing the architecture.

As mentioned in the design section of this chapter, the selection of the correct decomposition will create a robust, easy to maintain and reuse software code.

A typical robotic cycle is composed of a sense-decide-act executive cycle. Based on that concept a set of respective classes were developed that could actually map the concept. A Sense class, which would be responsible for all the reading of sensors and information, like reading depth and speed. The Decide class would be responsible for the determination of the execution flags in terms of matching them with the respective pre-defined flags, matching the commands read from the mission script and setting also the right parameters in order to act upon that command read. The Act class is the responsible for issue the respective command and interface with the actuators based on the command read. Finally the Control class is the one responsible for all the computation of control parameters required to successfully navigate the vehicle. A series of supporting classes also created but not discussed in this chapter.

1. Sense

Figure 16 shows the dependencies of the Sense class. Each class is shown inside the respective package. Each class is a high cohesive module and low coupling between the classes exist. The figure takes a look at a level of abstraction where unnecessary details are not shown. The dependency of the classes are basically in the interchange of information.

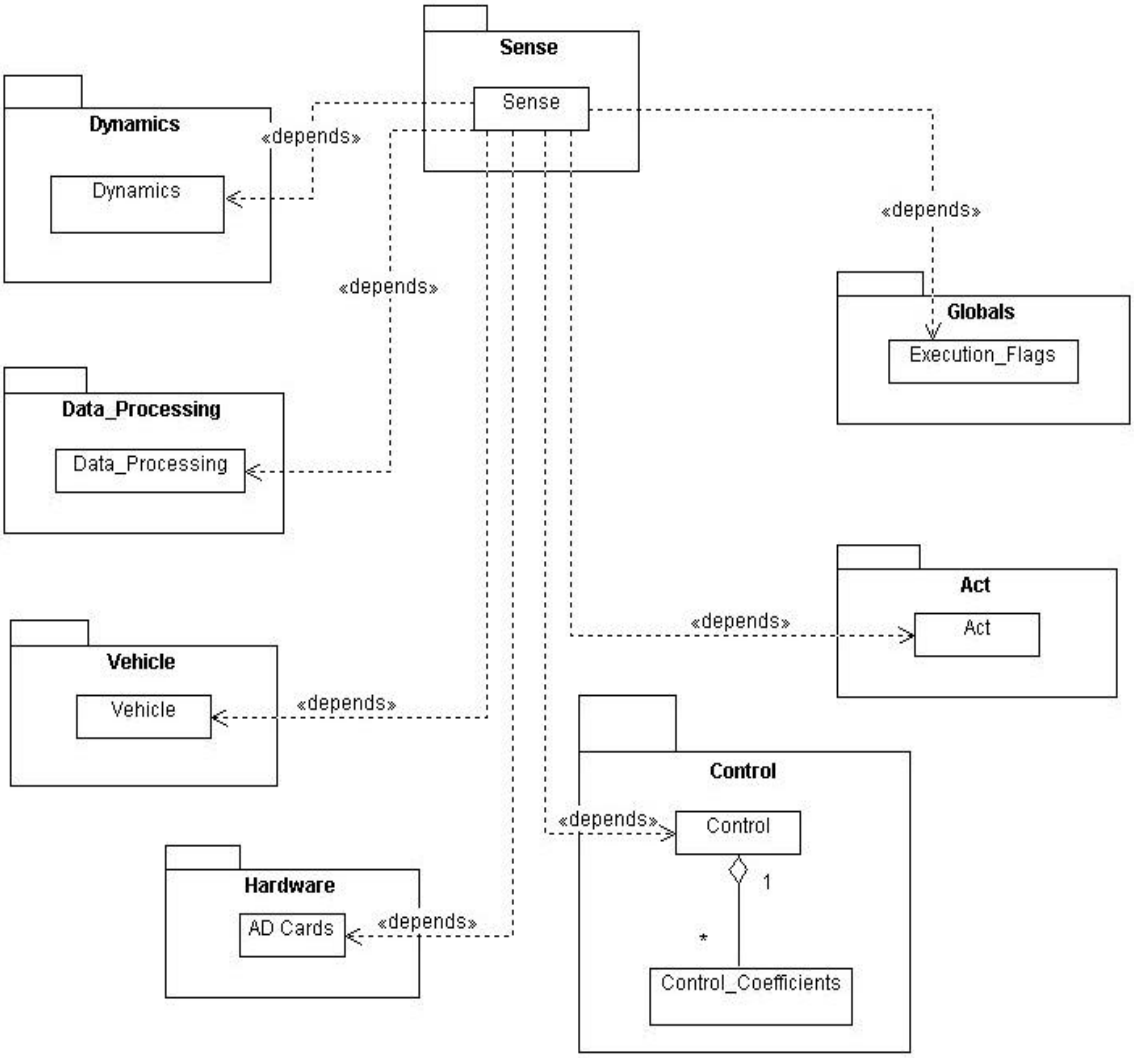

Figure 16. Sense Architecture

2. Decide

Figure 17 shows the Decide class dependencies as a class diagram. Once the information is passed as parameters to the Decide class, they are processed through the Data_Processing class, then required flags are set. The information is sent to the Control class for the computation of the required control parameters. Based on the control parameters the required act commands are invoked and communication with dynamics through the Network class is established.

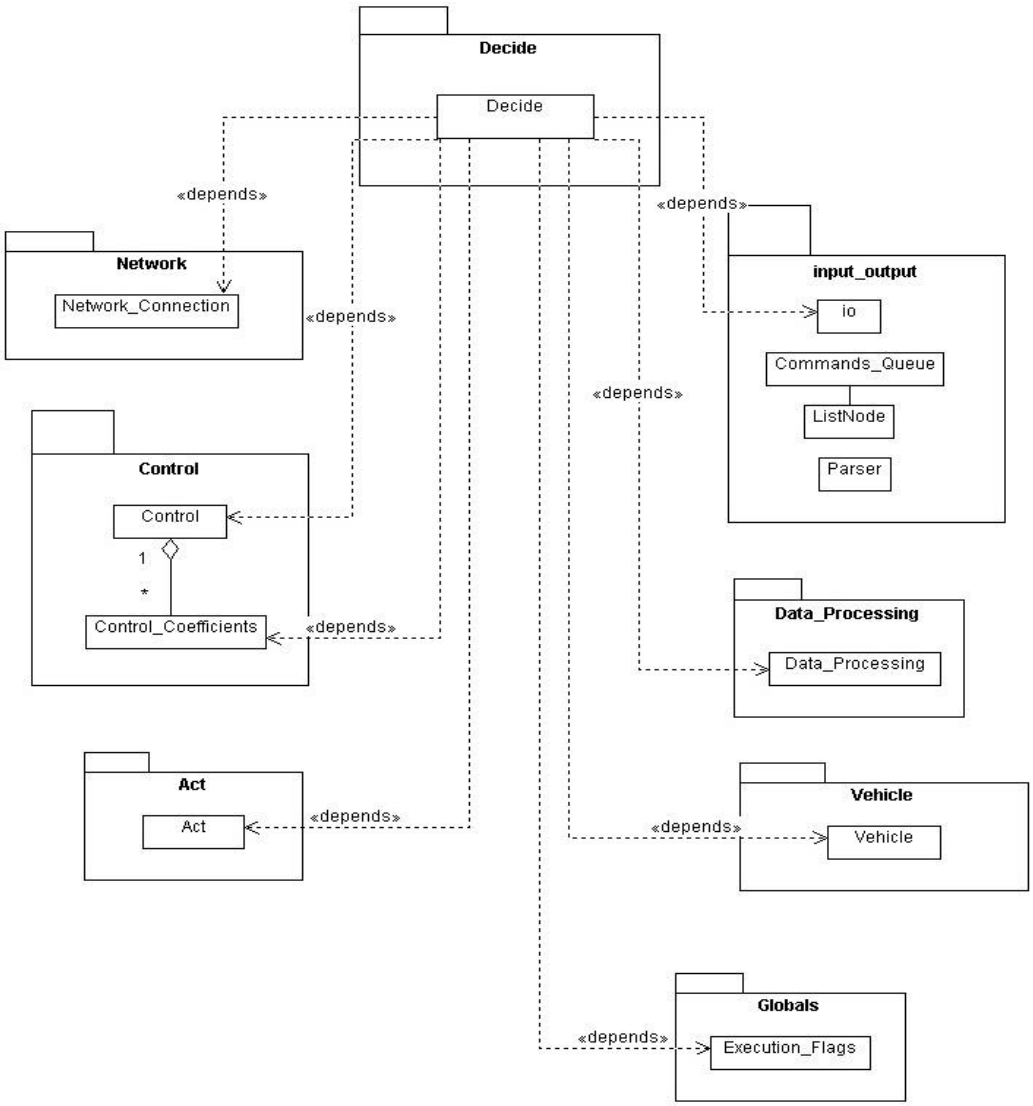

Figure 17. Decide Architecture

3. Act

Figure 18 shows the Act interaction with its dependency in a class diagram. The Act class will update flags on the Execution_Flags class, command necessary controls as required and compute them. Depending on the vehicle location, the command is sent to the digital-analog cards of the corresponding actuator or to dynamics in the virtual world.

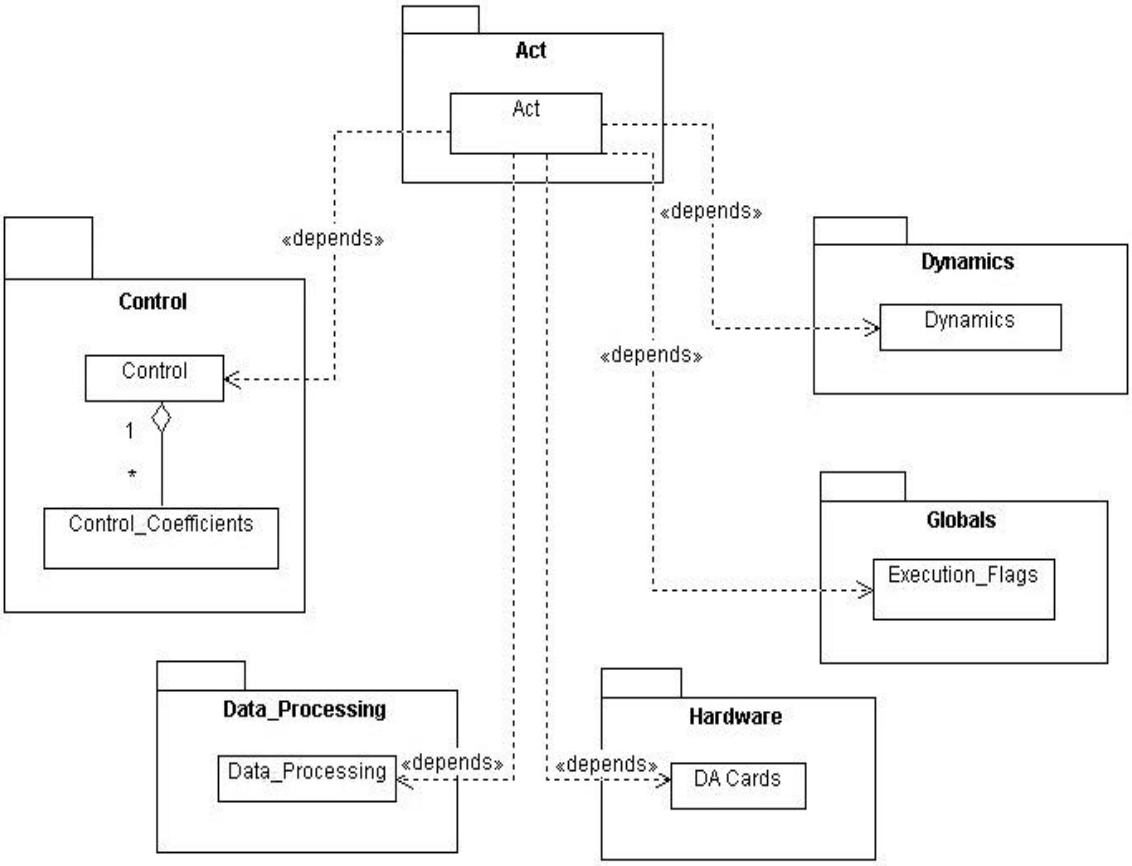

Figure 18. Act Architecture

4. Control

Figure 19 shows the Control class architecture or dependencies in a class diagram. The control class is the driven mechanism of the execution level code. As shown, it have communications with several classes that are essentials in the vehicle operation.

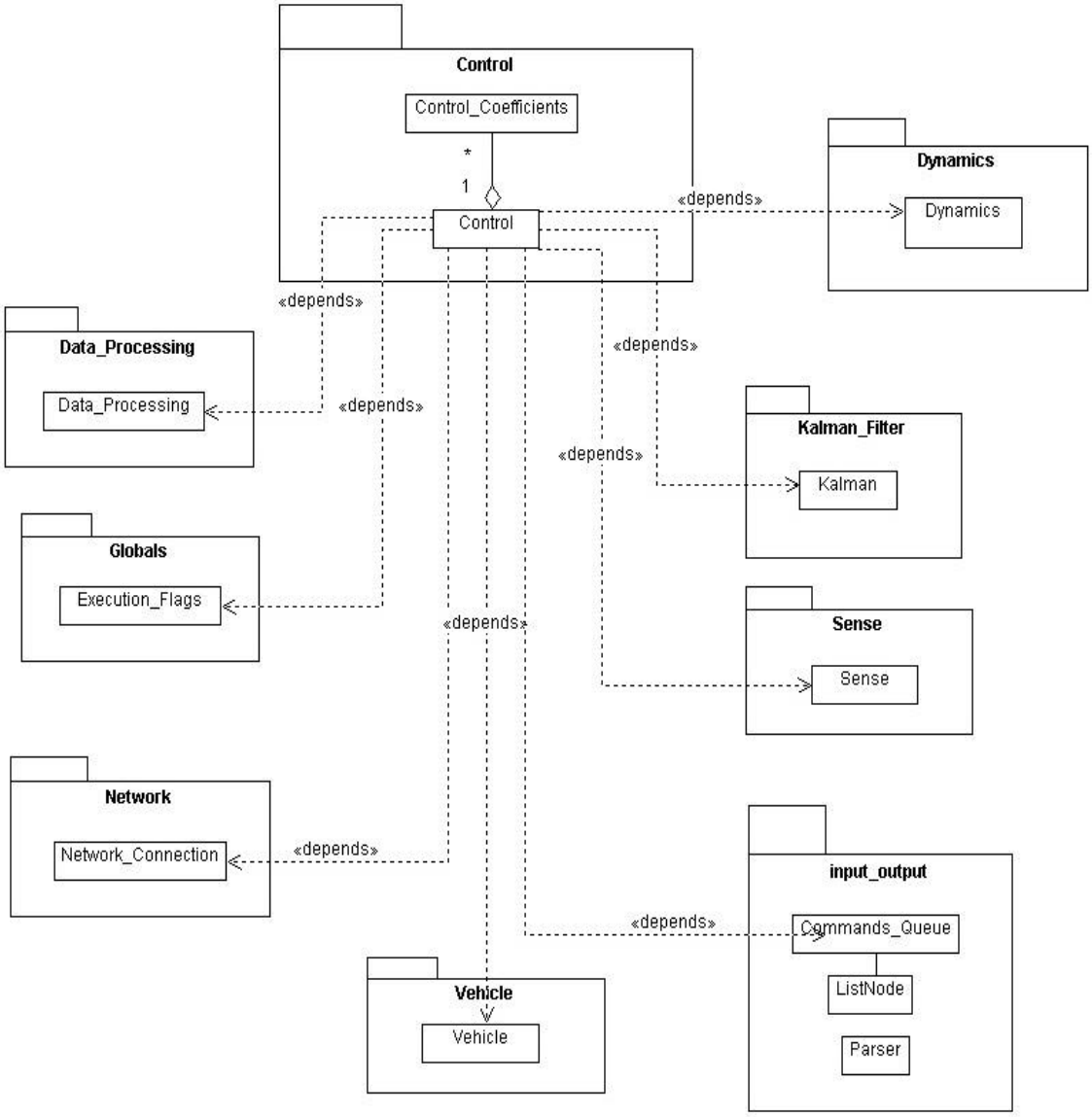

Figure 19. Control Architecture

G. SOFTWARE DEVELOPING TOOLS

1. Together®

Together®, from TogetherSoft Corporation (Figure 20) is a modeling tool to fully synchronize modeling diagrams and source code. It includes a numerous of features for to collaborative developing of software using a common language, diagrams. Together provides a single platform with a customizable user interface for all of their work

throughout the entire software development cycle. It simplifies and integrates the analysis, design, implementation, deployment, and debugging of applications. It supports Java, C++, IDL, Visual Basic 6, Visual Basic .NET, and C#, all in a single product. Some of the features provided in this developing tools are:

- Patterns: pattern repository includes J2EE patterns, UI patterns, and test cases. Developers can also customize and add to the repository.
- Refactoring: the refactoring tool makes sure that all changes are correctly propagated throughout an application.
- UML 1.3: diagrams include use case, activity, class, sequence, collaboration, state chart, component, and deployment.
- Audits and metrics: Built-in unit testing even helps uncover problems during the coding process.
- Multi-language editor: provides a editor for different languages supported, in that way the developer does not have to go outside the package.
- Documentation generation: Automatic documentation generation. Standardized documentation like Java doc is available.

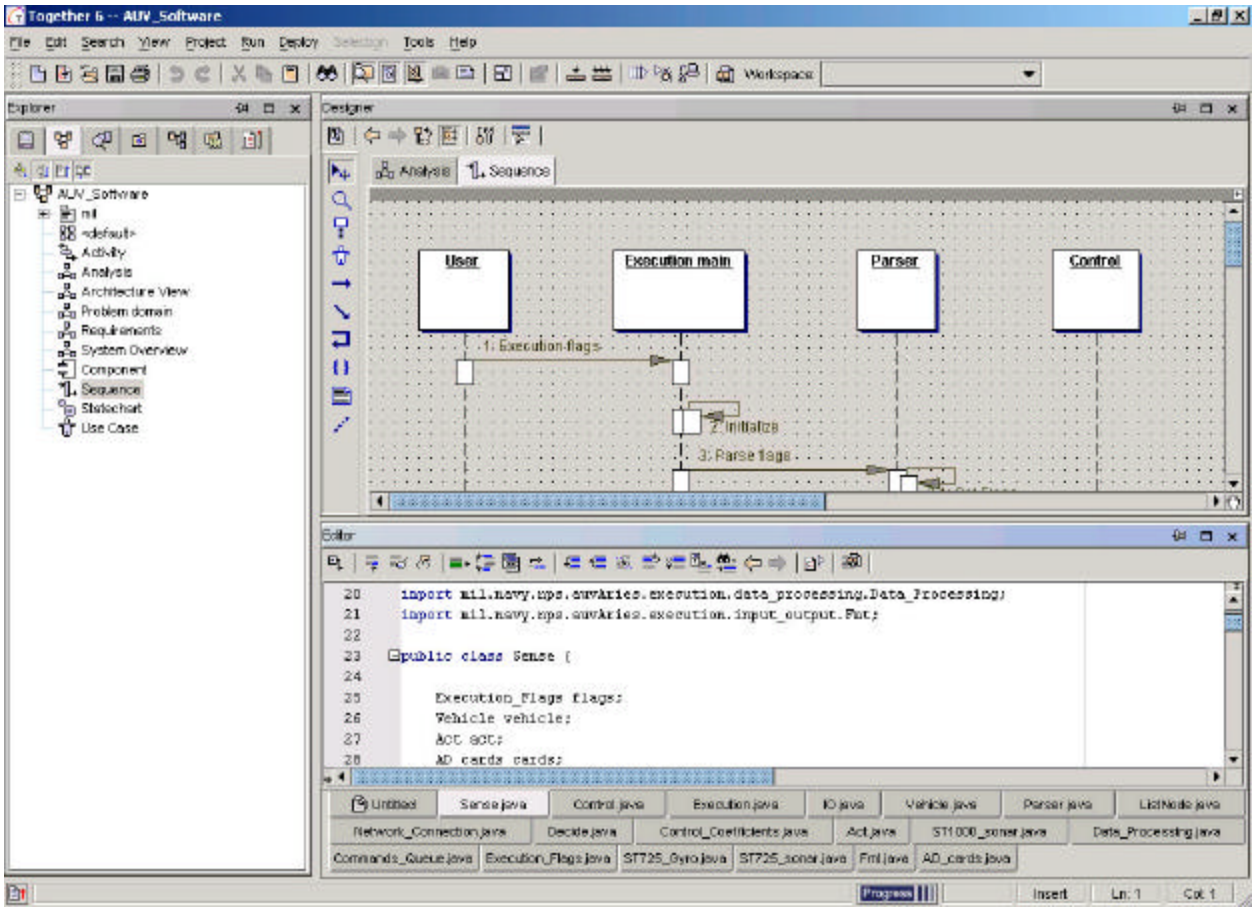

Figure 20. Together Control Center

a. UML Modeling

All UML major diagrams, including class, use case, sequence, collaboration, activity, state, and component are supported. Figure 21 shows a class diagram representing the AUV conceptual model and Figure 22 shows a Use Case model.

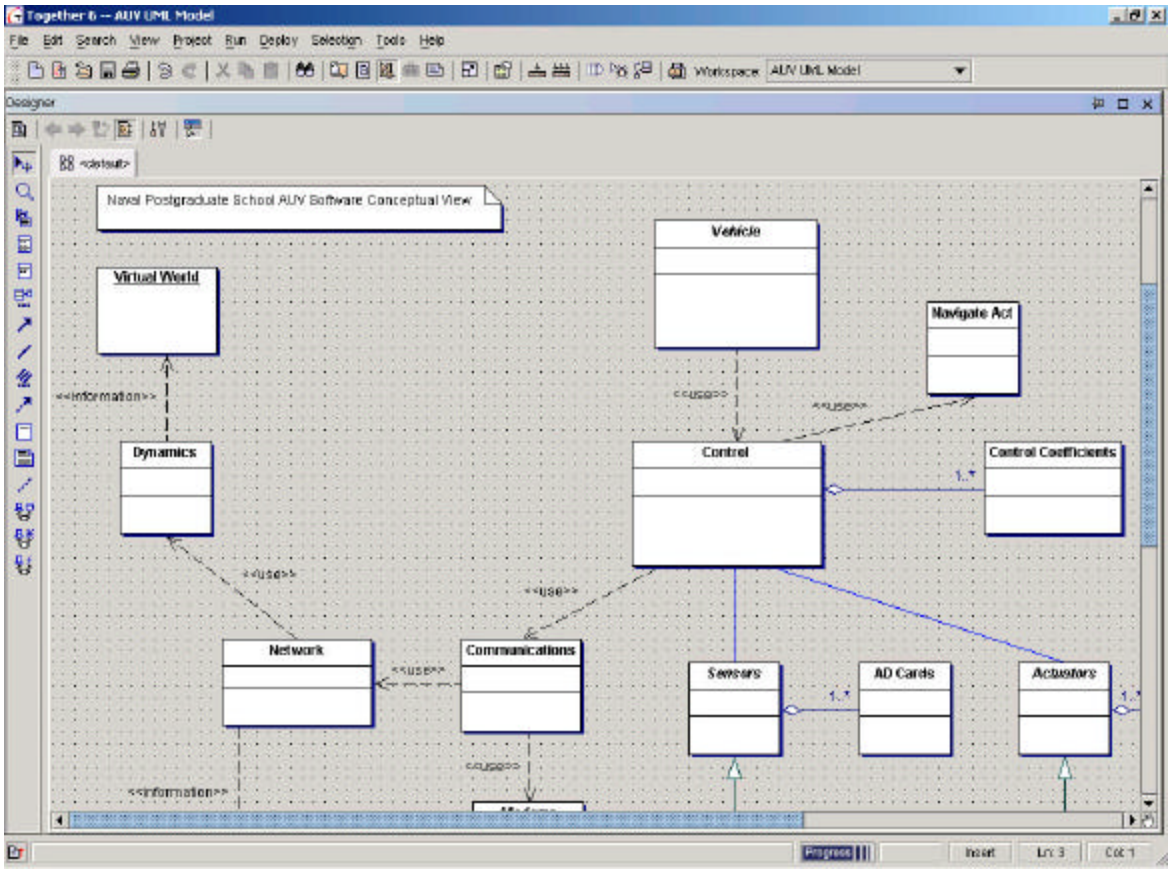

Figure 21. Class diagram as part of UML diagrams supported by Together

Advanced sequence diagram features:

- Reverse engineering of any operation into one or more sequence diagrams.
- Implementation of class operations generated from sequence diagram
- Specification of control statement, such as if, for, and while

Other modeling capabilities are:

- Visual XML modeling with XML structure diagram, with import/export capabilities, and XML editor
- State diagrams

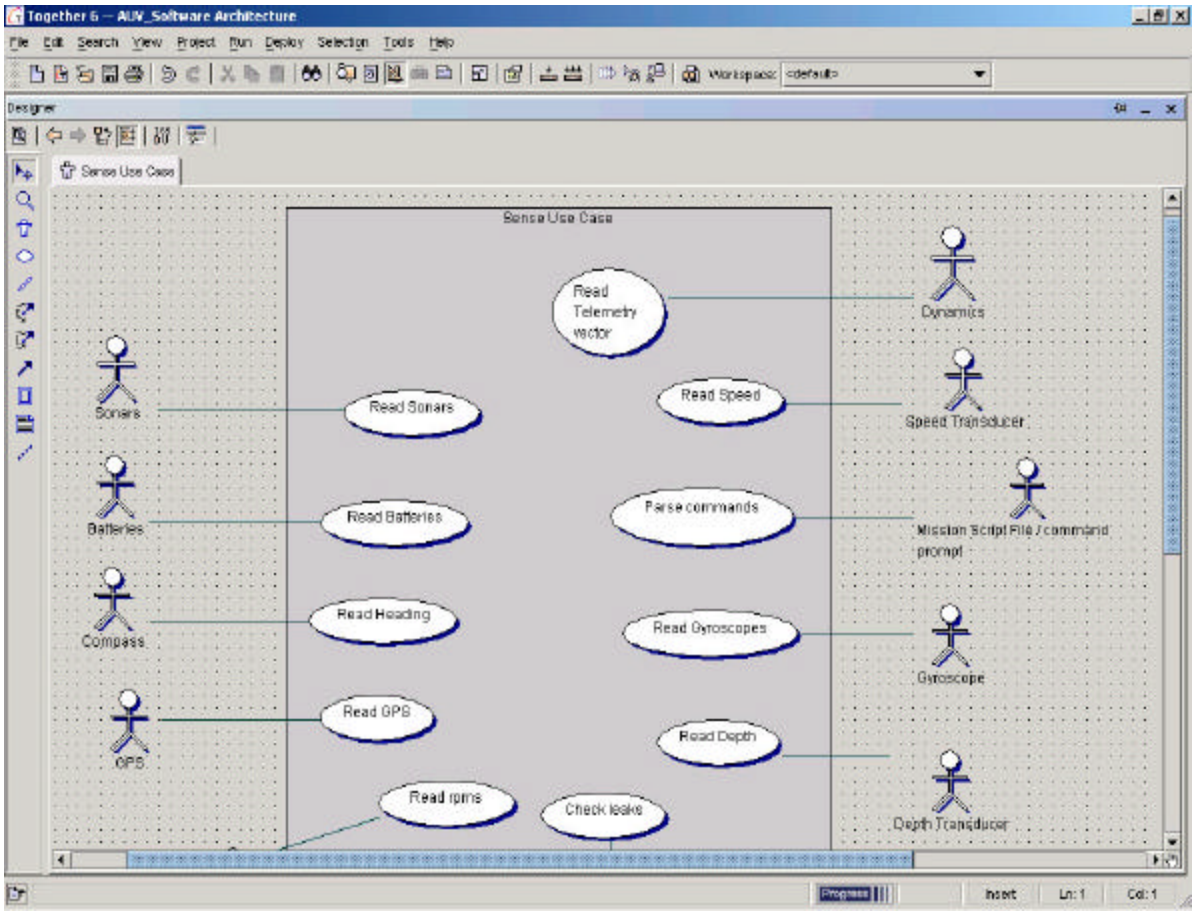

Figure 22. Class diagram as part of UML diagrams supported by Together

- Visual J2EE deployment profiling with web application, client application, and enterprise application diagrams
- Data modeling with entity relationship (ER) diagrams
- Create language-neutral class models without being tied to the particulars of a programming language
- Real–Time System Modeling:
  - ✓ Four diagrams, including system context, system architecture, event sheet, and interaction

- ✓ Extensions to standard UML diagrams, including concurrent state element on the state diagram, asynchronous messages on the sequence and collaboration diagram, and predefined activity and exception stereotypes for use case elements

b. Program Building

Together supports Java, C++, IDL, Visual Basic 6, Visual Basic.NET and C# all in a single product. It has various code assistance like common syntax error highlighting and rapid comment/uncomment blocks of code. With the feature of collapse/expand code regions, the programmer can easily navigate throughout the code and focus on in the portion he/she is working. Figure 23 shows the programmers editor pane where a full programming interface provides all the necessary tools for the developing of software. As shown in the figure the public methods from the vehicle class are shown in the window when using the dot notation. At the bottom of the editor pane the names of the opened source code are shown. The most left pane shows a typical directory structure, which represent user packages. It also shows your different UML diagrams.

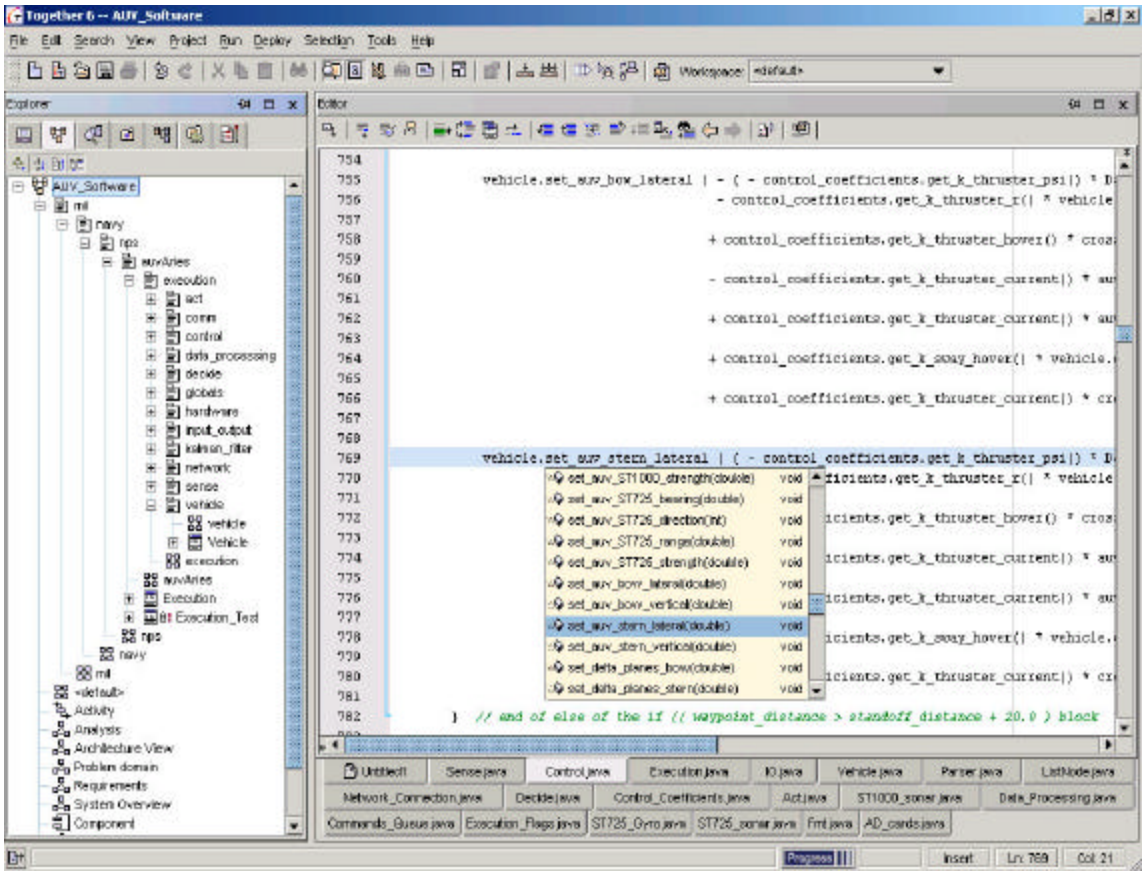

Figure 23. Together programming tools

Some of other programming tools available are:

- A full Java debugger with the ability to debug multi-threaded code, remote, distributed of servlets or any remote process, and applets among others. At run time, an output window is provided, Figure 24.
- Fully integrated version control support
- Easy code reuse and refactoring tools

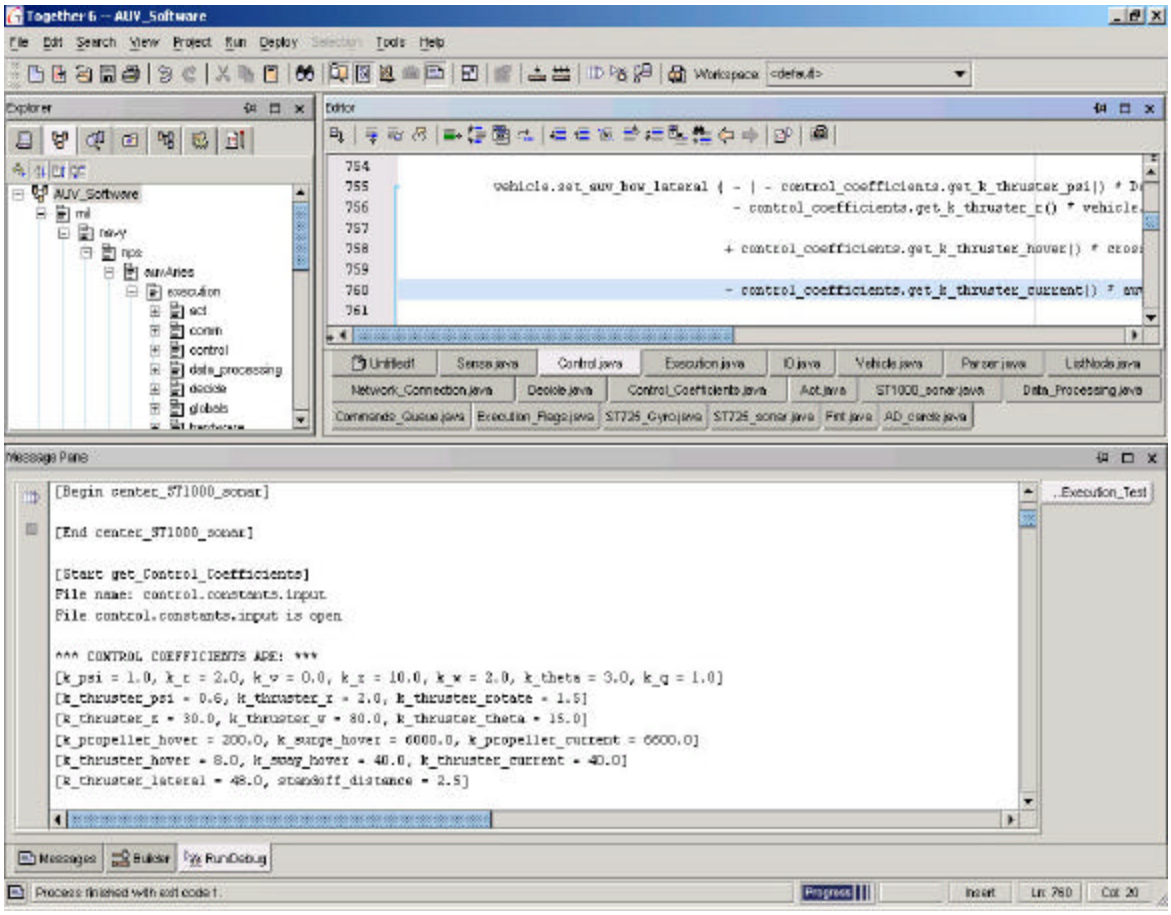

Figure 24. Together runtime output window.

c. Quality Assurance and Metrics

Extensive sets of metrics for finding and eliminating problem code is provided. Over 75 audits for Java and C++, including naming violations, performance inhibitors, and common errors are included. Metrics for Java, C++, C#, Visual Basic 6, Visual Basic.NET are provided to include cohesion, coupling, and complexity.

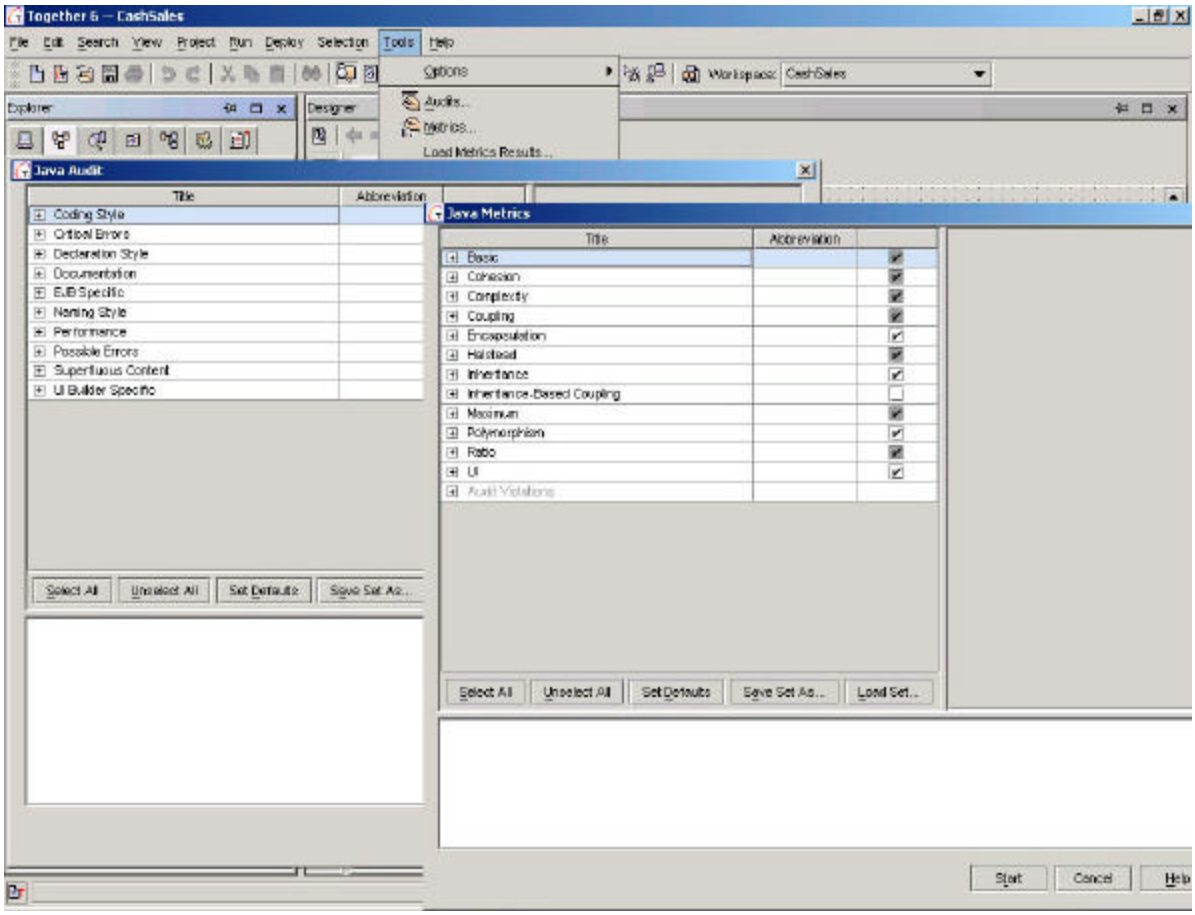

Figure 25. Together Metrics and Audit capabilities

d. Documentation Generation

A customizable, easy to use documentation generation is part of Together package. A full Javadoc style output for HTML report can be customize using templates or a new design with the options to create documentation for the entire project or just the portion you are working on. The user can generate fully customized reports based on user's templates. Printing of individual diagrams and documentation with preview option can be done as well as saving them to a file. Individual graphs diagrams can be saved to a file as images with the format svf, gif, wmg. Figures 26 and 27 shows the procedure and result of the documentation generation tool. The result is a Javadoc look alike

document. Pre-design templates as well as a design option can be used to tailored your documentation design.

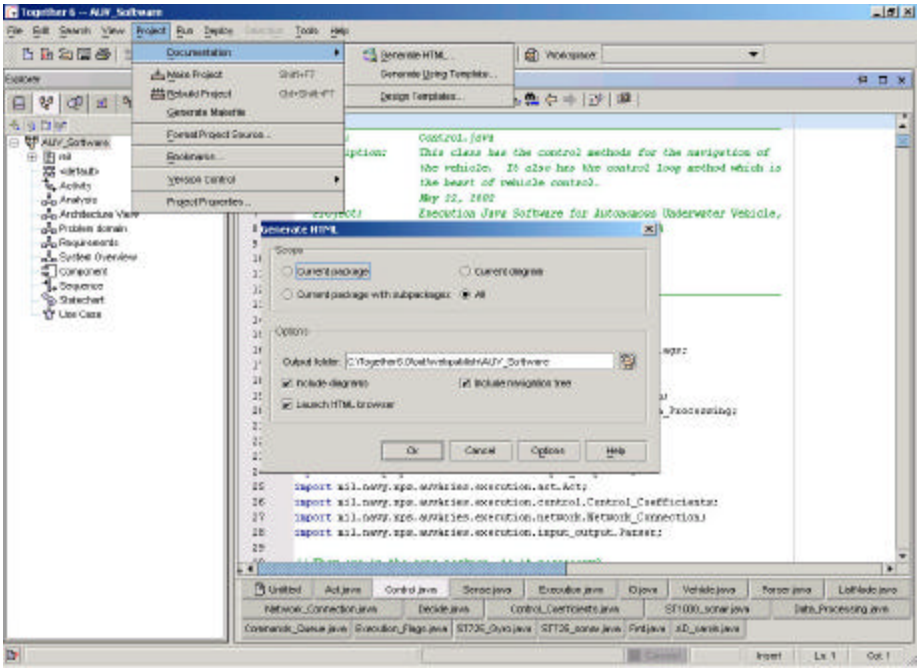

Figure 26. Together documentation generation tool

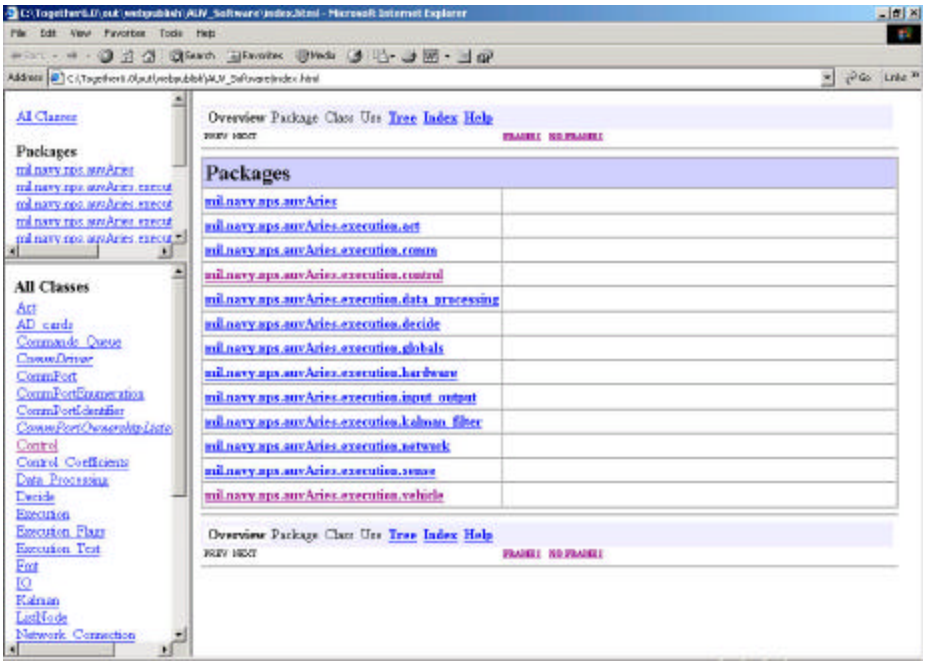

Figure 27. AUV document generated using Together

2. Sun's Forte for Java

Sun Microsystems' Forte for Java integrated development environment (IDE) supports all the editions of the Java 2 Platform: the Micro Edition (J2ME), the Standard Edition (J2SE), and the Enterprise Edition (J2EE). The Community Edition (CE) is offered at no charge and includes a set of tools, including support for CORBA and XML for developing cross-platform applications and applets written in Java. This edition includes all the functionality needed for developers to build sophisticated applications.

Forte for Java CE enable you to edit, compile, debug, browse, and deploy Java programs as well as design and create GUI applications. The IDE is built entirely from modules. The source editor, GUI editor, and debugger, and file explorer are some of the modules composing the IDE. Key benefits of Forte for Java include:

- Customizable, extensible development platform that supports a modular API
- The addition of new functionality by adding new modules from Sun, Sun's partners, and the open source community without dependencies on other elements of the IDE;
- A services-centered environment with an IDE that supports XML-based integration of Java language objects.
- 100% compatibility with Java.

H. SUMMARY

This chapter presents the general methodology of developing Java software for the NPS AUV. The software methodology is like a scientific method to follow in order

to produce quality software in potentially less time and money. The concepts were applied to the development of the Java code for the NPS AUV. The process described this chapter proceeds from analysis to software architecture design. Finally a descriptive overview of sophisticated software engineering design support environments is provided for Together and Forte.

V. REAL-TIME JAVA AND JAVA BOARD

A. INTRODUCTION

The embedded and real-time system marketplace is exploding in the "post-PC" era, especially as more and more devices are becoming Internet-enabled. Networked, real-time embedded systems are becoming common in markets such as telecommunications, industrial automation, home and building control, automotive systems, and medical instrumentation. The software content of these networked devices is soaring, putting a significant strain on scarce development resources. The real-time and embedded developer also faces an extremely diverse processing environment, with a wide range of different (and incompatible) processors, operating systems, and peripheral device types. Thus, engineers are increasingly looking to Java technologies to provide a more productive, portable development environment for real-time and embedded systems. These technologies include the Java object-oriented programming language; the Java Virtual Machine; as well as a large selection of runtime class libraries.

In the coming years, Java-enabled embedded systems will represent billions of units, ranging from smart cards to vehicle units. Java is gaining considerable popularity because it reduces software development, it is easy to learn and use and it provides flexibility and portability. The Real-time Java Specifications as November 11, 2001 addressed issues that needed to be implemented or extended in order to satisfy real-time requirements in the language. Some companies went ahead and started to develop processors capable of delivering real-time capabilities, including the embedded operating systems for the Java language. These devices are called Java boards. Some aspects of a Java board are:

a. A single chip Java micro-controllers directly execute Java Virtual Machine (JVM) byte-codes

b. Real-time Java threading primitives and a number of extended byte-codes for embedded operations.

c. The native JVM byte-code implementation eliminates the typical interpreter (JVM) or that software layers as well as the Real-Time Operating System (RTOS) kernel layer. This could provide the most optimal Java performance in both memory requirements and execution time.

d. Suited for real-time networked embedded products such as industrial controllers, smart mobile devices, and automotive communications devices.

e. Embedded real-time Operating System, reducing overhead.

In this chapter we will discuss the aspects of Real-Time Java and an overview of aJile's Java board.

B. REAL-TIME JAVA

Java is more secure than C and simpler than C++, and it has found a receptive audience in users dissatisfied with these languages. Unlike C and C++, Java has a built-in model for concurrency (threads) with low-level "building blocks" for mutual exclusion and communication that seem to offer flexibility in the design of multi-threaded programs. For example, javax.comm libraries are capable for manipulating serial and parallel devices, and V1.3 of the Java Software Development Kit (JSDK) has introduced utility classes for timed events (java.util.Date, java.util.Timer, java.tuil.TimerTask). Therefore, Java is now seems a viable candidate for real-time system.

Java is an object oriented programming language with syntax derived from C and C++ [21]. Many of applications that Java promises to enable on the Internet have associated real-time constraints. These applications include virtual reality, voice

processing, full-motion video and real-time audio for instruction and entertainment. Outside the Internet environment we find automobile, communications, and home applications using Java in embedded systems that are starting to embrace our daily lives.

In many ways, Java is a much better language than C and C++, two of the more popular languages for current implementation of embedded real-time systems. With the Java extension for real-time it will be well suited for real-time robotics, and in-vehicle navigation systems.

High-level abstractions and availability of reusable software components shorten the time to implementation and therefore the cost of development. Java's virtual machine execution model eliminates the need for complicated cross-compiler development systems, multiple platform version maintenance, and extensive rewriting and retesting each time the software is ported to a new host processor. For the NPS AUV development team, such time and money costs are major inhibitors and sometimes show-stoppers.

Unlike most existing real-time systems, many of the applications for which Real-Time Java is intended are highly dynamic. New real-time workloads arrive continually and must be integrated into the existing workload. This requires dynamic management of memory and schedulability analysis. An additional complication is that an application developer is not able to test the software in each environment in which it is expected to run. The same Java byte-code application would have to run within the same real-time constraints on a 50 MHz 486, 2 GHz 586, and on a Unix/Linux computer. Furthermore, each execution environment is likely to have a different mix of competing operating systems constraints and diverse applications with which this application code must contend for CPU time and memory resources. Finally, every Java byte-code program is

supposed to run on every Java virtual machine, even a virtual machine that is running as one of many tasks executing on a time-sharing host. Clearly, time-shared virtual machines are not able to offer the same real-time predictability as a specially designed real-time Java virtual machine embedded within a dedicated microprocessor environment. This diversity is acceptable nevertheless, since development can occur on diverse platforms while in-water operations can be highly optimized.

Java task scheduling is based on a hard-real-time model. Specially designed virtual machines offer guaranteed compliance with this model in the sense that each task is allowed to run no more than its budgeted time. Real-Time Java is an extension of traditional Java in that it offers additional syntax and additional time and memory related semantics to the Java programmer. However, it is a subset in that it forbids certain legal Java practices in cases when the use of these practices would interfere with the system's ability to support reliable compliance with real-time requirements.

Real-Time Java is implemented by a special preprocessor that converts the extended semantics into traditional Java code. This traditional Java code is then translated to Java byte codes by an ordinary Java compiler. The resulting Java byte codes can be executed on a traditional Java Virtual Machine (*JVM*) or on a specially designed Real-Time Java Virtual Machine (*RTJVM*). On a JVM, the translated Real-Time Java application will make best-effort attempts to comply with the specified real-time requirements. For best performance and real-time predictability, it would be necessary to run the translated byte codes on the RTJVM [19].

Real-Time Java consists of a combination of special class libraries, standard protocols for communicating with these libraries, and the addition of two time-related control structures to the standard Java syntax.

However, detailed study of Java reveals a number of obstacles that interfere with real-time programming. Details that might be of interest and applicable to the NPS AUV software developing team follow.

1. Challenges in the Current Java Language

The main problems for Java as a real-time technology fall into several areas, mostly related to predictability. Some of the problems categories are [6]:

- *Thread model:* The Java Language Specification explicitly states [JLS00, Section 17.12]:

  "*... threads with higher priority are generally executed in preference to threads with lower priority. Such preference is not, however, a guarantee that the highest priority thread will always be running, and thread priorities cannot be used to reliably implement mutual exclusion.*"

  This general flexibility makes it impossible to ensure that real-time threads will meet their deadlines. The implementation may or may not use priority as the criterion for choosing a thread to make ready when a lock is released. Even if it did, unbounded priority inversions could still occur since there is no requirement for the implementation to provide priority inversion avoidance policies such as priority inheritance or priority ceiling emulation. There is also no guarantee that priority is used for selecting which thread is awakened by a notify(), or which thread awakened by a notifyAll() is selected to run. Other

facets of the thread model also interfere with real-time requirements. The priority range (1 through 10) is too narrow, and the relative sleep() method is not sufficient.

- *Memory management:* Java provides no mechanism to reclaim storage but which instead are implemented with automatic memory reclamation by garbage collection. Efficient real-time garbage collection is still more a research topic than a mainstream technology. This is a particular issue for Java, since all objects (including arrays) go on the heap. The System.gc() method provides a strong hint that garbage collection is desired at a time of invocation.

- *Asynchrony:* A real-time program typically needs to respond to asynchronous events generated by either hardware or software, and sometimes needs to undergo asynchronous transfer of control (ATC), for example to time out if an operation is taking too long. The interrupt() method requires polling and thus is not an ATC mechanism. Java is rather weak in the area of asynchrony.

- *Performance:* Although "real-time" does not mean "real fast", run-time performance cannot be ignored. Java has several challenges in this area. The key to "write once, run anywhere" is the JVM and the binary portability of class files. Nevertheless any software interpreter introduces overhead, and hardware implementations are not yet mainstream technology. Garbage Collection has an obvious performance impact, but performance (or performance interval) of garbage collection is not guaranteed. This problem

often manifest itself as intermittent "pauses" or "hiccups" when garbage collection noticeable preempts other processes at inappropriate intervals.

a. Concurrency and Synchronization

i. Scheduling and Priorities

The Core Java specification supports a large range of priorities. Each implementation is required to support a minimum of 128 distinct values, with the highest N being used as interrupt priorities, where N is implementation-defined. In addition, the Core Java semantics require preemptive priority-based scheduling as defined by the *FIFO_Within_Priorities* policy. This model is in marked contrast to Baseline Java's small priority range (10 values) and absence of guarantee that a higher priority task will preempt a low priority task when it is ready to run. Alternative scheduling policies may be specified via profiles. The Core task class hierarchy is rooted at CoreTask. A *CoreTask* object must be explicitly started via the start() method. There are two specialized extensions of *CoreTask*:

- The SporadicTask class defines tasks that are readied by the occurrence of an *event* that is triggered either periodically or via an explicit call to its fire() method.
- The InterruptTask class defines tasks that are readied by the occurrence of an *interrupt* event, making them analogous to interrupt service routines.

ii. Task Synchronization Primitives

Task synchronization is provided in the Core Java specification via a number of different features, a first group of which supports priority inversion avoidance and a second group of which does not. In the first group, Baseline Java-style usage of ***synchronized*** methods and ***synchronized(this)*** constructs are both supported and define transitive priority inheritance to limit the effects of priority inversion.

b. Memory Management

i. Garbage Collection

A key requirement of the Core Java specification is that the system need not incur the overhead of traditional automatic garbage collection. This is intended to provide the necessary performance and predictability, avoiding overheads such as read/write barriers, object relocation due to compaction, stack/object scanning and object description tables, as well as avoiding the determinism problems associated with executing the garbage collector thread.

c. Asynchrony

i. Asynchronous Events

Real-time systems typically interact closely with the real-world. With respect to the execution of logic, the real-world is asynchronous. Three kinds of asynchronous event are defined by the Core Java specification:

- *PeriodicEvent* is defined to support periodic tasks. The event fires at the start of each period, which causes the associated periodic event handler task to become ready to execute its work() method.
- *SporadicEvent* is defined to support sporadic tasks that are triggered by software. The event is explicitly fired by a task which causes the associated sporadic event handler task to become ready to execute its work() method
- *InterruptEvent* is defined to support interrupt handling. The event can be explicitly fired by a task (to achieve a software interrupt) or implicitly fired by a hardware interrupt. This causes the associated interrupt event handler task to become ready to execute its work() method, which must implement the *Atomic* interface.

d. Time

The Core Java specification defines a Time class that includes methods to construct times in all granularities from nanoseconds through to days. These can be used to program periodic timer events to trigger cyclic tasks or to timeout overrunning task execution. In addition, the relative delay sleep() method and the

absolute delay sleepUntil() method provide a programmatic means of coding periodic activity. In both cases, the time quantum can be specified to the nanosecond level. There is also a method tickDuration() to return the length of a clock tick.

1. Real-Time Specification for Java (RTSJ)

The Real-Time Specifications for Java extends the core Java semantics in the following eight areas:

- Scheduling
- Memory Management
- Synchronization
- Asynchronous event handling
- Asynchronous Transfer of Control
- Asynchronous thread termination
- Physical Memory Access
- Exceptions

The Real-Time Java Expert Group Specifications follow the following principles:

- *Applicability to particular Java environments.* Usage is not to be restricted to particular versions of the Java Software Development Kit.
- *Backward compatibility*. Existing Java code can run on any implementation of the RTSJ.
- *"Write Once, Run Anywhere"*. This is an important goal but difficult to achieve for real-time systems (as a trivial example of the difficulties, the

correctness of a real-time program depends on the timing properties of the executing code, but different hardware platforms have different performance characteristics).

- *Predictable execution.* This is the highest priority goal; performance or throughput may need to be compromised in order to achieve it.

In summary, the requirements design provides real-time functionality in several areas. Below is some that could be applicable to the NPS AUV:

- *Thread scheduling and dispatching.* The RTSJ introduces the concept of a *real-time thread* and defines both a traditional priority-based dispatching mechanism and an extensible framework for implementation-defined (and also user-defined) scheduling policies.

- *Memory management.* The RTSJ provides a general concept of a *memory area* that may be used either explicitly or implicitly for object allocations. Examples of memory areas are the (garbage-collected) heap, and also "immortal" memory whose objects persist for the duration of an application's execution. Another important special case is a memory area that is used for object allocations during the execution of a dynamically determined "scope", and which is automatically emptied at the end of the scope. The RTSJ defines the concept of a "no-heap real-time thread" which is not allowed to reference the heap; this restriction means that such a thread can safely preempt the Garbage Collector.

- *Synchronization and resource sharing.* The RTSJ requires the implementation to supply one or more mechanisms to avoid unbounded priority inversion, and it defines two monitor control policies to meet this requirement: priority inheritance and priority ceiling emulation. The specification also defines several "wait free queues" to allow a no-heap real-time thread and a Baseline Java thread to safely synchronize on shared objects.

- *Asynchrony.* The RTSJ defines a general event model based on the framework found in the AWT and Java Beans. An event can be generated from software or from an interrupt handler. Event handlers behave like threads and are schedulable entities. The design is intended to be scalable to very large numbers of events and event handlers (tens of thousands), although only a small number of handlers are expected to be active simultaneously. The RTSJ also defines a mechanism for asynchronous transfer of control (ATC), supporting common idioms such as timeout and mode change. The affected code needs to explicitly permit ATC; thus code that is not written to be asynchronously interruptible will work correctly.

Specifications made by the RTSJ group applicable to the NPS AUV software developing team are introduced below.

a. Concurrency and Synchronization

i. Schedulable Objects

The basis of this approach to concurrency is the class RealtimeThread, a subclass of Thread. One of the concerns of real-time programming is to ensure the timely or predictable execution of sequences of machine instructions. Various scheduling schemes name these sequences of instructions differently. Typically

used names include threads, tasks, modules, and blocks. The RTSJ introduces the concept of a schedulable object.

Any instance of any class implementing the interface Schedulable is a schedulable object and its scheduling and dispatching will be managed by the instance of Scheduler to which it holds a reference. The RTSJ requires three classes that are: schedulable objects; RealtimeThread, NoHeapRealtimeThread, and AsyncEventHandler.

The programmer can determine by analysis of the program, testing the program on particular implementations, or both whether particular threads will always complete execution before a given timeliness constraint. This is the essence of real-time programming: the addition of temporal constraints to the correctness conditions for computation. Basically a scheduler:

- Have fixed priority preemptive, FIFO within priority
- Does not allow implicit modification of thread priority except for priority inversion
- Supports a minimum of 28 distinct priority levels, above the 10 Baseline Java levels
- Provide isFeasible() method for feasible analysis
- Assume priorities are assigned based on Rate Monotonic Analysis (RMA)
- Can be replaced by other scheduling policies at runtime

The set of classes:

- Allow the definition of schedulable objects.
- Manage the assignment of execution eligibility to schedulable objects.
- Perform feasibility analysis for sets of schedulable objects.
- Control the admission of new schedulable objects.
- Manage the execution of instances of the AsyncEventHandler and Realtime-Thread classes.

- Assign release characteristics to schedulable objects.
- Assign execution eligibility values to schedulable objects.
- Define temporal containers used to enforce correct temporal behavior of multiple schedulable objects.

b. Scheduling and Priorities

Meeting hard deadlines is one of the most fundamental requirements of a real-time operating system and is especially important in safety-critical systems. Depending on the system and the thread, missing a deadline can be a critical fault. Rate monotonic analysis (RMA) is frequently used by system designers to analyze and predict the timing behavior of systems [9]. In doing so, the system designer is relying on the underlying operating system to provide fast and temporally deterministic system services. Not only must the designer understand how long it takes to execute the thread's code, but also any overhead associated with the thread must be determined. Overhead typically includes context switch time, the time required to execute kernel system calls, and the overhead of interrupts and interrupt handlers firing and executing.

The RTSJ requires a base scheduler that is fixed-priority preemptive with at least 28 distinct priority levels, above the 10 Core Java levels. An implementation must map the 28 real-time priorities to distinct values, but the 10 non-real-time levels are not necessarily distinct. Constructors for the RealtimeThread class allow the programmer to supply scheduling parameters (SchedulingParameters class), release parameters (ReleaseParameters class), memory parameters (MemoryParameters class), a memory area (MemoryArea class), and processing group parameters (ProcessingGroupParameterx class). The scheduling parameters characterize the thread’s execution eligibility (for example, its priority). A real-time thread can have a priority in either the real-time range

or the Baseline Java range. The release parameters identify the real-time thread's execution requirements and properties (whether it is periodic, aperiodic or sporadic).

c. Synchronization

Core Java uses monitors to perform synchronization. An unbounded priority inversion in a thread synchronizing on a locked object can lead to missed deadlines, and the RTSJ accordingly requires that the implementation supply one or more monitor control policies to avoid this problem. By default the policy is priority inheritance, but the RTSJ also defines a priority ceiling emulation policy. Each policy can be selected either globally or per-object and the choice can be modified at run time. An implementation can supply a specialized form of priority ceiling emulation that prohibits a thread from blocking while holding a lock; this avoids the need for mutual exclusions.

d. Memory Management

Memory management is a particularly important feature of the Java programming environment. The RTSJ defines a memory allocation and reclamation specification that would:

- be independent of any particular GC algorithm,
- allow the program to precisely characterize a implemented GC algorithm's effect on the execution time, preemption, and dispatching of real-time Java threads,
- allow the allocation and reclamation of objects outside of any interference by any GC algorithm.

The RTSJ introduces the concept of a memory area. A memory area represents an area of memory that may be used for the allocation of objects. Some

memory areas exist outside of the heap and place restrictions on what the system and garbage collector may do with objects allocated within. Objects in some memory areas are never garbage collected; however, the garbage collector must be capable of scanning these memory areas for references to any object within the heap to preserve the integrity of the heap. There are four basic types of memory areas:

1. Scoped memory: provides a mechanism for dealing with a class of objects that have a lifetime defined by syntactic scope (the lifetime of objects on the heap).

2. Immortal memory: represents an area of memory containing objects that, once allocated, exist until the end of the application, i.e., the objects are immortal.

3. Physical memory: allows objects to be created within specific physical memory regions that have particular important characteristics, such as memory that has substantially faster access.

4. Heap memory: the Baseline Java heap memory. Heap memory represents an area of memory that is the heap. The RTSJ does not change the determinant of lifetime of objects on the heap. The lifetime is still determined by visibility.

Garbage-collected memory heaps have always been considered an obstacle to real-time programming due to the unpredictable latencies introduced by the garbage collector. The RTSJ addresses this issue by providing several extensions to the memory model, which support memory management in a manner that does not interfere

with the ability of real-time code to provide deterministic behavior. This goal is accomplished by allowing the allocation of objects outside of the garbage-collected heap for both short-lived and long-lived objects.

e. Asynchrony

The RTSJ supplies two mechanisms relevant to asynchronous communication: asynchronous event handling, and asynchronous transfer of control.

i. Asynchronous Event Handling

The asynchronous event facility comprises two classes: AsyncEvent and AsyncEventHandler. An AsyncEvent object represents something that can happen, like a hardware interrupt, or a computed event like an airplane entering a specified region. When one of these events occurs, which is indicated by the fire() method being called, the associated handleAsyncEvent() methods of instances of AsyncEventHandler are scheduled and thus perform the required logic. An instance of AsyncEvent manages two things:

- the unblocking of handlers when the event is fired, and
- the set of handlers associated with the event.

This set can be queried, have handlers added, or have handlers removed. An instance of AsyncEventHandler can be thought of as something roughly similar to a thread. It is a Runnable object: when the event fires, the handleAsyncEvent() methods of the associated handlers are scheduled. What distinguishes an AsyncEventHandler from a simple Runnable is that an AsyncEventHandler has associated instances of ReleaseParameters, SchedulingParameters and MemoryParameters that control the actual execution of the handler once the associated AsyncEvent is fired. When an event is fired, the handlers are executed asynchronously, scheduled according to the associated ReleaseParameters and SchedulingParameters objects, in a manner that looks like the handler has just been assigned to its own thread. It is intended that the system can cope well with situations where there are large numbers of instances of AsyncEvent and AsyncEventHandler (tens of thousands). The number of

fired (in process) handlers is expected to be smaller. A specialized form of an AsyncEvent is the Timer class, which represents an event whose occurrence is driven by time. There are two forms of Timers: the OneShotTimer and the PeriodicTimer. Instances of OneShotTimer fire once, at the specified time. Periodic timers fire off at the specified time, and then periodically according to a specified interval. Timers are driven by Clock objects. There is a special Clock object, Clock.getRealtimeClock() that represents the real-time clock. The Clock class may be extended to represent other clocks the underlying system might make available (such as a soft clock of some granularity).

f. Time and Timers

The RTSJ provides several ways to specify high-resolution (nanosecond accuracy) time:

- absolute time
- relative number of milliseconds and nanoseconds,
- rational time (a frequency, i.e. a number of occurrences of an event per relative time).

In a relative time 64 bits (a long) are used for the nanoseconds, and 32 bits (an int) for the milliseconds. The rational time class is designed to simplify application logic where a periodic thread needs to run at a given frequency. The implementation, and not the programmer, needs to account for round-off error in computing the interval between release points. The time classes provide relevant constructors, arithmetic and comparison methods, and utility operations. These classes are used in constructors for the various release parameters classes. The RTSJ defines a default real-time clock which can be queried (for example to obtain the current time) and which is the basis for two kinds of timers: a one-shot timer, and a periodic timer. Timer objects are instances of asynchronous events; the programmer can register an asynchronous event handler with a

timer to obtain the desired behavior when the event is fired. A handler for a periodic timer is similar to a real-time thread with periodic release parameters but is likely to be more efficient.

C. JAVA BOARD

While the portability of Java byte-code holds great appeal, neither interpretation nor Just-In-Time (JIT) compilation is adequate for real-time embedded use. Different companies started to look into the development of a hardware device capable of running byte-code directly and operating with a real-time operating system. A direct execution Java micro-controllers, could accomplish such task. A Java board is a board with a microprocessor onboard that contains all the instruction set and all the Java Virtual Machine bytecodes; no Java interpreter or JIT compiler is required. The space-efficient instruction set can more than halve the amount of code space required for a typical application, and has been designed for predictable Java bytecode execution.

1. aJile aJ-100EVB

The aJile Systems aJ-100EVB Java single-board computer makes real-time embedded Java a promising solution for embedded applications like the NPS AUV. Based on the aJ-100 direct-execution Java microprocessor, the aJ-100EVB provides a platform for the development of real-time embedded applications entirely in Java.

The aJ-100 instruction set architecture was designed with an embedded systems focus, and thus maximizes the amount of runtime data that can be embedded. The aJ-100 directly supports the Java thread model in hardware. aJ-100 also defines a set of extended instructions for physical hardware interfacing and other systems programming tasks.

The aJile Systems aJ-100 is a second-generation direct execution Java microprocessor. It is a compact, low-power board that is suited as a micro controller core in application areas such as automotive or robotics operations. aJ-100 supports 32, 16, or

8 bit external data buses, and provides a test interface via a connection through a parallel port. With aJ-100, real-time embedded developers can use the Java language, with its proven productivity advantages, to produce applications that are as space and time efficient as those written in languages such as C for other micro controller platforms.

a. aJ-100EVB Features

One of the unique features of the aJ-100 is its hardware support for real-time Java threads. aJ-100 implements the basic synchronization and thread scheduling routines in microcode. There are several benefits from this approach. For example, it requires no Real-Time Operating System (RTOS) kernel, thus saving memory and creating a small footprint and low power consumption, a desire characteristic in autonomous vehicles. The aJ-100 hardware supports periodic thread dispatching, and also implements priority inversion control.

The aJ-100EVB real-time Java single-board computer, bundled with Sun's Java 2 Micro Edition (J2ME) Connected Limited Device Configuration (CLDC) Java-based runtime system, optimizing application builder, and debugging tools provides a complete solution for developing real-time and mobile networked embedded Java applications. Using commercial Java IDEs, application developers can create standalone Java applications totally in Java with the performance and memory efficiency of systems programmed in C and assembly.

The aJ-100EVB is bundled with software to build and debug optimized real-time embedded Java applications. Features of the aJ-100EVB include:

- 32 bit Real-time low-power direct execution Java processor
- JVM bytecodes are native instructions
- Single and double precision floating-point arithmetic

- Native Java threading support
- Hard real-time, multithreading kernel in hardware
- Thread to thread yield time of less than <1 msec.
- Built-in deterministic scheduling queues
- Ethernet (10 Base-T)
- 1 MB SRAM
- 4 MB Flash
- 2 Serial Ports
- LCD controller
- 4-wire resistive touch screen controller
- SPI Port with 3 (4 if touch screen not used) available select lines
- 13 General-Purpose I/Os
- Java Runtime software, including networking
- Optimizing Linker/Application Builder
- PC-based debugger; interfaces to aJ-100EVB through the PC parallel port

Figure 28 shows aJile's aJ100EVB Java board. Notice the Ethernet, serial and parallel port connections.

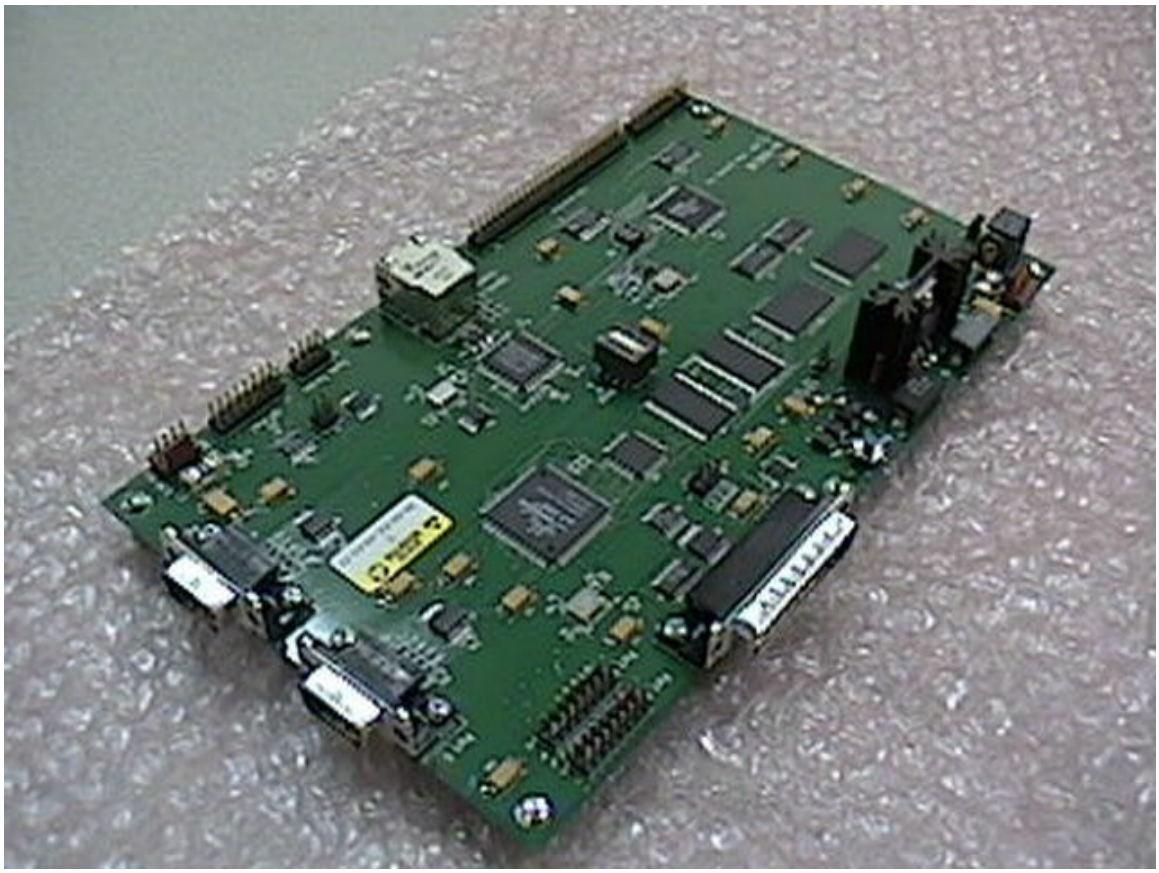

Figure 28. aJile's aJ100EVB board

b. Programming in the aJ100EVB

Programming for the Java board follows the conventional real-time embedded applications but using Java as the programming language. The traditional compile/link/load development cycle is the one used on this board. For such development cycle software tools are provided for the compilation and linking of the program into the Java board. You need to have installed the Java Development Kit (JDK) 1.3.1 into the directory C:\jdk1.3.1 before starting the process of linking.

i. Linking the Java code

Once the code is compiled using traditional compilation like javac in the command prompt or through a developing software, the classes must be linked for embedded execution. aJile Systems provides a linker/loader tool calls JemBuilder. JemBuilder provides the user with a quick and easy way to link the code.

Basically the steps are:

1. Compile your classes
2. Launch JemBuilder and create a new Project
    a. Create a JVM
    b. Name the Main class
    c. Enter the classpath used during compilation phase, not the JDK classpath
    d. Add any drivers to the JVM of any hardware resources that would be managed by the created JVM
    e. Select the runtime classes like Runtime_CLDC (Java Micro Edition)
3. Save the project
4. Build the project.

A final screen will look like Figure 29.

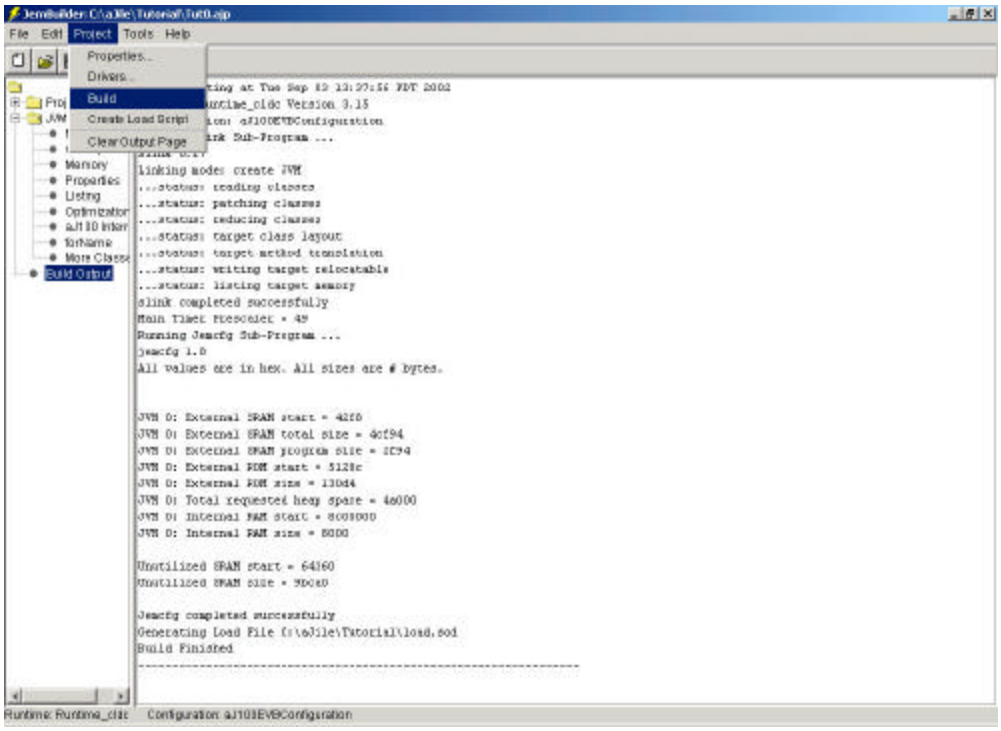

Figure 29. JemBuilder graphical interface at the building phase

ii. Loading and Executing

After the build is finished, the load and execution of the code is done by another software tool provided by aJile Systems called Charade, Figure 30. Basically after launching Charade:

1. In the main menu under File select Execute
2. Select the load.sod created by JemBuilder. It will be located in the classpath provided to JemBuilder in the building phase.
3. Run the program by clicking Go

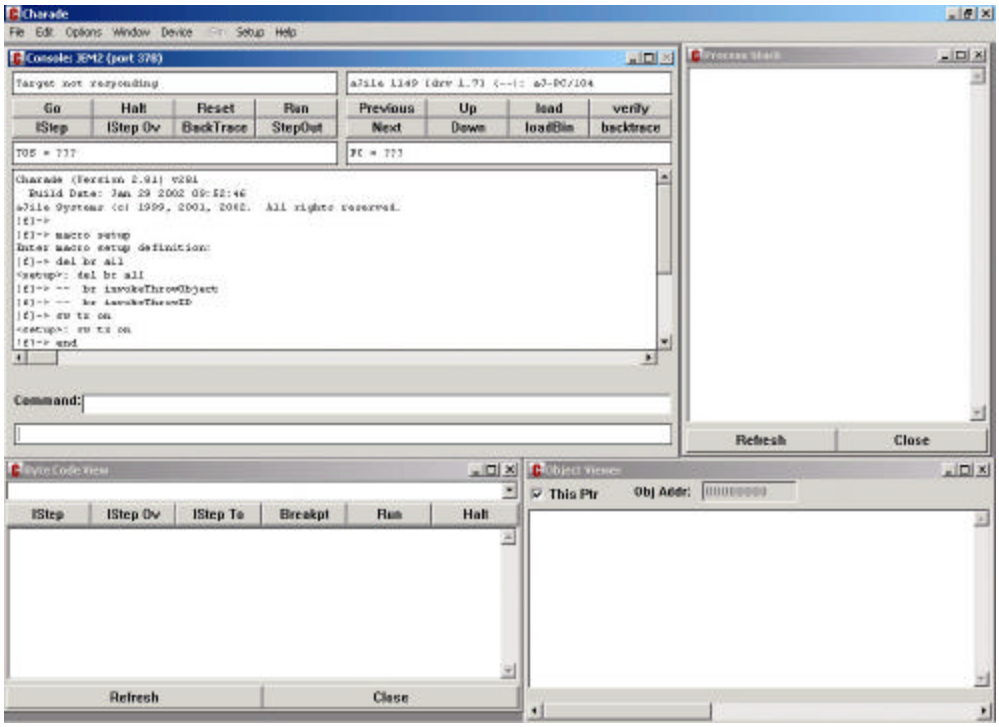

Figure 30. Charade loading and executing software tool

2. Imsys Cjip

The Cjip from Imsys AB located at Upplands Väsby, Sweden, is a processor for embedded applications with low power consumption, featuring native Java byte code execution as well as Assembler/C/C++ support. The Cjip COM Evaluation Kit is designed to be used for evaluation and developing applications for the Cjip, using Java or C. It contains several different communication peripherals, in addition to the Cjip processor and memory. As the aJ-100EVB, a parallel port cable connects it to the Imsys Developer development system for program load and software debugging. The Imsys Developer is an IDE for the Cjip processor family, in which you can perform all programming work. A source code editor, compilers of different types, assembler, linker, debugger and other tools in a visual screen environment enables the developer to work from a single package. Some of the features of the evaluation kit are:

- Ethernet port for 10/100 Base-T
- User port IO connector with 10 user signals
- Ports A, B and C on the Cjip are directly available in connectors
- Parallel port for software debugging
- Embedded Real-Time Operating System

The Cjip executes Java byte codes directly, rather than requiring a software Java Virtual Machine (JVM). The Cjip is fully compliant with Sun's J2ME specification with support for CLDC.

D. SUMMARY

Real-Time Java is getting to be a reality. The specifications are out and javax.realtime package has been in test for several months now. With the numerous advantages that Java offers, real-time software development will have another true object oriented language with wide support. The use of Java as a language of choice has gone a little bit further. With companies like aJile Systems in San Jose, California, the development of Java boards which provides a microprocessor capable of running byte-code directly and operating with a real-time operating system is a reality. This combination would provide a fast, reliable and economical way to create embedded software for multiple applications using the Java language for real-time applications.

In this chapter an overview of Real-Time Java was presented. A description of some specifications that could be of interest for the NPS AUV software developing team was discussed. Viable candidate for Java hardware were presented. The aJile aJ-100EVB was discussed with more detailed as a selection for the hardware incorporation of Java board.

VI. EXECUTION JAVA SOURCE CODE DESCRIPTION

A. INTRODUCTION

The Java execution level code of the NPS AUV follows the same algorithms used in the previous C code for computation of control parameters, flags and command parsing and matching, and acts command. An in-depth description of these algorithms and procedure are discussed by Michael L. Burns in his thesis [Burns 96].

In this chapter a discussion of the flow and execution of the Java execution level code as well as the dynamics of the object-oriented approach of the different classes that composed the entire software system. An important difference in the new code is the elimination of global variables for the telemetry of the vehicle and control variables. Those variables are encapsulated in the respective classes an appropriate classification types have been assigned.

B. EXECUTION JAVA CODE PACKAGES AND CLASSES

The execution Java code is composed of 10 packages and 20 classes respectively:

- Sense: All the reading of sensors
- Decide: Matching flags and commands
- Act: Actuators commands
- Vehicle: Telemetry holding
- Control: Control calculation and control coefficients
- Data_Processing: Data processing and Kalman filter
- Globals: Execution flags
- Hardware: Any hardware-specific code
- Input_Output: All write/read methods for files and user input
- Network: Network communication between execution and dynamics as well as tactical level

The main class is the execution.java under the mil.navy.nps.auvAries.execution package. The Javadoc in Appendix B gives a good documentation of the composition of the execution Java code, its classes and the packages where they belong.

1. Sense

The Sense class contains all the read methods like `read_depth()` and `read_heading()`. The methods are closely related to physical devices onboard the AUV. Since the target of this thesis was first the virtual world, most of the code in the Sense class has been commented and left as C code. The applicable virtual world code has been developed in Java and implemented.

2. Decide

a. Decide

The Decide class has two major methods, `match_command_line_flags()` and `match_commands()`. The purpose of the method `match_command_line_flags()` method is to take the parameters from the command prompt, parse them, match them against a pre-defined set of flags and update the respective flags accordingly. The method `match_commands()` takes two parameters, a command and the respective parameters associated with that command, example position 10 8 5. Based on that command passed, it will go through to try to match it with a set of pre-defined set of commands. Based on the match a corresponding action takes place. For example, if the command was position 10 8 5, the command position matches and the action of setting the vehicles x, y, and z position will take place in the form of `vehicle.set_x( Double.parseDouble(parameters[1] ))` respectively.

b. Parser

This class is responsible for parsing mission commands from the mission script file and to parse the telemetry string and update the telemetry based on that parsing.

3. Act

The Act class is responsible to act upon the decisions made on the decide class. This class has methods like command_rudders(double angle), which will instruct the rudders to a specific angle. Several methods deal with code that addresses the real world. Such methods are implemented as skeleton, the source code for the real world has been omitted. Several of those classes provide methods to check the flag `LOCATIONLAB` and to take the appropriate action.

4. Control

This package has two classes, control and control_coefficients.

a. Control

The Control algorithms are heart of all calculations and vehicle navigation. Control has all the methods that compute control parameters for the navigation of the vehicle. It is by far the most largest and complex class. It has all the control computational methods and the control loop method, which is like an cyclic executive. The control loop methods must operate in a cyclic loop of less than 10 Hz in order to have a good control feedback loop of the vehicle.

b. Control_Coefficients

This class has all the control coefficients required in order to control the vehicle. The constructor assigns a pre-defined set of coefficients calculated for the ARIES AUV. The coefficients in the class could be changed or a file could be created in order to accommodate another type of AUV. The name of the file is

control.constants.insput and it must follow a predefined format. Figure 31 shows the required format for the file.

5. Input_Output

The input/output operations like writing and reading files are in this package.

a. IO

This class is responsible for opening the mission script file, read the commands and save them in a queue. It also has a method to build the telemetry string from the variables from the vehicle object. The control coefficients are read through a method from this class. The telemetry is also saved to a file using a method from this class.

b. Commands_Queue and List_Node

The combination of these two classes will maintain the commands in a queue and will pop, push, or print the commands from the queue. The List_Node class is queue implementation using a link list.

6. Vehicle

The vehicle class holds all the telemetry information.

7. Network

a. Network_Connection

The network_connection class is responsible to establish communication between the virtual world (dynamics), tactical and the execution Java code. During this communication, the execution Java code reads telemetry information from dynamics as well as send telemetry information to dynamics.

8. Globals

a. Execution_Flags

All the execution flags are hold in this class. The constructor initialized all the flags to a pre-determine values. Through out the run time of the program this flags are updated depending the state and conditions of the AUV.

9. Data_Processing

a. Kalman

Kalman filter processes are performed using the methods contain in this class.

10. Execution

This constitutes the main program. The integration of execution uses all the classes by creating all the required objects.

C. EXECUTION JAVA CODE INTEGRATION

The execution Java code starts with the import of all the dependent packages and instantiation of all the classes that will be used. Some of the class instantiations are done by passing already instantiated classes objects as parameters since these objects will be used on those classes. Basically the same procedure or flow is followed from the previous C code explained by Burns [Burns 96]. Since Java is an object-oriented

```
AUV execution level control algorithm coefficients     28 March 97

k_psi   k_r    k_v    k_z    k_w    k_theta     k_q
1.00    2.00   0.00  10.00   2.00    3.00       1.00

k_thruster_psi      k_thruster_r    k_thruster_rotate
     0.60                2.00               1.5

k_thruster_z        k_thruster_w    k_thruster_theta
   30.00                80.00              15.0

k_propeller_hover   k_surge_hover   k_propeller_current
     200.00           6000.00            6600.00

k_thruster_hover    k_sway_hover    k_thruster_current
     8.00                40.00              40.00

k_thruster_lateral  standoff_distance
   48.00                 2.50

*****                                                          *****
*****          Do not modify any formats above this line!      *****
*****                                                          *****

         To modify control constants prior to running execution:
~user/execution> cp control.constants.input.auv  control.constants.input
or use the following command in a script or on the command line:
      CONSTANTS auv
```

Figure 31. Required format for the file control.constants.input

language, differences in the structure and flow of the program will be encountered. The parsing of the command prompt flags is done by calling the `match_command_line_flags` method from the decide class as `decide.match_command_line_flags(args)`. This method is basically the same algorithm as the previous C code. The mission script file is opened with the method `io.open_File(io.AUVSCRIPTFILENAME)`and assigned to a dummy variable `commandsBuffer`. After this, the mission script is read using the io object method `io.readMissionScript(queue, commandsBuffer)`. Notice that it takes two parameters, the queue where the commands will be saved and a reference to the mission script file that was just recently opened. The `readMissionScript()` method reads the mission script file line by line. When it encounters commented lines identified by "*" or "#" it disregard them. The method uses the queue in order to "push" the command line from the file. At the all the commands lines are saved in the queue. A command line can be composed of a command and a set of parameters like ***position 80 10 5*** or without a parameters like ***thrusters-on***. The parser object is created and network communication are established. The network communication is done by using the Java network socket. Basically a socket is created and input/output streams are acquired. Something good about Java is the enforcing of exception handling. The entire process just described is encapsulated in a try/catch block, which enforces the use of exception handling code in order to recover from exceptions in the process of network connections.

The control coefficients are read from the file `control.constants.input`. This file follows a specific self-documented format as shown in Figure 30. Based of that format and knowing that the format can not changed, the method `get_Control_Coefficients()` from the io object reads the control coefficients. The method basically reads the lines 9, 15, 21, 27, 33, and 39. The line read is then parsed in the respective tokens and since a predefined order in the file exists the respective control coefficient is set from the token just read.

The initial position of the vehicle is then sent to the virtual world by the network socket. Notice that the method from the network class

```
network.write_Telemetry_to_dynamics( "position " + aries.get_x() + " " + aries.get_y() + " " + aries.get_z() + "\n" )
```

gets the values from the object vehicle. The constructor of the vehicle class has a default values in which are the ones used for the initial position.

The control loop method is called with `control.control_loop(ST1000, ST725, act, network, parser)`. Since the approach is the virtual world, the control loop method omit the real world code by commented them. The update and use of the telemetry information is done by calling the respective "set" and "get" methods of the vehicle object.

D. SUMMARY

In this chapter the description of the components of the execution Java code was done. The different packages and classes were briefly described. Since the algorithms for control, parsing commands and flags, and acting are basically the same from the C code, a general description of the object oriented approach for the execution level using Java was discussed in this chapter. Some of the differences from the C code and the Java code is the use of a queue to storage the commands instead of keeping a file open and read from it until the end of the execution level program. The control coefficients are kept in a class and an instantiation provides default constant coefficients as well as the ability to read from a constants file.

VII. EXECUTION JAVA CODE TESTING

A. INTRODUCTION

Testing software components is an essential part of the software developing process. The main objective of testing is to prove that the software product as a minimum meets a set of pre-established acceptance criteria under a prescribed set of environmental circumstances.

The testing of individual classes is somehow difficult for this project. The fact that the C source code is available but without a debugger for the code make the comparison of results extremely difficult. An approach that might be done is to take the step by step execution of the C code and with the features provided by the code of STEP execution and TRACE options, a black box approach could be performed and compare the results of what goes in and what comes out. This takes time and a lot of patience.

Even though testing of software product is a very involved process which includes the verification of the design against the Software Requirements Specifications, the purpose of this chapter is to give a brief description of how the component "debugging" and the integration was performed.

B. COMPONENT TESTING

Code a little, test a little! Component or class testing basically is follows that approach. Debugging works best from a runnable piece of code that gets only small incremental changes before repeated testing. With that idea in mind, the test of the different classes was performed as soon as the class was finish. Some classes had to wait until the development of dependent classes were finished.

1. Vehicle Class

The vehicle class was tested and verified. The constructor successfully creates a vehicle object and all the telemetry variables are initialized with the default values. The "set" and "get" methods successfully update and return the respective variables.

2. Network_Connection Class

The network_connection class was tested and verified. The method `open_network_connection()` successfully creates a network socket and get input and output streams for the virtual world communication. The communication of telemetry between execution Java level code and dynamics using `write_Telemetry_to_dynamics(String telemetry_to_dynamics)` and `read_Telemetry_from_dynamics()` successfully accomplished their purpose. A successful telemetry string was written to dynamics and read back from it.

3. Execution_Flags

The Execution_Flags class holds all the execution flags. This class was implemented using all public variables, therefore the access of the variables can be done directly through the object and not using "set" or "get" methods. The use of the object was tested with the access and setting of the flags always achieved.

4. Commands_Queue and List_Node

The implementation of these two classes provides the queue object to store the commands read from the mission script file. The testing of the "push", "pop", and "print" was done with results as expected.

5. Parser

The parsing of telemetry was tested and results were as expected. The parse of mission script commands results were as expected.

6. IO

This class has several methods to test. The `build_telemetry_string()` method test resulted with a good telemetry string created. The

`build_telemetry_file_header()` method successfully created a header for the telemetry output file. The method `get_Control_Coefficients()` successfully opened, read, and set the control coefficients read from the file control.constant.input. The method `open_File()` is therefore working correctly since it was used on the `get_Control_Coefficients()` method.

The `open_telemetry_save_file()` created the file were the telemetry of the vehicle will be saved every 0.1 seconds successfully. The method `readMissioScript()` reads the mission script command and put the commands read into the queue successfully. The method `recordData()` uses the `save_to_file()` method and methods from network object in order to save the telemetry into the file mission.output.telemetry and into dynamics.

7. Control_Coefficients

The Control_Coefficients class is like the Vehicle class, a variable holder. This class holds all the control coefficients that are in used in the execution level code program. A "set" and "get" methods are responsible for the setting and getting of the coefficients. The testing of this class was conducted with successful results.

8. Decide

The Decide class is composed of three methods. The method `match_command_line_flags()` successfully match the flags passed as parameters from the command prompt and updates the Execution_Flags object. The method `match_commands()` was tested with the commands:

```
position 12 80 5
speed 700
thrusters-on
standoff-distance .75
```

```
waypoint 110 80 15
hover
course 90
wait 10
waypoint 110 95 15
hover
course 180
wait 10
waypoint 20 95 5
hover
course 270
wait 10
hover 12 80 5
course 0
wait 10
speed 0
thrusters-off
```

and successfully match the read command and update the respective information. More test is needed for commands or flags conditions like `TIMESTEP`, `TACTICALPARSE`, `WAITUNTIL`, `REALTIME` and conditions where the sonars are installed. Conditions for the real world are not tested.

9. Data_Processsing

The Data_Processing class has miscellaneous data processing methods like normalization of values. A complete testing of the methods has not been done.

10. Sense

Most of the Sense class methods are related to the real world operations of the AUV. Some portions of the code are commented out because of that reason. The applicable part for the virtual world are not completely tested. Problems with the reading of the rpms have not been solved. The rpms are showing as zero when it is read and the code automatically adjust them to 400. After receiving the command of speed-700 and thrusters-on, the rpms remains on 400 regardless of the command just read.

11. Act

The Act class methods cover a great deal the real world. The code for the real world has been omitted from these methods. Methods like initialization of AD cards and DA cards, if the flag `LOCATIONLAB` is true, a simple return is executed. A complete test of this methods must be performed.

12. Kalman

Even though that the algorithm of the methods in this class are the same as the C code and the translation to Java is relatively the same, the testing of these methods must be performed in order to have a good assurance of the functionality.

13. ST725_sonar

This class has only one method which is `control_ST725_sonar()`. It has not been tested. Some variables have been commented out since no usage where found from the translation of the C code.

14. ST1000_sonar

The ST1000_sonar class is under the same situation as the ST725_sonar class. The detail testing of the class must be performed.

15. Control

The Control class is the biggest class and most complex one. The most important method is the `control_loop()`. This method is like a cyclic executive which provides feedback to the operations of the vehicle. All the methods have been implemented but not tested. A point of observation is that in the method `compute_followlight_controls()` a call to methods `RenderCalculation()`, `SaveImage()`, `CalculatePsiLightReference()`, and `CalculateCspeedDpsiDz()` could not be found in the entire C code. The

commented lines were left in the Java code as reference. In the `control_loop()` method a portion of the code was commented in the part where `CLReaddmod()` appears. This method could not be found in the C source code. The `REALTIME` flag portion of the code was commented. The implementation in the C code uses a data structure that access the cpu clock directly. The implementation in Java needs to be figured out.

16. Execution

The Execution class constitutes the main method for the entire software package. Basically integrates all the classes. It starts with the instantiation of the classes which it does it as expected. The same methodology or flow from the C code is followed in the implementation of the Execution class. Some parts of the C code are not implemented yet like the `EMAIL` part and `LOOPFOREVER` where it performs back-ups and restart the mission. The overall performance is as expected. Detail testing must be done with all the possible combination of commands and flags.

Looking the execution level Java code as an entire package, the program runs with some logical errors. The sending of the telemetry to dynamics and to the file looks to be having problems after a refactoring procedure performed in the code. After sending the initial position to dynamics, the reading of the telemetry from dynamics on the 0.1 seconds is done correctly. Execution perform all the required calculations, build the telemetry for 0.2 seconds and send it to dynamics. When it is time to read from dynamics again it reads a telemetry string for 0.1 seconds, the same one read previously, something that was not happening before the refactoring. Table 5 shows portion of the telemetry output file written by the execution Java code.

	t	x	y	z
auv_state	0	12	80	5
auv_state	0.1	12	80	5
auv_state	0.2	12	80	5
auv_state	0.2	12.0011	80	5
auv_state	0.3	12.0022	80	5
auv_state	0.3	12.0032	80	5
auv_state	0.4	12.0054	80	5
auv_state	0.3	12.0032	80	5
auv_state	0.4	12.0054	80	5
auv_state	0.4	12.0063	80	5
auv_state	0.5	12.0096	80	5
auv_state	0.4	12.0063	80	5

Table 5. Portion of mission.output.telemetry file written by execution Java code

Notice that at 0.4 seconds it seems to repeat the telemetry line. Table 6 shows the portion of the same file but this time written by the C code. Both files correspond to the same mission script file.

	t	x	y	z
uvw_state	0	12	80	5
uvw_state	0.1	12	80	5
uvw_state	0.2	12.001	80	5.001
uvw_state	0.3	12.003	80	5.002
uvw_state	0.4	12.007	80	5.005
uvw_state	0.5	12.011	80	5.008
uvw_state	0.6	12.016	80	5.012
uvw_state	0.7	12.023	80	5.017

Table 6. Portion of mission.output.telemetry file written by execution C code

As shown in table 5, the integration of the execution Java code needs to be tested in the logic flow of the operations. Even though no syntax errors are found, logical error can be noticed, the most difficult kind of errors to be found.

C. SUMMARY

In this chapter the overall integration and testing of the different classes was discussed. It was found that more extensive and detail testing are needed for some classes. The integration of all the classes to compose the execution Java code has some logical or integration errors. These errors most likely come from the control, act, or sense class.

VIII. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

A. INTRODUCTION

The main purpose of this work was to develop execution level Java code for the NPS AUV. The approach taken was to start with the virtual world first and continue the work by extending it to the real world. A high level UML use cases, sequence diagrams, conceptual model and architecture views were developed. A Java source code has been created and the continue development, testing and refactoring of such code is a continuous process.

An indirect goal of this thesis was to show a systematic software engineering process and emphasize the important of such approach in the development of software for complex systems.

In this chapter, thesis goals and results are evaluated. The problem statement will be addressed. Finally, future work will be recommended.

B. RESEARCH CONCLUSIONS

1. Are the Goals Met?

In Chapter III the goals for this thesis were stated. The goals of building a Java based execution level code for the NPS AUV capable of being interchangeable with the real and virtual world without any effect and to prepare the software for the Real-Time Java migration and Java Board integration are partially met. The first phase of the project was met, the creation of a Java execution level code for the virtual world. Even though more testing and refactoring is required, the foundation has been laid. The beginning of a fully working source code of the execution level code in Java has been developed. Research into the Java extension for Real-Time has been conducted and explained in Chapter V. Two Java boards have been purchased for the second phase of the project, the integration of the software into the Java board and the physical devices of the AUV.

2. Execution Java Code

The complexity of the execution level C code in terms of software engineering procedures made understanding very difficult. The use of global variables complicated the situation even further since the track of the variables and the updates of them was not very clear. A different approach was followed. An object-oriented analysis was performed and the identification of objects and the creation of classes was performed. A good understanding of the requirements and legacy code is crucial for the success of a complex project like this one.

A good implementation of the execution level code was created in the Java language. The features of Java provided an excellent way to develop a true object-oriented approach and implementation.

3. Real-Time Java

Real-Time Java was discussed in chapter V. The existing Java core, even though it addresses some real-time issues like timed related methods, some issues like memory management, priority, schedulability, and timing issues were not appropriate for hard real-time systems. Two different groups started to developed the Real-Time Specification for Java (RTSJ) were those issues mentioned above and others are covered and discussed. The Java extension javax.realtime is a product that implements all the requirements stated in the RTSJ. With this extension for the core Java real-time development can be done. The use of a regular JVM can be used but is not assured to comply with the real-time requirements. A special JVM is under development which could provide real-time support and regular core Java code execution without any difference.

4. Java Board

In Chapter V the discussion of the Java board was done. Two Java boards aJ-100EVB from aJile Systems were purchased in order to continue the transition of the execution level code to Java using a real-time environment. A basic discussion of the Java board and programming procedure was performed.

Java boards demonstrated a feasible way to develop software for real-time systems.

C. FUTURE WORK RECOMMENDATIONS

This thesis established the foundation for future work in real-time software using the object oriented language Java and Java boards for autonomous underwater vehicles, or other robotic systems using a systematic software engineering approach.

Additional work is still required in the area of testing, refactoring and continue development of the source code developed through this thesis. The verification of the integration of the different classes that compose the execution Java code must be performed as well as in-depth test of the classes.

The code for the integration with the real world must be performed. The use of a package like javax.comm. where it provides classes for access to parallel and serial ports provides a useful way to communicate with the different sensors and actuators of the vehicle.

Finally the integration of the Java code into the Java board must be performed. Since Java boards offer RTOS, the usage of the package javax.realtime must be use as part of the Java execution level code. Interface with the sensors and actuators must be done and extensive testing must be perform.

D. SUMMARY

In this chapter conclusion and future work recommendations were discussed. The goals of creating a Java based execution level code, a real-time approach and discussion of Java Real-Time, and introduction of the Java board were met.

Java Real-Time is a reality. The features of the language as "write once, run anywhere" can be taken advantage in the real-time development for systems. With a fully object oriented programming language as Java, a strong support for proper software engineering techniques like software reuse, the time for developing software is reduce as well as cost.

The use of Java boards as platform for the real-time environment provides a stable environment for software development. The fact that a RTOS is embedded on the microprocessor reduce overhead, moreover, the direct execution of bytecode into the Java processor.

APPENDIX A EXECUTION JAVA SOURCE CODE

A. SENSE.JAVA

```
/* ____________________________________________________________________________________________
   Title:          Sense.java
   Description:    This class is responsible for the sensing of devices
   Date:           23 May, 2002
   Project:        Execution Java Software for Autonomous Underwater Vehicle,
                   Naval Postgraduate School, Monterey, CA
   Compiler:       JDK 1.3.1
   Author:         Miguel A. Ayala
   Version:        1.0
   ____________________________________________________________________________________________
 */

package mil.navy.nps.auvAries.execution.sense;

import mil.navy.nps.auvAries.execution.globals.Execution_Flags;
import mil.navy.nps.auvAries.execution.vehicle.Vehicle;
import mil.navy.nps.auvAries.execution.act.Act;
import mil.navy.nps.auvAries.execution.hardware.AD_cards;
import mil.navy.nps.auvAries.execution.control.Control;
import mil.navy.nps.auvAries.execution.data_processing.Data_Processing;
import mil.navy.nps.auvAries.execution.input_output.Fmt;

/**
* This class is responsible for the sensing of devices
*/

public class Sense {

   Execution_Flags flags;
   Vehicle vehicle;
   Act act;
   AD_cards cards;
   Control control;
   Data_Processing data_Processing;
   int pulse;
   double local_stbd_rpm;
   double local_port_rpm;
   double depth_cell_bias;
   double z_val0;
   double angle;
   double rate;
   int val;
   double pitch_0;
   double speed_per_rpm;
   // use by read speed
   static int old_count1;
   static int old_count2;
   static boolean start;
   int count;
   //  unsigned char lobyte,hibyte;
```

```
double freq;
double avg_speed;
final int PORT_PROP = 0;
final int STBD_PROP = 1;
final int BOW_VERTICAL = 2;
final int BOW_LATERAL = 3;
final int STERN_VERTICAL = 4;
final int STERN_LATERAL = 5;
double value;
double rpm;
double battery_voltage;
double motor_gyro_battery_voltage;
double leak;

/**
 * Sense Class cnostructor.
 * @param:       Vehicle object reference and Data_Processing object reference
 * @return:      None
 */
public Sense(Vehicle vehicleObj, Data_Processing Data_ProcessingRef) {

   vehicle = vehicleObj;
   data_Processing = Data_ProcessingRef;
   pulse = 0;
   local_stbd_rpm = 0.0;
   rpm = 0.0;
   value = 0.0;
   depth_cell_bias = 0.0;
   z_val0 = 0.0;
   angle = 0.0;
   rate = 0.0;
   val = 0;
   speed_per_rpm = 0.0;
   // use by read speed
   old_count1 = 0;
   old_count2 = 0;
   start = true;
   count = 0;
   // unsigned char lobyte,hibyte;
   freq = 0.0;
   avg_speed = 0.0;
   battery_voltage = 0.0;
         motor_gyro_battery_voltage = 0.0;
         leak = 0.0;

}

/**
 * Reads rpm from STBD_PROP
 * @param:       None
 * @return:      double
 */
public double read_stbd_motor_rpm() { // VERIFIED
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[start  read_stbd_motor_rpm ()]\n");
   }
```

```
      local_stbd_rpm = read_motor(STBD_PROP);
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("[finish read_stbd_motor_rpm () returns " +
            local_stbd_rpm + "]\n");
      }
      return (local_stbd_rpm);
   } // end read_stbd_motor_rpm ()

   /**
    * Returns rpm from single propellor or thruster
    * @param:       None
    * @return:      double
    */
   public double read_motor(int motor) { // VERIFIED

/*     motor = 0  Left  Propeller            PORT_PROP      RPM
            1  Right Propeller            STBD_PROP      RPM
            2   Bow Vertical Thruster      BOW_VERTICAL   volts
            3   Bow Lateral  Thruster     STERN_VERTICAL   volts
            4  Stern Vertical Thruster      BOW_LATERAL    volts
            5  Stern Lateral  Thruster     STERN_LATERAL    volts
*/

      int count;
      double freq, rps;
      //                unsigned char lobyte,hibyte;
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("[start  read_motor ()]\n");
      }
      if (flags.LOCATIONLAB == false) { // in water
         System.out.println("IN WATER CALCULATIONS");

/*
               switch ( motor ) {

                  case PORT_PROP:
                        //  Sel Cntr 1 HOLD Reg. Card 3
                           write_tim1a ( 3, tim_1a_control_reg, 17 );
                           lobyte = read_tim1ac1 ( 3, tim_1a_data_reg );
                           hibyte = read_tim1ac1 ( 3, tim_1a_data_reg );
                           count = (int) ( 256 * hibyte ) + (int) lobyte;
                           if ( v_dls < 512 ) {
                              count = -count; // Account for Direction of Rot.
                     }
                           break;

                        case STBD_PROP:
                        // Sel Cntr 2 HOLD Reg. Card 3
                           write_tim1a ( 3, tim_1a_control_reg, 18 );
                           lobyte = read_tim1ac1 ( 3, tim_1a_data_reg );
                           hibyte = read_tim1ac1 ( 3, tim_1a_data_reg );
                           count = (int) ( 256 * hibyte ) + (int) lobyte;
                           if( v_drs < 512 ) {
                              count = -count; // Account for Direction of Rot.
                           }
                           break;
```

```
            case BOW_VERTICAL:
// Sel Cntr 1 HOLD Reg. Card 2
                write_tim1a( 2, tim_1a_control_reg, 17 );
                lobyte = read_tim1ac1 ( 2, tim_1a_data_reg );
                hibyte = read_tim1ac1 (2, tim_1a_data_reg );
                count = (int) ( 256 * hibyte ) + (int) lobyte;
                if( v_dbvt < 512 ) {
                        count = -count; // Account for Direction of Rot.
                }
                break;

            case STERN_VERTICAL:
// Sel Cntr 2 HOLD Reg. Card 2
                write_tim1a (2, tim_1a_control_reg, 18 );
                lobyte = read_tim1ac1 ( 2, tim_1a_data_reg );
                hibyte = read_tim1ac1 (2, tim_1a_data_reg );
                count = (int) ( 256 * hibyte ) + (int) lobyte;
                if( v_dblt < 512 ) {
                   count = -count; // Account for Direction of Rot.
                }
                break;

            case BOW_LATERAL:
// Sel Cntr 3 HOLD Reg. Card 2
                write_tim1a (2, tim_1a_control_reg, 19 );
                lobyte = read_tim1ac1 (2, tim_1a_data_reg );
                hibyte = read_tim1ac1 ( 2, tim_1a_data_reg );
                count = (int) ( 256 * hibyte ) + (int) lobyte;
                if ( v_dsvt < 512 ) {
                   count = -count; // Account for Direction of Rot.
                }
                break;

            case STERN_LATERAL:
// Sel Cntr 4 HOLD Reg. Card 2
                        write_tim1a (2, tim_1a_control_reg, 20 );
                   lobyte = read_tim1ac1 ( 2, tim_1a_data_reg );
                        hibyte = read_tim1ac1 (2, tim_1a_data_reg );
                        count = (int) ( 256 * hibyte ) + (int) lobyte;
                        if ( v_dslt < 512 ) {
                           count = -count; // Account for Direction of Rot.
                        }
                        break;

            default:
            if ( flags.DISPLAYSCREEN ) {
                        System.out.println ( "[read_motor () error:
      illegal motor value " + motor + "]\n" );
                        }
                        break;
            }

            if ( count != 0 ) {

            // F1 (1 Mhz) The 4.0 is in there
```

```
                    freq = ( 1.0 / count ) * 4.0 * Math.pow( 10.0, 6.0 );
                                                // as a scale factor from God
                }
                else {

                // Sensor Not Counting
                    freq = 0.0;
                }

                // 500 Counts Per Rev
                rps = ( freq / 500.0 );
                if(( Math.abs( rps ) < 1.0 ) || ( Math.abs( rps ) > 1000.0 )) {
                                rps = 0.0;
                }
                if ( flags.TRACE && flags.DISPLAYSCREEN ) {
                    System.out.println ( "[finish read_motor () returns "
                                                                + rps + " \n" );
                }

                value = rps * 60.0; // convert from per-seconds to per-minutes

*/
        }
        else { // LOCATIONLAB == true
            value = rpm;
        }
        return value;
    }

    /**
     * Returns rpm frpm Port prop
     * @param:                None
     * @return:       double
     */
    public double read_port_motor_rpm() { // VERIFIED
        int pulse;
        local_port_rpm = 700;
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[start  read_port_motor_rpm ()]\n");
        }
        local_port_rpm = read_motor(PORT_PROP);
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[finish read_port_motor_rpm () returns "
                                                + local_port_rpm + "]\n ");
        }
        return (local_port_rpm);
    } // end read_port_motor_rpm ()

    /**
     * Return depth in feet
     * @param:       None
     * @return:       double
     */
    public double read_depth() { // VERIFIED
        int val = 0;
        double new_z = 0.0; // zz in dave's execf.c code /
```

```
      double z_offset = 0.0;
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("\n[start read_depth ()]");
      }
      if (flags.LOCATIONLAB && flags.DEADRECKON) {
         new_z = control.z_command;
      }
      else if (flags.LOCATIONLAB) {
         new_z = vehicle.get_z(); // no change, use virtual world value
      }
      else { // in-water

/*
            // val = adc1(DEPTH_CELL_CH);  // Channel 7

            // 0.0728 = 0.0182 * 4.0
            // Since A/D now has 0-1023 range instead of 0-4095
            // new_z = 0.0728*( (double) (val - z_val0)) + z_offset;

            // adc2 card has 0 - 4095 resolution
            val = get_adc2 ( cards.DEPTH_CELL_CH, 0 );
            new_z = 0.0182 * ( ( double ) ( val - z_val0 )) + z_offset;

            // Calibration for Signal Amp
            //new_z = 0.0034285*( (double) (z_val0 - val)) + z_offset;
 */

         System.out.println("IN WATER CALCULATIONS");
      }
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("\n[finish read_depth (), returns "
                                                + new_z + "\n");
      }
      return (new_z + depth_cell_bias);
   } // end read_depth ()

   /**
    * Returns yaw posture in world coordinates in degrees
    * @param:      None
    * @return:     double
    */
   public double read_psi() { // VERIFIED
      //     unsigned short psi_bit;
      //     int psi_bit_int,psi_bit_old_int,delta_psi_bit;
      double tpi;
      //
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("[start read_psi ()]\n");
      }
      if (flags.LOCATIONLAB && flags.DEADRECKON) {
         angle = control.psi_command;
      }
      else if (flags.LOCATIONLAB) {
                              // no change, use virtual world value
         vehicle.set_psi(vehicle.get_psi());
         angle = vehicle.get_psi(); // set up for function return
```

```
        }
        else { // in-water
           System.out.println("IN WATER CALCULATIONS");

/*         psi_bit = Read_PortAB ( 0xFFF00700 );
              psi_bit &= 0x3FFF;
        psi_bit_int = psi_bit;
        psi_bit_old_int = psi_bit_old;

        delta_psi_bit = psi_bit_int - psi_bit_old_int;
        psi_bit_old = psi_bit;

        if ( Math.abs ( delta_psi_bit ) > 10000 ) {

              wrap_count = wrap_count - delta_psi_bit / Math.abs (delta_psi_bit);
        }

        angle = start_psi + Math.toDegrees((( read_heading () -
                     dg_offset + 2.0 * Math.PI ((double) wrap_count ))));

        if ( Math.abs( angle ) < 0.0001 ) {
              angle = 0.0;
        }
        //printf("%f %f %f %d %d\n",
        //       angle,read_heading (),dg_offset,wrap_count,psi_bit);
*/
        }
        if (flags.TRACE && flags.DISPLAYSCREEN) {
           System.out.println("[finish read_psi () returns " + angle + " ]\n");
        }
        return (data_Processing.normalize(angle));
     } // end read_psi ()

     /**
      * Returns roll rate in Degrees/sec
      * @param:       None
      * @return:      double
      */
     public double read_roll_rate_gyro() { // VERIFIED
        if (flags.TRACE && flags.DISPLAYSCREEN) {
           System.out.println("[start  read_roll_rate_gyro ()]\n");
        }
        if (flags.LOCATIONLAB) {
           rate = vehicle.get_p(); // no change, use virtual world value
           if (Math.abs(rate) < 0.0001) {
              rate = 0.0;
           }
        }
        else { // in-water
           System.out.println("IN WATER CALCULATIONS");

              /*          val = get_adc2 ( cards.ROLL_RATE_CH, 0 );
                                 // Next two lines from old method
                                 // val = val >> 2;  // Quick fix for new res
                                 // rate = ( roll_rate_0 / 3.2113 - .31062 * val ) / 57.295779;
                                 rate = Math.toDegrees ( 0.07785 * ( roll_rate_0 - val ) / 57.295779);
```

```
                    if ( Math.abs ( rate ) < 0.0001 ) {
                        rate = 0.0;
                    }
            */
        }
        rate = data_Processing.normalize2(rate);
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[finish read_roll_rate_gyro () returns " + rate
                                                    + "]\n");
        }
        return (rate);
    }

    /**
     * Returns pitch rate in Degrees/sec
     * @param:      None
     * @return:     double
     */
    public double read_pitch_rate_gyro() { // VERIFIED
        val = 0;
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[start  read_pitch_rate_gyro ()]\n");
        }
        if (flags.LOCATIONLAB) {
            rate = vehicle.get_q(); // no change, use virtual world value
            if (Math.abs(rate) < 0.0001) {
                rate = 0.0;
            }
        }
        else { // in-water
            System.out.println("IN WATER CALCULATIONS");

/*          val = get_adc2 ( cards.PITCH_RATE_CH, 0 );
         // Next two lines from old method
         // val = val >> 2;*/

           /* Quick fix for new res
         // rate = ( pitch_rate_0 / 13.69399 - .0730001 * val ) / 57.295779;
            rate = Math.toDegrees( 0.01825 * (pitch_rate_0 - val) / 57.295779);
            if ( Math.abs ( rate ) < 0.0001 ) {
            rate = 0.0;
                }
*/
        }
        rate = data_Processing.normalize2(rate);
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[finish read_pitch_rate_gyro () returns " + rate
                                                    + "]\n");
        }
        return (rate);
    } // end read_pitch_rate_gyro ()

    /**
     * Return yaw rate in degrees/sec
     * @param:      None
     * @return:     double
```

```
 */
public double read_yaw_rate_gyro() { // VERIFIED
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[start  read_yaw_rate_gyro ()]\n");
   }
   if (flags.LOCATIONLAB) {
      rate = vehicle.get_r(); // no change, use virtual world value
      if (Math.abs(rate) < 0.0001) {
         rate = 0.0;
      }
   }
   else { // in-water
      System.out.println("IN WATER CALCULATIONS");

/*       // Below for adc1 Card
    // val = adc1(YAW_RATE_CH); // Channel 10
    // rate = 2.78 * (( (double) yaw_rate_0) / 13.653216 - 0.0732362
                                                    * ( (double) val) ) / 57.295779;

         val = get_adc2( cards.YAW_RATE_CH, 0 );
    // Next two lines from old method
    // val = val >> 2; // Quick fix for new res
    // rate = 2.78 * (yaw_rate_0 / 13.653216 - .0732362 * val) / 57.295779;
         rate = Math.toDegrees ( 0.0509 * ( yaw_rate_0 - val ) / 57.295779 );
      if ( Math.abs ( rate ) < 0.0001 ){
         rate = 0.0;
                              }
*/
   }
   rate = data_Processing.normalize2(rate);
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[finish read_yaw_rate_gyro () returns " + rate
                                                    + "]\n");
   }
   return (rate);
} // end read_yaw_rate_gyro ()

/**
 * Return roll angles in degrees
 * @param:      None
 * @return:     double
 */
public double read_roll_angle() { // VERIFIED
   int val = 0;
   double angle = 0.0;
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[start  read_roll_angle ()]\n");
   }
   if (flags.LOCATIONLAB) {
      angle = vehicle.get_phi(); // no change, use virtual world value
      if (Math.abs(angle) < 0.0001) {
         angle = 0.0;
      }
   }
   else { // in-water
      System.out.println("IN WATER CALCULATION");
```

```
/*          val = get_adc2 ( cards.ROLL_ANGLE_CH, 0 );
         // Next three lines from old method
            // val = val >> 2; // Quick fix for new res
                                                // convert to radians
            // angle = (( 516.578 - val ) / 5.7572 ) / 57.295779;
            // angle = ( -.1737 * val + .1737 * roll_0 )/ 57.295779;
            angle = 0.043425 * ( roll_0 - val ) / 57.295779;
            if ( Math.abs ( angle ) < 0.0001 ) {
            angle = 0.0;
            }
*/
      }
      angle = data_Processing.normalize2(angle);
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("[finish read_roll_angle () returns " + angle
                                                        + "]\n");
      }
      return (angle);
   } // end of read_roll_angle()

   /**
    * Return pitch angles in degrees
    * @param:       None
    * @return:      double
    */
   public double read_pitch_angle() { // VERIFIED
      int val = 0;
      double angle = 0.0;
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("[start  read_pitch_angle ()]\n");
      }
      if (flags.LOCATIONLAB) {
         angle = vehicle.get_theta(); // no change, use virtual world value
         if (Math.abs(angle) < 0.0001) {
            angle = 0.0;
         }
      }
      else { // in-water
         System.out.println("IN WATER CALCULATIONS");

/*          val = get_adc2 ( cards.PITCH_ANGLE_CH, 0 );
       // Next three lines from old method
       // val = val >> 2;         // Quick fix for new res
                                  //convert to radians
       // angle = (( 520.153 - val ) / 8.340 ) / 57.295779;
       // angle = (( -.1199 * val + .1199 * pitch_0 ) / 57.295779 );
            angle = Math.toDegrees ( 0.02997 * ( pitch_0 - val ) / 57.295779 );
            if ( Math.abs ( angle ) < 0.0001 ) {
            angle = 0.0;
            }
*/
      }
      angle = data_Processing.normalize2(angle);
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("[finish read_pitch_angle () returns " + angle
```

```
                                                    + "]\n");
        }
        return (angle);
    }

    /**
     * Return heading angle with respect to local magnetic north in radians
     * from directional gyro
     * @param:      None
     * @return:     double
     */
    public double read_heading() { // VERIFIED
        //    unsigned short dg_bit;
        double angle = 0.0;
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[start  read_heading ()]\n");
        }
        if (flags.LOCATIONLAB && (flags.DEADRECKON == false)) {
            angle = vehicle.get_psi();
            if (Math.abs(angle) < 0.0001) {
                angle = 0.0;
            }
        }
        else if (flags.LOCATIONLAB && flags.DEADRECKON) {
            angle = control.psi_command;
            if (Math.abs(angle) < 0.0001) {
                angle = 0.0;
            }
        }
        else { // in-water
            System.out.println("IN WATER CALCULATIONS");

/*          // dg_bit = Read_PortAB( MFI_BASE )
            dg_bit = Read_PortAB ( 0xFFF00700 );  // why not a #define here? <
            // dg_bit = 10000;
            dg_bit &= 0x3FFF;

            angle = ( 3.8350e-4 ) * ( (double) dg_bit );
            // printf("Angle = %f %d\n",angle, dg_bit );
            // if ( fabs( angle ) < 0.001) angle = 0.0;//
*/
        }
        angle = data_Processing.normalize(angle);
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[finish read_heading () returns " + angle
                                                    + "]\n");
        }
        return (angle);
    }

    /**
     * Return speed and filter it
     * @param:      None
     * @return:     double
     */
    public double read_speed() { // VERIFIED
```

```
    if (flags.TRACE && flags.DISPLAYSCREEN) {
       System.out.println("[start  read_speed (), LOCATIONLAB = "
                                                  + flags.LOCATIONLAB + "]\n");
    }
    if (flags.LOCATIONLAB) {
       if (flags.TRACE && flags.DISPLAYSCREEN) {
          System.out.println("[finish read_speed () returns "
                                                  + vehicle.get_speed() + "]\n");
       }
       return (vehicle.get_speed()); // from virtual world-paddlewheel
                                          //          speed = u = surge
    }
    else if (flags.DEADRECKON) {
       if (flags.TRACE && flags.DISPLAYSCREEN) {
          System.out.println("[finish read_speed () DEADRECKON returns ");
       }
       avg_speed = (speed_per_rpm * (vehicle.get_port_rpm()
                                       + vehicle.get_stbd_rpm()) / 2.0);
       if (flags.TRACE && flags.DISPLAYSCREEN) {
          System.out.println("Avg speed " + avg_speed + " ]\n");
       }
       return (avg_speed);
    }

/*        else {
            // I think this is Dave's speed averaging code
                    if ( start ) {
                       old_count1 = 0;
                       old_count2 = 0;
                       start = false;
                    }

                    write_tim1a ( 3, tim_1a_control_reg, 19 );
                    lobyte = read_tim1ac1 ( 3, tim_1a_data_reg );
                    hibyte = read_tim1ac1( 3, tim_1a_data_reg );
                    count = (int) ( 256 * hibyte ) + (int) lobyte;

                    if(( old_count1 == count ) &&
                      ( old_count2 == count )) {

           old_count1 = old_count2;
                       old_count2 = count;
                       return( 0.0 );
                    }

                    old_count1 = old_count2;
                    old_count2 = count;

        }
*/

    if (flags.TRACE && flags.DISPLAYSCREEN) {
       System.out.println("[finish read_speed () returns " + avg_speed
                                                  + "]\n");
    }
    if (count != 0) {
```

```
            freq = (1.0 / (2.0 * count)) * 4.0 * 10000.0; // F3 (10,000 Hz)
            if (freq >= 17) {
                freq = freq * 2.0;
            }
            else if (freq > 15) {
                freq = freq * (1.0 + ((freq - 15.0) / 2.0));
            }
        }
        else {
            // Sensor Not Counting
            freq = 0.0;
        }
    // Polyfit for Calibration data in marco:/vault2/marco/AUV/turbo_probe/tp.m
        if (freq >= 4999.0) {
            return (Math.abs(vehicle.get_speed()));
        }
        else {
            return (0.00000973701619 * freq * freq + 0.02934498907499 * freq
                        + 0.15845400316984);
        }
    } // end of read speed()

    /**
     *
     */
            public double read_motor_gyro_battery_voltage() {

        return motor_gyro_battery_voltage;

    }

    /**
     *
     */
    public double read_computer_battery_voltage() {
        return battery_voltage;
    }

    /**
     *
     */
    public boolean leak_check() {
        return false;
    }

} // end of Sense class
```

B. DECIDE.JAVA

```
/* ____________________________________________________________________________
  Title:          Decide.java
  Description:    The purpose of this class is to make the decisions part of
                  the AUV. Take the flags parameters inputted by the user and
                  based on the parameters sets the respective flags for the
                  AUV operections.  This flags are contained in an object
                  called flags from the class Execution_Flags.  Once the flags
                  are set this object is used by the rest of the execution.
  Date:           April 10, 2002
  Project:        Execution Java Software for Autonomous Underwater Vehicle,
                  Naval Postgraduate School, Monterey, CA
  Compiler:       JDK 1.3.1
  Author:         Miguel A. Ayala
  Version:        1.0
  ____________________________________________________________________________
*/

package mil.navy.nps.auvAries.execution.decide;

/** The purpose of this class is to make the decisions part of
*          the        AUV. Take the flags parameters inputted by the user and
*   based on the parameters sets the respective flags for the
*   AUV operections.  This flags are contained in an object
*   called flags from the class Execution_Flags.  Once the flags
*   are set this object is used by the rest of the execution.
*/
import mil.navy.nps.auvAries.execution.globals.Execution_Flags;
import mil.navy.nps.auvAries.execution.input_output.*;
import mil.navy.nps.auvAries.execution.control.*;
import mil.navy.nps.auvAries.execution.vehicle.Vehicle;
import mil.navy.nps.auvAries.execution.data_processing.Data_Processing;
import mil.navy.nps.auvAries.execution.act.Act;
import mil.navy.nps.auvAries.execution.network.Network_Connection;
import java.util.StringTokenizer;
import java.io.*;

/**
* This class match the flags provided by the user in the command prompt and
* read by the program as parameters and the begining of the main program.
* This class also match the commands read from the mission script file.
*/
public class Decide {
   IO io;
   Execution_Flags flags;
   Control_Coefficients control_coefficients;
   Control control;
   Vehicle vehicle;
   Parser parser;
   Data_Processing data_Processing;
   Act act;
   Network_Connection network;
   int numargs;
   int index, i;
   public String[] parameters;
```

```
boolean return_value;
double time_postgps_dive;
double time_gps_complete;
double z_command;
double previous_z_command;
double time_next_command; // this one I think is coming from control
public final int TRUE = 1;
public final int FALSE = 0;
public final int RELATIVE = 0;
private String inputCommand;

/**
 *  Constructor
 *  @param:      Execution_Flags, IO, Control, Control_Coefficients,
 *                  Vehicle, Act, Data_Processing references.
 *  @return:     None
 *  @exception:   None
 */
public Decide( Execution_Flags execFlags, IO ioObj, Control controlRef,
                              Control_Coefficients control_coefficientsRef,
                              Vehicle vehicleObj, Act actObj,
            Data_Processing Data_ProcessingObj,
            Network_Connection networkRef ) {

      flags = execFlags;
      io = ioObj;
      data_Processing = Data_ProcessingObj;
      control = controlRef;
      control_coefficients = control_coefficientsRef;
      vehicle = vehicleObj;
      act = actObj;
      network = networkRef;
      int index, read_another_line, parameters_read;
      i = 0;
      index = 0;
      numargs = 0;
      boolean return_value = false;
      //         double time_postgps_dive = 0.0;
      //         double time_gps_complete = 0.0;
}

/**
 * The method parses the command line arguments and match them against a predifined
 * list of flags.  If a match is found then that flag is set.
 *  @param:      String. Arguments from the command line.
 *  @return:     None
 */

public void match_command_line_flags(String[] argument) {

   for (index = 0; index < argument.length; index++) {
      if ((argument[index].equalsIgnoreCase("HELP"))
         || (argument[index].equalsIgnoreCase("?"))
         || (argument[index].equalsIgnoreCase("/?"))
         || (argument[index].equalsIgnoreCase("-?"))) {
```

```
        printTrace_Display("[ print_help ] ");
        io.print_valid_keywords();
        System.exit(1);
}
else if ((argument[index].equalsIgnoreCase("KEYBOARD"))
   || (argument[index].equalsIgnoreCase("KEY-BOARD"))
   || (argument[index].equalsIgnoreCase("KEYBOARD-INPUT"))
   || (argument[index].equalsIgnoreCase("KEYBOARDINPUT"))) {

        printTrace_Display("[ KEYBOARDINPUT = TRUE ] ");
        flags.KEYBOARDINPUT = true;
}
else if (argument[index].equalsIgnoreCase("TRACE")) {
   printTrace_Display("[ TRACE = TRUE ] ");
   flags.TRACE = true;
}
else if ((argument[index].equalsIgnoreCase("TRACEOFF"))
   || (argument[index].equalsIgnoreCase("TRACE-OFF"))
   || (argument[index].equalsIgnoreCase("NOTRACE"))
   || (argument[index].equalsIgnoreCase("NO-TRACE"))) {

        printTrace_Display("[ TRACE = FALSE ] ");
        flags.TRACE = false;
}
else if ((argument[index].equalsIgnoreCase("DISPLAYSCREEN"))
   || (argument[index].equalsIgnoreCase("DISPLAY-SCREEN"))
   || (argument[index].equalsIgnoreCase("DISPLAY"))) {

        printTrace_Display("[ DISPLAYSCREEN = TRUE ] ");
        flags.DISPLAYSCREEN = true;
}
else if ((argument[index].equalsIgnoreCase("LOOPFOREVER"))
   || (argument[index].equalsIgnoreCase("LOOP-FOREVER")) ||
   (argument[index].equalsIgnoreCase("LOOP"))) {

        printTrace_Display("[ LOOP-FOREVER ] ");
        flags.LOOPFOREVER = true;
}
else if ((argument[index].equalsIgnoreCase("LOOPONCE"))
   || (argument[index].equalsIgnoreCase("LOOP-ONCE"))) {

                printTrace_Display("[ LOOP-ONCE ] ");
                flags.LOOPFOREVER = false;
}
else if ((argument[index].equalsIgnoreCase("LOOPFILEBACKUP")) ||
   (argument[index].equalsIgnoreCase("LOOP-FILE-BACKUP"))) {

        printTrace_Display("[ LOOPFILEBACKUP ] ");
        flags.LOOPFILEBACKUP = true;
}
//*****************************************************************
//
// =========>>> NEED TO FIND OUT HOW TO DO THIS ONE
//
//*****************************************************************
else if ((argument[index].equalsIgnoreCase("CONSTANTSFILE"))
```

```
   || (argument[index].equalsIgnoreCase("CONSTANTS"))) {
   printTrace_Display("[ LOOPFILEBACKUP ] ");
}
//*****************************************************************
else if ((argument[index].equalsIgnoreCase("ENTERCONTROLCONSTANTS"))
 || (argument[index].equalsIgnoreCase("ENTER-CONTROL-CONSTANTS"))) {
      printTrace_Display("[ ENTERCONTROLCONSTANTS ] ");
      flags.ENTERCONTROLCONSTANTS = true;
}
else if ((argument[index].equalsIgnoreCase("SHOWCONTROLCONSTANTS"))
 ||  (argument[index].equalsIgnoreCase("SHOW-CONTROL-CONSTANTS"))) {
      printTrace_Display(" [ SHOW-CONTROL-CONSTANTS ] ");
      flags.SHOWCONTROLCONSTANTS = true;
}

else if ((argument[index].equalsIgnoreCase("TACTICAL"))
   || (argument[index].equalsIgnoreCase("TACTICAL-HOST"))
   || (argument[index].equalsIgnoreCase("TACTICALHOST"))
   || (argument[index].equalsIgnoreCase("STRATEGIC"))
   || (argument[index].equalsIgnoreCase("STRATEGILHOST"))
   || (argument[index].equalsIgnoreCase("STRATEGIC-HOST"))) {
      flags.TACTICAL = true;
      index++;
      if (index >= argument.length) {
         io.print_valid_keywords();
      }
      else {
         flags.KEYBOARDINPUT = false;
         argument[index] = network.TACTICAL_REMOTE_HOST_NAME;
      }
}
else if ((argument[index].equalsIgnoreCase("NO-TACTICAL"))
   || (argument[index].equalsIgnoreCase("TACTICAL-OFF"))) {
   flags.TACTICAL = false;
}
else if ((argument[index].equalsIgnoreCase("SONARTRACE"))
   || (argument[index].equalsIgnoreCase("SONAR-TRACE")) ||
   (argument[index].equalsIgnoreCase("SONAR-TRACE-ON"))) {
      printTrace_Display("[ SONAR-TRACE ]");
      flags.SONARTRACE = true;
}
else if ((argument[index].equalsIgnoreCase("SONARTRACEOFF")) ||
   (argument[index].equalsIgnoreCase("SONAR-TRACE-OFF"))) {
      printTrace_Display("[ SONAR-TRACE-OFF ]");
      flags.SONARTRACE = false;
}
else if ((argument[index].equalsIgnoreCase("SONARINSTALLED")) ||
   (argument[index].equalsIgnoreCase("SONAR-INSTALLED"))) {
      printTrace_Display("[ SONAR-INSTALLED ]");
      flags.ST725INSTALLED = true;
      flags.ST1000INSTALLED = true;
}
else if ((argument[index].equalsIgnoreCase("NOSONARINSTALLED")) ||
   (argument[index].equalsIgnoreCase("NO-SONAR-INSTALLED"))) {
      printTrace_Display("[ NO-SONAR-INSTALLED ]");
      flags.ST725INSTALLED = false;
```

```
        flags.ST1000INSTALLED = false;
}
else if ((argument[index].equalsIgnoreCase("ST1000INSTALLED")) ||
   (argument[index].equalsIgnoreCase("ST1000-INSTALLED"))) {
        printTrace_Display("[ ST1000-INSTALLED ]");
        flags.ST1000INSTALLED = true;
}
else if ((argument[index].equalsIgnoreCase("NOST1000INSTALLED")) ||
   (argument[index].equalsIgnoreCase("NO-ST1000-INSTALLED"))) {
        printTrace_Display("[ NO-ST1000-INSTALLED ]");
        flags.ST1000INSTALLED = false;
}
else if ((argument[index].equalsIgnoreCase("ST725INSTALLED")) ||
   (argument[index].equalsIgnoreCase("ST725-INSTALLED"))) {
        printTrace_Display("[ ST725-INSTALLED ]");
        flags.ST725INSTALLED = true;
}
else if ((argument[index].equalsIgnoreCase("NOST725INSTALLED")) ||
   (argument[index].equalsIgnoreCase("NO-ST725-INSTALLED"))) {
        printTrace_Display("[ NO-ST725-INSTALLED ]");
        flags.ST725INSTALLED = false;
}
else if ((argument[index].equalsIgnoreCase("PARALLELPORTTRACE")) ||
   (argument[index].equalsIgnoreCase("PARALLEL-PORT-TRACE"))) {
        printTrace_Display("[ PARALLEL-PORT-TRACE ]");
        flags.PARALLELPORTTRACE = true;
}
else if ((argument[index].equalsIgnoreCase("VIRTUALHOST"))
   || (argument[index].equalsIgnoreCase("VIRTUAL-HOST")) ||
   (argument[index].equalsIgnoreCase("VIRTUAL"))
   || (argument[index].equalsIgnoreCase("REMOTE"))
   || (argument[index].equalsIgnoreCase("REMOTEHOST"))
   || (argument[index].equalsIgnoreCase("REMOTE-HOST"))
   || (argument[index].equalsIgnoreCase("DYNAMIC"))
   || (argument[index].equalsIgnoreCase("DYNAMICS"))) {
        i++;
        if (i >= numargs) {
            io.print_valid_keywords();
        }
        else {
            io.VIRTUAL_WORLD_REMOTE_HOST_NAME = argument[index];
            printTrace_Display("[VIRTUAL-HOST ] "
                + io.VIRTUAL_WORLD_REMOTE_HOST_NAME);
        }
}
//*************************************
// NEED TO VERIFY THIS ONE
//*************************************
else if ((argument[index].equalsIgnoreCase("TELEMETRY"))
   || (argument[index].equalsIgnoreCase("TELEMETRYFILE")) ||
   (argument[index].equalsIgnoreCase("TELEMETRY-FILE"))) {
        i++;
        if (i >= numargs) {
            io.print_valid_keywords();
        }
        else {
```

```
        io.TELEMETRY_FILE_NAME = argument[index];
        // if (telemetry_file = fopen (TELEMETRYFILENAME,"r")) {
        printTrace_Display("[%s %s]\n" + (numargs - 1)
           + io.VIRTUAL_WORLD_REMOTE_HOST_NAME);
        flags.REPLAY = true;
        // force SILENT to prevent pronunciation of telemetry
        printTrace_Display("\n[SILENT]\n");

      }
}
else if ((argument[index].equalsIgnoreCase("REALTIME"))
   || (argument[index].equalsIgnoreCase("REAL-TIME"))) {
   printTrace_Display("[REALTIME] ");
   flags.REALTIME = true;
}
else if ((argument[index].equalsIgnoreCase("NOREALTIME"))
   || (argument[index].equalsIgnoreCase("NO-REALTIME"))
   ||
   (argument[index].equalsIgnoreCase("NO-REAL-TIME"))
   || (argument[index].equalsIgnoreCase("NOWAIT"))
   || (argument[index].equalsIgnoreCase("NO-WAIT"))
   || (argument[index].equalsIgnoreCase("NOPAUSE"))
   || (argument[index].equalsIgnoreCase("NO-PAUSE"))) {
      printTrace_Display("[NOWAIT]");
      flags.REALTIME = false;
}
else if ((argument[index].equalsIgnoreCase("EMAIL"))
   || (argument[index].equalsIgnoreCase("EMAIL-ON"))
   || (argument[index].equalsIgnoreCase("E-MAIL"))
   || (argument[index].equalsIgnoreCase("E-MAIL-ON"))
   || (argument[index].equalsIgnoreCase("EMAILON"))) {
      printTrace_Display("[EMAIL ON]");
      flags.EMAIL = true;
}
else if ((argument[index].equalsIgnoreCase("EMAILOFF"))
   || (argument[index].equalsIgnoreCase("E-MAILOFF"))
   || (argument[index].equalsIgnoreCase("EMAIL-OFF"))
   || (argument[index].equalsIgnoreCase("E-MAIL-OFF"))
   || (argument[index].equalsIgnoreCase("NO-E-MAIL"))
   || (argument[index].equalsIgnoreCase("NO-EMAIL"))
   || (argument[index].equalsIgnoreCase("NO-E-MAIL"))
   || (argument[index].equalsIgnoreCase("NOEMAIL"))) {
      printTrace_Display("[NO EMAIL]");
      flags.EMAIL = false;
}
else if ((argument[index].equalsIgnoreCase("LOCATIONLAB"))
   || (argument[index].equalsIgnoreCase("LOCATION-LAB"))) {
   flags.LOCATIONLAB = true;
}
else if ((argument[index].equalsIgnoreCase("TETHER"))
   || (argument[index].equalsIgnoreCase("TETHERED"))) {
   flags.LOCATIONLAB = true;
   flags.DISPLAYSCREEN = true;
   flags.REALTIME = true;
}
else if ((argument[index].equalsIgnoreCase("UNTETHER"))
```

```
   || (argument[index].equalsIgnoreCase("UNTETHERED"))
   || (argument[index].equalsIgnoreCase("NOTETHER"))
   || (argument[index].equalsIgnoreCase("NO-TETHER"))) {
      flags.LOCATIONLAB = false;
      flags.DISPLAYSCREEN = false;
      flags.REALTIME = true;
}
else if ((argument[index].equalsIgnoreCase("TEXT"))
   || (argument[index].equalsIgnoreCase("TEXT-ON"))) {
   flags.DISPLAYSCREEN = true;
}
else if ((argument[index].equalsIgnoreCase("NOTEXT"))
   || (argument[index].equalsIgnoreCase("NO-TEXT"))) {
   flags.DISPLAYSCREEN = false;
}
else if ((argument[index].equalsIgnoreCase("GYROERROR"))
   || (argument[index].equalsIgnoreCase("GYRO-ERROR"))
   || (argument[index].equalsIgnoreCase("GYRO_ERROR"))) {
      index++;
      if (index > numargs) {
         io.print_valid_keywords();
         printTrace_Display("Warning! invalid GYRO-ERROR command.\n");
      }
      else {
         // need to add here page 8 of parse_functions
      }
}
else if ((argument[index].equalsIgnoreCase("DEPTH-CELL-BIAS")) ||
   (argument[index].equalsIgnoreCase("DEPTHCELLBIAS")) ||
   (argument[index].equalsIgnoreCase("DEPTH-CELL-ERROR")) ||
   (argument[index].equalsIgnoreCase("DEPTHCELLERROR")) ||
   (argument[index].equalsIgnoreCase("DEPTH-BIAS")) ||
   (argument[index].equalsIgnoreCase("DEPTHBIAS")) ||
   (argument[index].equalsIgnoreCase("DEPTH-ERROR")) ||
   (argument[index].equalsIgnoreCase("DEPTHERROR"))) {
      index++;
      if (index >= numargs) {
         io.print_valid_keywords();
         printTrace_Display("Warning! invalid DEPTH-CELL-BIAS command.\n");
      }
      else {
         // need to add here page 9 of parse_functions
      }
}
else if ((argument[index].equalsIgnoreCase("BENCH")) ||
   (argument[index].equalsIgnoreCase("BENCH-TEST")) ||
   (argument[index].equalsIgnoreCase("BENCHTEST"))) {
      flags.DISPLAYSCREEN = true;
      flags.LOCATIONLAB = false;
      flags.KEYBOARDINPUT = true;
      flags.DIVETRACKER = false;
      flags.EMAIL = false;
      flags.REALTIME = true;
      flags.BENCHTEST = true;
}
else if ((argument[index].equalsIgnoreCase("NOSCRIPT")) ||
```

```
    (argument[index].equalsIgnoreCase("NO-SCRIPT"))) {
    flags.NOSCRIPT = true;
}
else if ((argument[index].equalsIgnoreCase("MISSION")) ||
    (argument[index].equalsIgnoreCase("SCRIPT")) ||
    (argument[index].equalsIgnoreCase("FILE")) ||
    (argument[index].equalsIgnoreCase("FILENAME"))) {
        index++;
        if (index >= numargs) {
            io.print_valid_keywords();
            printTrace_Display("Warning! invalid command.\n"
                + argument[index]);
        }
        else {
            // ADD TO HERE Page 9 - 10 parse functions
        }
}
else if ((argument[index].equalsIgnoreCase("ST1000SCANWIDTH")) ||
    (argument[index].equalsIgnoreCase("ST1000-SCAN-WIDTH"))) {
        index++;
        if (index >= numargs) {
            io.print_valid_keywords();
        }
        else {
            if (Double.parseDouble(argument[index]) > 0.0) {
                flags.ST1000SCANWIDTH = Double.parseDouble(argument[index]);
                printTrace_Display("[ ST1000SCANWIDTH ] " +
                    flags.ST1000SCANWIDTH);
            }
            else {
                printTrace_Display("Illegal  ST1000SCANWIDTH  value, ignored");
                printTrace_Display("[ ST1000SCANWIDTH ] " +
                    flags.ST1000SCANWIDTH);
            }
        }
}
else if ((argument[index].equalsIgnoreCase("OCEANCURRENT")) ||
    (argument[index].equalsIgnoreCase("OCEAN-CURRENT"))) {
        if (index + 2 <= numargs) {
            control.auv_oceancurrent_x = Double.parseDouble(argument[i + 1]);
            control.auv_oceancurrent_y = Double.parseDouble(argument[i + 2]);
            printTrace_Display("[ OCEANCURRENT ]\n " + control.auv_oceancurrent_x
                + " " + control.auv_oceancurrent_y);
            index += 2;
            control.current_magnitude = Math.sqrt(
                Math.pow(control.auv_oceancurrent_x, 2) +
                Math.pow(control.auv_oceancurrent_y, 2) +
                Math.pow(control.auv_oceancurrent_z, 2));

            if (control.current_magnitude > 1.0) {
                printTrace_Display("\nWarning!!!, ocean current magnitude is " +
                                                        control.current_magnitude);
                printTrace_Display("\nARIES may not be able to maintain position aboce 1.0
knots\n");
            }
        }
```

```
        }
    } // end of for loop
    //                  return flags;
} // end of match_command_line_flag

/**
 *  This method match the command and its parameters to a pre-defined
 *  set of commands.
 *  @param:        String parsed command, String[] parameters parsed
 *  @return:  Boolean
 */
public boolean match_commands(String commandPassed, String[] parametersPassed) {
    String command = commandPassed;
    parameters = parametersPassed;
    // If Shutdown in Progress, Ignore Commands and Go to Shutdown Script
    if (flags.HALTSCRIPT) {
        return (false);
    }
    // If telemetry replay in Progress, no further action required
    if (flags.REPLAY == true) {
        return (false);
    }
    // do not skip to next command in KEYBOARD or script mode until ready  */
    if ((vehicle.get_time() < control.time_next_command) &&
        (flags.TACTICAL == false) && (flags.TACTICALPARSE == false)) {
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("\n[skip parse_mission_script_commands () until ");
            System.out.println("t > time_next_command]\n");
        }
        return (false);
    }
    if ((flags.GPSFIXINPROGRESS) && (vehicle.get_time() >= control.time_postgps_dive)) {
        if (flags.TACTICAL) {
            if (flags.TRACE && flags.DISPLAYSCREEN)
                System.out.println("write_to_tactical_socket ( STABLE GPS TIMEOUT )]");
            network.write_to_tactical("STABLE GPS TIMEOUT");
        }
        flags.GPSFIXINPROGRESS = false;
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("\n[Start match_mission_script_commands()]\n");
    }
    if ((flags.GPSFIXINPROGRESS) && (vehicle.get_time() >= time_gps_complete) &&
        (vehicle.get_time() < control.time_postgps_dive)) {
            control.z_command = control.previous_z_command;
            control.time_postgps_dive = vehicle.get_time() + 30.0; // head back to ordered depth
            control.time_next_command = control.time_postgps_dive;
            control.time_gps_complete = control.time_postgps_dive + 1.0;
            if (flags.DISPLAYSCREEN) System.out.println("\n[GPS-FIX complete.]\n");
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("Command read  = " + command);
        System.out.println("Parameters read  = " + command);
        for (int i = 1; i < parameters.length; i++) {
            System.out.println("Parameter [" + i + "] = " + parameters[i]);
        }
```

```
}
// ************************************
// ********* Match block ************
// ************************************
if (flags.KEYBOARDINPUT) {
   System.out.println("*** Enter command here ***: ");
   try {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      command = br.readLine();
      network.write_Telemetry_to_dynamics( inputCommand );
      if (flags.TRACE) {
         System.out.println("Command entered: " + command);
      }
   }
   catch (IOException e) {
      System.out.println("IOException " + e + " while reading command from user ");
   }
}
if (flags.TACTICAL) {
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("RECEIVE TACTICAL COMMAND *** HERE ***\n");
   }
   command = network.read_from_tactical();
   if (command.length() == 0) { // no tactical message received
      control.time_next_command = vehicle.get_time() + control.dt;
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("no tactical command received, STEP & recheck\n");
      }
   }
}
if ((command.equalsIgnoreCase("HELP")) ||
   (command.equalsIgnoreCase("?")) ||
   (command.equalsIgnoreCase("-?")) ||
   (command.equalsIgnoreCase("/?"))) {
      if (flags.DISPLAYSCREEN) System.out.println("\n[HELP] ");
      io.print_valid_keywords();
}
else if ((command.equalsIgnoreCase("FOLLOWLIGHT"))) {
   if (flags.DISPLAYSCREEN) System.out.println("\n\n[Light-following starts]\n");
   if (flags.DISPLAYSCREEN) System.out.println("\n\n[Look-Around starts]\n");
   flags.FOLLOWLIGHTCONTROL = true;
   flags.HOVERCONTROL = false;
}
else if ((command.equalsIgnoreCase("SONAR_725")) ||
   (command.equalsIgnoreCase("SONAR725")) ||
   (command.equalsIgnoreCase("SONAR-725")) ||
   (command.equalsIgnoreCase("ST_725")) ||
   (command.equalsIgnoreCase("ST725")) ||
   (command.equalsIgnoreCase("ST-725"))) {
      if (parameters.length == 5) {
         if (flags.DISPLAYSCREEN && flags.LOCATIONLAB) {
            System.out.println("\n[ keyword = " + command +
               " bearing = " + parameters[1] + " range = " +
               parameters[2] + " power = " + parameters[3] +
               " direction = " + parameters[4] + " ]\n");
         }
```

```
    vehicle.set_auv_ST725_bearing(data_Processing.normalize(
        Double.parseDouble((parameters[1])))); // controlled by AUV

    if (flags.LOCATIONLAB == false) {
        // only virtual world provides sonar values when out of water */
        vehicle.set_auv_ST725_range(Double.parseDouble(parameters[2]));
        vehicle.set_auv_ST725_strength(Double.parseDouble(parameters[3]));
    }
    if ((parameters[4].equalsIgnoreCase("TRUE")) ||
        (parameters[4].equalsIgnoreCase("T"))) {
        vehicle.set_auv_ST725_direction(TRUE); // TRUE DIRECTION
    }
    else if ((parameters[4].equalsIgnoreCase("RELATIVE")) ||
        (parameters[4].equalsIgnoreCase("REL")) ||
        (parameters[4].equalsIgnoreCase("R"))) {
            vehicle.set_auv_ST725_direction(RELATIVE); // RELATIVE
    }
    else if (flags.DISPLAYSCREEN && flags.LOCATIONLAB) {
        System.out.println("[ " + parameters[4] +
            " is illegal direction (not TRUE/RELATIVE), assume RELATIVE]\n");
        vehicle.set_auv_ST725_direction(RELATIVE); // RELATIVE
    }
} // end of if parameters.length == 5

else if (parameters.length == 4) {
    if (flags.DISPLAYSCREEN && flags.LOCATIONLAB) {
        System.out.println("\n[ keyword = " + command +
            " bearing = " + parameters[1] + " range = " +
            parameters[2] + " power = " + parameters[3] + " ]\n");
    }
    vehicle.set_auv_ST725_bearing(data_Processing.normalize(
        Double.parseDouble(parameters[1]))); // controlled by AUV

    if (flags.LOCATIONLAB == false) {
        // only virtual world provides sonar values when out of water
        vehicle.set_auv_ST725_range(Double.parseDouble(parameters[2]));
        vehicle.set_auv_ST725_strength(Double.parseDouble(parameters[3]));
    }
} // end of if parameters.length == 4

else if (parameters.length == 3) {
    if (flags.DISPLAYSCREEN && flags.LOCATIONLAB) {
        System.out.println("\n[ keyword = " + command +
            " bearing = " + parameters[1] + " range = " +
            parameters[2] + " ]\n");
        if (vehicle.get_auv_ST725_direction() == TRUE) {
            System.out.println(" direction = TRUE]\n");
        }
        else {
            System.out.println(" direction = RELATIVE]\n");
        }
    }
    vehicle.set_auv_ST725_bearing(data_Processing.normalize(
        Double.parseDouble(parameters[1]))); // controlled by AUV

    if (flags.LOCATIONLAB == false) {
```

```
            // only virtual world provides sonar values when out of water
            vehicle.set_auv_ST725_range(Double.parseDouble(parameters[2]));
         }
      } // end of if parameters.length == 3

      else if (parameters.length == 2) {
         if (flags.DISPLAYSCREEN && flags.LOCATIONLAB) {
            System.out.println("\n[ keyword = " + command +
               " bearing = " + parameters[1] + " ]\n");
            if (vehicle.get_auv_ST725_direction() == TRUE) {
               System.out.println(" direction = TRUE]\n");
            }
            else {
               System.out.println(" direction = RELATIVE]\n");
            }
         }
         vehicle.set_auv_ST725_bearing(data_Processing.normalize(
            Double.parseDouble(parameters[1]))); // controlled by AUV

      } // end of if parameters.length == 2

      else if (flags.DISPLAYSCREEN)
         System.out.println("Warning! invalid SONAR_725 command, ignored\n");
   } // end of (( command.equalsIgnoreCase ( "SONAR_725" )) block

   else if ((command.equalsIgnoreCase("SONAR_1000")) ||
      (command.equalsIgnoreCase("SONAR1000")) ||
      (command.equalsIgnoreCase("SONAR-1000")) ||
      (command.equalsIgnoreCase("ST_1000")) ||
      (command.equalsIgnoreCase("ST1000")) ||
      (command.equalsIgnoreCase("ST-1000"))) {

      if (parameters.length == 3) {
         if (flags.DISPLAYSCREEN && flags.LOCATIONLAB) {
            System.out.println("\n[ keyword = " + command +
               " bearing = " + parameters[1] + " direction = " +
               parameters[2] + " ]\n");
         }
         if ((parameters[2].equalsIgnoreCase("TRUE")) ||
            (parameters[2].equalsIgnoreCase("T"))) {
            vehicle.set_auv_ST1000_direction(TRUE); // TRUE
         }
         else if ((parameters[4].equalsIgnoreCase("RELATIVE")) ||
            (parameters[4].equalsIgnoreCase("REL")) ||
            (parameters[4].equalsIgnoreCase("R"))) {
               vehicle.set_auv_ST725_direction(RELATIVE); // RELATIVE
         }
         else if (flags.DISPLAYSCREEN && flags.LOCATIONLAB) {
            System.out.println("[ " + parameters[4] +
               " is illegal direction (not TRUE/RELATIVE), assume RELATIVE]\n");
            vehicle.set_auv_ST725_direction(RELATIVE); // RELATIVE
         }
         flags.ST1000SCANMODE = flags.SONARSCANMANUAL; // manual control
         control.ST1000_command = data_Processing.normalize(
            Double.parseDouble(parameters[1]));
```

```
                } // end of ( parameters.length == 3 )

                else if (parameters.length == 2) {
                  if (flags.DISPLAYSCREEN && flags.LOCATIONLAB) {
                    System.out.println("\n[ keyword = " + command +
                      " bearing = " + parameters[1] + " ]\n");
                    if (vehicle.get_auv_ST1000_direction() == TRUE) {
                      System.out.println(" direction = TRUE]\n");
                    }
                    else {
                      System.out.println(" direction = RELATIVE]\n");
                    }
                  }
                  flags.ST1000SCANMODE = flags.SONARSCANMANUAL; // manual control
                  control.ST1000_command = data_Processing.normalize(
                    Double.parseDouble(parameters[1]));
                }
                else {
                  if (flags.DISPLAYSCREEN && flags.LOCATIONLAB)
                    System.out.println("Warning! invalid SONAR_1000 command," +
                    " ignored\n" + command);
                } //  end of ( parameters.length == 2 )

            } // end of (( command.equalsIgnoreCase ( "SONAR_1000" )) block

            else if ((command.equalsIgnoreCase("POSITION")) || (command.equalsIgnoreCase("LOCATION")) ||
              (command.equalsIgnoreCase("FIX"))) {
                // note this command must be sent to virtual world (AUVsocket.C tests)
                if (parameters.length == 4) {
                  vehicle.set_x(Double.parseDouble(parameters[1]));
                  vehicle.set_y(Double.parseDouble(parameters[2]));
                  vehicle.set_z(Double.parseDouble(parameters[3])); // note depth cell will likely update z
                  // What is this ==>// skip line in telemetry file to break point-to-point lines
                  // What is this ==>// if (( flags.TACTICALPARSE ) || ( flags.TACTICAL == false ))
                  // What is this ==>// fprintf (auvdatafile, "\n");
                  if (flags.DISPLAYSCREEN)
                    System.out.println("\n" + command + "," +
                    parameters[1] + ", " + parameters[2] + ", " +
                    parameters[3]);
                  if (flags.TRACE && flags.DISPLAYSCREEN)
                    System.out.println("\nsending fix to virtual world:  ");
                  //***************************
                  // SENDING TO VW NEEDS TO BE MATCHED
                  //*******************************

                  String commandLine = command + " " + parameters[1] + " " +
                    parameters[2] + " " + parameters[3] + "\n";
                  network.write_Telemetry_to_dynamics(commandLine);
                }
                else if (parameters.length == 3) {
                  vehicle.set_x(Double.parseDouble(parameters[1]));
                  vehicle.set_y(Double.parseDouble(parameters[2]));
                  // skip line in telemetry file to break point-to-point lines */
                  //   if ((TACTICALPARSE) || (TACTICAL == FALSE))
```

```
        //      fprintf (auvdatafile, "\n");
        if (flags.DISPLAYSCREEN)
           System.out.println("\n" + command + ", " +
           parameters[1] + ", " + parameters[2]);
        if (flags.TRACE && flags.DISPLAYSCREEN)
           System.out.println("\nsending fix to virtual world: \n");
        //       strcpy (buffer, command_buffer); /* copy command to buffer*/
        //      send_buffer_to_virtual_world_socket (); /* send to vw     */
        String commandLine = command + " " + parameters[1] + " " +
           parameters[2] + "\n";
        network.write_Telemetry_to_dynamics(commandLine);
     }
     else if (flags.DISPLAYSCREEN)
        System.out.println("Warning! invalid x/y/z fix position, ignored\n");
} // end of if (( command.equalsIgnoreCase ( "POSITION" ))...

else if ((command.equalsIgnoreCase("HOVER")) ||
   (command.equalsIgnoreCase("HOVER-ON"))) {
   if (parameters.length == 6) {
      if (flags.DISPLAYSCREEN) {
         System.out.println("\n[ " + command + ", " + parameters[1] +
            ", " + parameters[2] + ", " + parameters[3] +
            ", " + parameters[4] + ", " + parameters[5] + " ]\n");
      }
      flags.HOVERCONTROL = true;
      flags.REPORTSTABLE = true;
      flags.WAYPOINTCONTROL = false;
      flags.ROTATECONTROL = false;
      flags.LATERALCONTROL = false;
      flags.THRUSTERCONTROL = true;
      flags.TARGETCONTROL = false;
      flags.RECOVERYCONTROL = false;
      flags.INTEGRALDEPTHCONTROL = FALSE;
      flags.time_int_control_on = vehicle.get_time() + 10.0; // give PD control 10 seconds
      flags.ST725SCANMODE = flags.SONARSCANSWATH; // Forward Scan
      flags.DEADSTICKRUDDER = true;
      flags.DEATH_SPIRAL_RESET = true;
      control.rudder_command = 0.0;
      control.x_command = Double.parseDouble(parameters[1]);
      control.y_command = Double.parseDouble(parameters[2]);
      control.z_command = Double.parseDouble(parameters[3]);
      control.psi_command = data_Processing.normalize(
         Double.parseDouble(parameters[4]));
      control.psi_command_hover = control.psi_command;
      control_coefficients.set_standoff_distance(
         Double.parseDouble(parameters[5]));
   }
   else if (parameters.length == 5) {
      if (flags.DISPLAYSCREEN) {
         System.out.println("\n[ " + command + ", " + parameters[1] +
            ", " + parameters[2] + ", " + parameters[3] +
            ", " + parameters[4] + " ]\n");
      }
      System.out.println("\n[ " + command + ", " + parameters[1] +
         ", " + parameters[2] + ", " + parameters[3] +
         ", " + parameters[4] + " ]\n");
```

```
    flags.HOVERCONTROL = true;
    flags.REPORTSTABLE = true;
    flags.WAYPOINTCONTROL = false;
    flags.ROTATECONTROL = false;
    flags.LATERALCONTROL = false;
    flags.THRUSTERCONTROL = true;
    flags.TARGETCONTROL = false;
    flags.RECOVERYCONTROL = false;
    flags.INTEGRALDEPTHCONTROL = FALSE;
    flags.time_int_control_on = vehicle.get_time() + 10.0; // give PD control 10 seconds
    flags.ST725SCANMODE = flags.SONARSCANSWATH; // Forward Scan
    flags.DEADSTICKRUDDER = true;
    flags.DEATH_SPIRAL_RESET = true;
    control.rudder_command = 0.0;
    control.x_command = Double.parseDouble(parameters[1]);
    control.y_command = Double.parseDouble(parameters[2]);
    control.z_command = Double.parseDouble(parameters[3]);
    control.psi_command = data_Processing.normalize(
        Double.parseDouble(parameters[4]));
    control.psi_command_hover = control.psi_command;
}
else if (parameters.length == 4) {
    if (flags.DISPLAYSCREEN) {
        System.out.println("\n[ " + command + ", " + parameters[1] +
            ", " + parameters[2] + ", " +
            parameters[3] + " ]\n");
    }
    flags.HOVERCONTROL = true;
    flags.REPORTSTABLE = true;
    flags.WAYPOINTCONTROL = false;
    flags.ROTATECONTROL = false;
    flags.LATERALCONTROL = false;
    flags.THRUSTERCONTROL = true;
    flags.TARGETCONTROL = false;
    flags.RECOVERYCONTROL = false;
    flags.INTEGRALDEPTHCONTROL = FALSE;
    flags.time_int_control_on = vehicle.get_time() + 10.0; // give PD control 10 seconds
    flags.ST725SCANMODE = flags.SONARSCANSWATH; // Forward Scan
    flags.DEADSTICKRUDDER = true;
    flags.DEATH_SPIRAL_RESET = true;
    control.rudder_command = 0.0;
    control.x_command = Double.parseDouble(parameters[1]);
    control.y_command = Double.parseDouble(parameters[2]);
    control.z_command = Double.parseDouble(parameters[3]);
    control.psi_command_hover = control.psi_command;
}
else if (parameters.length == 3) {
    if (flags.DISPLAYSCREEN) {
        System.out.println("\n[ " + command + ", " + parameters[1] +
            ", " + parameters[2] + " ]\n");
    }
    flags.HOVERCONTROL = true;
    flags.REPORTSTABLE = true;
    flags.WAYPOINTCONTROL = false;
    flags.ROTATECONTROL = false;
    flags.LATERALCONTROL = false;
```

```
        flags.THRUSTERCONTROL = true;
        flags.TARGETCONTROL = false;
        flags.RECOVERYCONTROL = false;
        flags.INTEGRALDEPTHCONTROL = FALSE;
        flags.time_int_control_on = vehicle.get_time() + 10.0; // give PD control 10 seconds
        flags.ST725SCANMODE = flags.SONARSCANSWATH; // Forward Scan
        flags.DEADSTICKRUDDER = true;
        flags.DEATH_SPIRAL_RESET = true;
        control.rudder_command = 0.0;
        control.x_command = Double.parseDouble(parameters[1]);
        control.y_command = Double.parseDouble(parameters[2]);
        control.psi_command_hover = control.psi_command;
    }
    else if (parameters.length == 2) {
        if (flags.DISPLAYSCREEN) {
            System.out.println("\n" + command);
        }
        flags.HOVERCONTROL = true;
        flags.REPORTSTABLE = true;
        flags.WAYPOINTCONTROL = false;
        flags.ROTATECONTROL = false;
        flags.LATERALCONTROL = false;
        flags.THRUSTERCONTROL = true;
        flags.TARGETCONTROL = false;
        flags.RECOVERYCONTROL = false;
        flags.INTEGRALDEPTHCONTROL = FALSE;
        flags.time_int_control_on = vehicle.get_time() + 10.0; // give PD control 10 seconds
        flags.ST725SCANMODE = flags.SONARSCANSWATH; // Forward Scan
        flags.DEADSTICKRUDDER = true;
        flags.DEATH_SPIRAL_RESET = true;
        control.rudder_command = 0.0;
        control.x_command = vehicle.get_x(); // stay put!
        control.y_command = vehicle.get_y();
        control.z_command = vehicle.get_z();
        control.psi_command_hover = vehicle.get_psi(); // "Meet Her"
    }
    else {
        if (flags.DISPLAYSCREEN) {
            System.out.println("\n" + command);
            System.out.println("Warning! improper number of values, ignored\n");
        }
    }
} // end of if (( command.equalsIgnoreCase ( "HOVER" )) block

else if ((command.equalsIgnoreCase("GPS")) ||
    (command.equalsIgnoreCase("GPSFIX")) ||
    (command.equalsIgnoreCase("GPS-FIX"))) {
        if (flags.TACTICALPARSE == false) {
            control.previous_z_command = control.z_command;
            if (control.z_command > 40.0) {
                control.z_command = 41.0; // deep test tank
            }
            else {
                control.z_command = 0.0; // rapid shallow
            }
            flags.GPSFIXINPROGRESS = true;
```

```
            control.time_gps_complete = vehicle.get_time() + 30.0;
            control.time_postgps_dive = vehicle.get_time() + 60.0;
            control.time_next_command = time_gps_complete;
            // fixed guestimate of GPS fix interval
            // assume GPS-FIX behavior is properly controlled by tactical level
         }
         if (flags.DISPLAYSCREEN) System.out.println("\n[GPS-FIX]\n");
      } // end of if (( command.equalsIgnoreCase ( "GPS" ))

      else if ((flags.GPSFIXINPROGRESS) &&
         ((command.equalsIgnoreCase("GPS-COMPLETE")) ||
         (command.equalsIgnoreCase("GPS-FIX-COMPLETE")) ||
         (command.equalsIgnoreCase("GPSCOMPLETE")) ||
         (command.equalsIgnoreCase("GPSFIXCOMPLETE")))) {
         control.z_command = control.previous_z_command;
         control.time_postgps_dive = vehicle.get_time() + 30.0; // head back to ordered depth
         control.time_next_command = control.time_postgps_dive;
         control.time_gps_complete = control.time_postgps_dive + 1.0;
         //          read_another_line = false;
         if (flags.DISPLAYSCREEN) {
            System.out.println("\n[GPS-FIX complete.]\n");
         }
      } // end of else if (( flags.GPSFIXINPROGRESS ) && ...

      else if ((command.equalsIgnoreCase("TARGET-STATION")) ||
         (command.equalsIgnoreCase("TARGETSTATION"))) {
         if (parameters.length == 3) {
            if (flags.DISPLAYSCREEN) {
               System.out.println("\n[ " + command + ", " + parameters[1] +
                  ", " + parameters[2] + " ]\n");
            }
            flags.TARGETCONTROL = true;
            flags.NEWTARGET = true;
            flags.RECOVERYCONTROL = false;
            flags.ST1000SCANMODE = flags.SONARSCANLOCATE; // Locate Target with
ST1000
            flags.TARGETPOINTING = true;
            flags.TARGETEDGETRACK = false;
            flags.HOVERCONTROL = false;
            flags.WAYPOINTCONTROL = false;
            flags.FOLLOWWAYPOINTMODE = false;
            flags.LATERALCONTROL = false;
            flags.ROTATECONTROL = false;
            flags.REPORTSTABLE = true;
            flags.NEWTARGETSTATION = true;
            control.target_range_command = Double.parseDouble(parameters[1]);
            control.target_bearing_command = data_Processing.normalize(
               Double.parseDouble(parameters[2]));
         }
         else if (parameters.length == 4) {
            if (flags.DISPLAYSCREEN) {
               System.out.println("\n[ " + command + ", " + parameters[1] +
                  ", " + parameters[2] + ", " +
                  parameters[3] + " ]\n");
            }
            flags.TARGETCONTROL = true;
```

```
                flags.NEWTARGET = true;
                flags.RECOVERYCONTROL = false;
                flags.ST1000SCANMODE = flags.SONARSCANLOCATE; // Locate Target with
ST1000
                flags.TARGETPOINTING = false;
                flags.TARGETEDGETRACK = false;
                flags.HOVERCONTROL = false;
                flags.WAYPOINTCONTROL = false;
                flags.FOLLOWWAYPOINTMODE = false;
                flags.LATERALCONTROL = false;
                flags.ROTATECONTROL = false;
                flags.REPORTSTABLE = true;
                flags.NEWTARGETSTATION = true;
                control.target_range_command = Double.parseDouble(parameters[1]);
                control.target_bearing_command = data_Processing.normalize(
                   Double.parseDouble(parameters[2]));
                control.psi_command = data_Processing.normalize(
                   Double.parseDouble(parameters[3]));
             }
             else if (parameters.length == 5) {
                if (flags.DISPLAYSCREEN) {
                   System.out.println("\n[ " + command + ", " + parameters[1] +
                      ", " + parameters[2] + ", " + parameters[3] +
                      ", " + parameters[4] + " ]\n");
                }
                flags.TARGETCONTROL = true;
                flags.NEWTARGET = true;
                flags.RECOVERYCONTROL = false;
                flags.ST1000SCANMODE = flags.SONARSCANLOCATE; // Locate Target with
ST1000
                flags.TARGETPOINTING = true;
                flags.TARGETEDGETRACK = false;
                flags.HOVERCONTROL = false;
                flags.WAYPOINTCONTROL = false;
                flags.FOLLOWWAYPOINTMODE = false;
                flags.LATERALCONTROL = false;
                flags.ROTATECONTROL = false;
                flags.REPORTSTABLE = true;
                flags.NEWTARGETSTATION = true;
                control.target_range_command = Double.parseDouble(parameters[3]);
                control.target_bearing_command = data_Processing.normalize(
                   Double.parseDouble(parameters[4]));
                control.target_range = Double.parseDouble(parameters[1]);
                control.target_bearing = data_Processing.normalize(
                   Double.parseDouble(parameters[2]));
                //*****************************************************************
                // Casting to Integer.  In the C code the return of dsign and
                // normalize is double.
                // ST1000SCANDIRECTION is defined in the C code as Integer 1 for
                // RIGHT, -1 for LEFT.
                // It must has shown a warning.
                //*****************************************************************
                flags.ST1000SCANDIRECTION = (int)data_Processing.dsign(
                   data_Processing.normalize2(control.target_bearing -
                   data_Processing.normalize(vehicle.get_psi() -
                   vehicle.get_auv_ST1000_bearing())));
```

```
            }
            else if (parameters.length == 6) {
                if (flags.DISPLAYSCREEN) {
                    System.out.println("\n[ " + command + ", " + parameters[1] +
                        ", " + parameters[2] + ", " + parameters[3] +
                        ", " + parameters[4] + ", " + parameters[5] + " ]\n");
                }
                flags.TARGETCONTROL = true;
                flags.NEWTARGET = true;
                flags.RECOVERYCONTROL = false;
                flags.ST1000SCANMODE = flags.SONARSCANLOCATE; // Locate Target with
ST1000
                //**************************
                //
                //          flags.TARGETPOINTING changes it flag from parameters read to
                //  parameters read.  IS THAT RIGHT?
                //
                //**************************
                flags.TARGETPOINTING = false;
                flags.TARGETEDGETRACK = false;
                flags.HOVERCONTROL = false;
                flags.WAYPOINTCONTROL = false;
                flags.FOLLOWWAYPOINTMODE = false;
                flags.LATERALCONTROL = false;
                flags.ROTATECONTROL = false;
                flags.REPORTSTABLE = true;
                flags.NEWTARGETSTATION = true;
                control.target_range_command = Double.parseDouble(parameters[3]);
                control.target_bearing_command = data_Processing.normalize(
                    Double.parseDouble(parameters[4]));
                control.psi_command = data_Processing.normalize(
                    Double.parseDouble(parameters[5]));
                control.target_range = Double.parseDouble(parameters[1]);
                control.target_bearing = data_Processing.normalize(
                    Double.parseDouble(parameters[2]));
                //*****************************************************************
                // Casting to Integer.  In the C code the return of dsign
                // and normalize is double.
                // ST1000SCANDIRECTION is defined in the C code as Integer 1 for
                // RIGHT, -1 for LEFT.
                // It must has shown a warning.
                //*****************************************************************
                flags.ST1000SCANDIRECTION = (int)data_Processing.dsign(
                    data_Processing.normalize2(control.target_bearing -
                    data_Processing.normalize(vehicle.get_psi() -
                    vehicle.get_auv_ST1000_bearing())));
            }
        } // end of else if (( command.equalsIgnoreCase( "TARGET-STATION" )) block

        else if ((command.equalsIgnoreCase("EDGE-STATION")) ||
            (command.equalsIgnoreCase("EDGESTATION"))) {
            if (parameters.length == 3) {
                if (flags.DISPLAYSCREEN) {
                    System.out.println("\n[ " + command + ", " + parameters[1] +
                        ", " + parameters[2] + " ]\n");
                }
```

```
                flags.TARGETCONTROL = true;
                flags.NEWTARGET = true;
                flags.RECOVERYCONTROL = false;
                flags.ST1000SCANMODE = flags.SONARSCANLOCATE; // Locate Target with
ST1000
                flags.TARGETPOINTING = true;
                flags.TARGETEDGETRACK = true;
                flags.HOVERCONTROL = false;
                flags.WAYPOINTCONTROL = false;
                flags.FOLLOWWAYPOINTMODE = false;
                flags.LATERALCONTROL = false;
                flags.ROTATECONTROL = false;
                flags.REPORTSTABLE = true;
                flags.NEWTARGETSTATION = true;
                control.target_range_command = Double.parseDouble(parameters[1]);
                control.target_bearing_command = data_Processing.normalize(
                    Double.parseDouble(parameters[2]));
            }
            else if (parameters.length == 4) {
                if (flags.DISPLAYSCREEN) {
                    System.out.println("\n[ " + command + ", " + parameters[1] +
                        ", " + parameters[2] + ", " +
                        parameters[3] + " ]\n");
                }
                flags.TARGETCONTROL = true;
                flags.NEWTARGET = true;
                flags.RECOVERYCONTROL = false;
                flags.ST1000SCANMODE = flags.SONARSCANLOCATE; // Locate Target with
ST1000
                flags.TARGETPOINTING = false;
                flags.TARGETEDGETRACK = true;
                flags.HOVERCONTROL = false;
                flags.WAYPOINTCONTROL = false;
                flags.FOLLOWWAYPOINTMODE = false;
                flags.LATERALCONTROL = false;
                flags.ROTATECONTROL = false;
                flags.REPORTSTABLE = true;
                flags.NEWTARGETSTATION = true;
                control.target_range_command = Double.parseDouble(parameters[1]);
                control.target_bearing_command = data_Processing.normalize(
                    Double.parseDouble(parameters[2]));
                control.psi_command = data_Processing.normalize(
                    Double.parseDouble(parameters[3]));
            }
            else if (parameters.length == 5) {
                if (flags.DISPLAYSCREEN) {
                    System.out.println("\n[ " + command + ", " + parameters[1] +
                        ", " + parameters[2] + ", " + parameters[3] +
                        ", " + parameters[4] + " ]\n");
                }
                flags.TARGETCONTROL = true;
                flags.NEWTARGET = true;
                flags.RECOVERYCONTROL = false;
                flags.ST1000SCANMODE = flags.SONARSCANLOCATE;

                /* Locate Target with ST1000 */
```

```
                flags.TARGETPOINTING = true;
                flags.TARGETEDGETRACK = true;
                flags.HOVERCONTROL = true;
                flags.WAYPOINTCONTROL = false;
                flags.FOLLOWWAYPOINTMODE = false;
                flags.LATERALCONTROL = false;
                flags.ROTATECONTROL = false;
                flags.REPORTSTABLE = true;
                flags.NEWTARGETSTATION = true;
                control.target_range_command = Double.parseDouble(parameters[3]);
                control.target_bearing_command = data_Processing.normalize(
                   Double.parseDouble(parameters[4]));
                control.target_range = Double.parseDouble(parameters[1]);
                control.target_bearing = data_Processing.normalize(
                   Double.parseDouble(parameters[2]));
                flags.ST1000SCANDIRECTION = (int)data_Processing.dsign(
                   data_Processing.normalize2(control.target_bearing -
                   data_Processing.normalize(vehicle.get_psi() -
                   vehicle.get_auv_ST1000_bearing())));
             }
             else if (parameters.length == 6) {
                if (flags.DISPLAYSCREEN) {
                   System.out.println("\n[ " + command + ", " + parameters[1] +
                      ", " + parameters[2] + ", " + parameters[3] +
                      ", " + parameters[4] + ", " + parameters[5] + " ]\n");
                }
                flags.TARGETCONTROL = true;
                flags.NEWTARGET = true;
                flags.RECOVERYCONTROL = false;
                flags.ST1000SCANMODE = flags.SONARSCANLOCATE; // Locate Target with
ST1000
                flags.TARGETPOINTING = false;
                flags.TARGETEDGETRACK = true;
                flags.HOVERCONTROL = false;
                flags.WAYPOINTCONTROL = false;
                flags.FOLLOWWAYPOINTMODE = false;
                flags.LATERALCONTROL = false;
                flags.ROTATECONTROL = false;
                flags.REPORTSTABLE = true;
                flags.NEWTARGETSTATION = true;
                control.target_range_command = Double.parseDouble(parameters[3]);
                control.target_bearing_command =
data_Processing.normalize(Double.parseDouble(parameters[4]));
                control.psi_command =
data_Processing.normalize(Double.parseDouble(parameters[5]));
                control.target_range = Double.parseDouble(parameters[1]);
                control.target_bearing =
data_Processing.normalize(Double.parseDouble(parameters[2]));
                flags.ST1000SCANDIRECTION = (int)data_Processing.dsign(
                   data_Processing.normalize2(control.target_bearing -
                   data_Processing.normalize(vehicle.get_psi() -
                   vehicle.get_auv_ST1000_bearing())));
             }
          } // end of else if (( command.equalsIgnoreCase( "EDGE-STATION" )) block
```

```
else if ((command.equalsIgnoreCase("ENTER-TUBE")) || (
   command.equalsIgnoreCase("ENTERTUBE"))) {
   System.out.println("\n[ " + command + ", " + parameters[1] + ", " +
      parameters[2] + " ]\n");
   flags.ST1000SCANMODE = flags.SONARSCANMANUAL;
   flags.ST1000_command = data_Processing.normalize(-75.0);
   flags.HOVERCONTROL = false;
   flags.WAYPOINTCONTROL = false;
   flags.LATERALCONTROL = false;
   flags.ROTATECONTROL = false;
   flags.TARGETCONTROL = false;
   flags.RECOVERYCONTROL = true;
   flags.REPORTSTABLE = true;
   flags.NEWRECOVERYCOMMAND = true;
   control.range_from_recovery_pt = Double.parseDouble(parameters[1]);
   control.psi_command = data_Processing.normalize(
      Double.parseDouble(parameters[2]));
   control.psi_command_hover = control.psi_command;
   control.x_command = vehicle.get_x();
   control.y_command = vehicle.get_y();
   //  z_command         = z;  not needed, use previously ordered value
} // end of else if (( command.equalsIgnoreCase( "ENTER-TUBE" )) block
else if ((command.equalsIgnoreCase("WAIT")) ||
   (command.equalsIgnoreCase("RUN"))) {

   if (flags.DISPLAYSCREEN) {
      System.out.println("\n[ " + command + ", " + parameters[1] + " ]");
   }
   if ((parameters.length == 2) && (Double.parseDouble(parameters[1]) >= 0.0)) {
      if (flags.TACTICALPARSE) {
         return (false);
      }
      control.time_next_command = vehicle.get_time() +
         Double.parseDouble(parameters[1]);
      if (flags.DISPLAYSCREEN) System.out.println("time of next command = " +
         control.time_next_command + "\n");
   }
   else if (flags.DISPLAYSCREEN) {
      System.out.println("Warning! illegal time value, ignored\n");
   }
} // end of else if (( command.equalsIgnoreCase( "WAIT" )) block

else if ((command.equalsIgnoreCase("WAITUNTILTIME")) ||
   (command.equalsIgnoreCase("WAIT-UNTIL-TIME")) ||
   (command.equalsIgnoreCase("WAIT_UNTIL_TIME")) ||
   (command.equalsIgnoreCase("WAITUNTIL")) ||
   (command.equalsIgnoreCase("WAIT-UNTIL")) ||
   (command.equalsIgnoreCase("WAIT_UNTIL")) ||
   (command.equalsIgnoreCase("PAUSEUNTIL")) ||
   (command.equalsIgnoreCase("PAUSE-UNTIL")) ||
   (command.equalsIgnoreCase("PAUSE_UNTIL")) ||
   (command.equalsIgnoreCase("TIME"))) {

      if (flags.DISPLAYSCREEN) {
         System.out.println("\n[ " + command + ", " + parameters[1] +
            " ]\n");
```

```
        }
        if (parameters.length == 2) {
            if (flags.TACTICALPARSE) {
                return (false);
            }
            control.time_next_command = Double.parseDouble(parameters[1]);
            if (Double.parseDouble(parameters[1]) <= vehicle.get_time()) {
                vehicle.set_time(Double.parseDouble(parameters[1]));
                if (flags.DISPLAYSCREEN) {
                    System.out.println("Warning! time value in past has reset AUV clock,\n");
                    System.out.println("  velocities reset to zero.\n");
                }
                vehicle.set_u(0.0);
                vehicle.set_v(0.0);
                vehicle.set_w(0.0);
                vehicle.set_p(0.0);
                vehicle.set_q(0.0);
                vehicle.set_r(0.0);
                vehicle.set_x_dot(0.0);
                vehicle.set_y_dot(0.0);
                vehicle.set_z_dot(0.0);
                vehicle.set_phi_dot(0.0);
                vehicle.set_theta_dot(0.0);
                vehicle.set_psi_dot(0.0);
                //read_another_line = TRUE;  // no PDU
            }
        }
        else if (flags.DISPLAYSCREEN) {
            System.out.println("Warning!  illegal time value, ignored.\n");
        }
} // end of else if (( command.equalsIgnoreCase( "WAITUNTILTIME" )) block

// VERIFY WITH C CODE. NOT SURE HERE TO TRANSLATION
else if ((command.equalsIgnoreCase("TIMESTEP")) ||
    (command.equalsIgnoreCase("TIME-STEP"))) { // different than STEP
    if ((parameters.length) == 2) {
        if (flags.TACTICALPARSE) {
            return (false);
        }
        if ((Double.parseDouble(parameters[1]) > 0.0) &&
            (Double.parseDouble(parameters[1]) <= 5.0)) {
            control.dt = Double.parseDouble(parameters[1]);
            //   if (DISPLAYSCREEN)
            //        printf ("\n[TIMESTEP   %6.2f] ", dt);
            //   if (TACTICALPARSE == FALSE)
            //   if (NOSCRIPT == FALSE) fprintf (auvordersfile,
            //      "# timestep:  %4.2f seconds\n", dt);
        }
        else
            io.print_valid_keywords();
    }
    else
        io.print_valid_keywords();
} // end of else if (( command.equalsIgnoreCase( "TIMESTEP" )) block
else if ((command.equalsIgnoreCase("PAUSE")) ||
    (command.equalsIgnoreCase("-PAUSE"))) {
```

```
        if (flags.DISPLAYSCREEN) {
          System.out.println("\n[PAUSE]\n");
          //            strcpy (buffer, " Press any key to continue");
          //            send_buffer_to_virtual_world_socket (); // buffer msg sent
          //            printf ("\n%s *** HERE ***:  ", buffer);
          //            answer = getchar (); /* pause */
          System.out.println("*** Press any key to continue ***: ");
          try {
            BufferedReader br = new BufferedReader(
              new InputStreamReader(System.in));
            command = br.readLine();
            if (flags.TRACE) {
              System.out.println("Command entered: " + command);
            }
          }
          catch (IOException e) {
            System.out.println("IOException " + e +
              " while reading command from user ");
          }
        }
      } // end of else if (( command.equalsIgnoreCase(  "PAUSE" ))

      else if ((command.equalsIgnoreCase("REALTIME")) ||
        (command.equalsIgnoreCase("REAL-TIME"))) {
        if (flags.DISPLAYSCREEN) System.out.println("\n[REALTIME] ");
        flags.REALTIME = true;
      } // end of else if (( command.equalsIgnoreCase( "REALTIME" ))

      else if ((command.equalsIgnoreCase("MISSION")) ||
        (command.equalsIgnoreCase("SCRIPT")) ||
        (command.equalsIgnoreCase("FILE")) ||
        (command.equalsIgnoreCase("FILENAME "))) {
          if (parameters.length == 2) {
            if (flags.DISPLAYSCREEN) {
              System.out.println("\n[ " + command + " " +
                parameters[1] + " ]\n");
            }

/*
                sprintf (backupcommand, "cp   %s %s", new_filename,
              AUVSCRIPTFILENAME);

            sprintf (backupcommand, "copy %s %s", new_filename,
                AUVSCRIPTFILENAME);

            if (DISPLAYSCREEN)
              printf ("%s\n", backupcommand);
            system   (       backupcommand);
            auvscriptfile == NULL; // force re-read
*/
          }
          else {
            if (flags.DISPLAYSCREEN) {
              System.out.println("\n " + command +
                "  Warning!  no filename present, ignored\n");
            }
```

```
            }
        } // end of else if (( command.equalsIgnoreCase( "MISSION" )) block

        return (true);
        //*****************************************************
    } // end of match commands

    /**
     * Print the flags string passed.
     *  @param:                 String flag
     *  @return:     None
     */
    private void printTrace_Display(String flagString) {
        if ((flags.TRACE && flags.DISPLAYSCREEN) || (flags.DISPLAYSCREEN)) {
            System.out.println(flagString);
        }
    } // end of printTrace_Display
} // end of class Decide
```

C. ACT.JAVA

```
/* ____________________________________________________________________________________
  Title:        Act.java
  Description:  This class is responsible for acting based on the decisions
                made by the Decide class.  It basically commands the
                actuators to act based on the controls parameters already
                calculated.
  Date:         23 May, 2002
  Project:      Execution Java Software for Autonomous Underwater Vehicle,
                Naval Postgraduate School, Monterey, CA
  Compiler:     JDK 1.3.1
  Author:       Miguel A. Ayala
  Version:      1.0
  ____________________________________________________________________________________
 */

package mil.navy.nps.auvAries.execution.act;

import mil.navy.nps.auvAries.execution.globals.Execution_Flags;
import mil.navy.nps.auvAries.execution.data_processing.Data_Processing;
import mil.navy.nps.auvAries.execution.control.Control;

/**
* This class is responsible for acting based on the decisions made by the Decide
* class.  It basically commands the actuators to act based on the controls
* parameters already calculated.
*/

public class Act {

   private Execution_Flags flags;
   private Data_Processing data_Processing;
   private Control control;
   private final int BOW_RUDDER_TOP = 1;
   private final int BOW_RUDDER_BOTTOM = 1;
   private final int BOW_PLANE_STBD = 3;
   private final int BOW_PLANE_PORT = 3;
   private final int STERN_RUDDER_TOP = 2;
   private final int STERN_RUDDER_BOTTOM = 2;
   private final int STERN_PLANE_STBD = 4;
   private final int STERN_PLANE_PORT = 4;
   private final int PORT_PROP = 0;
   private final int STBD_PROP = 1;
   private final int BOW_VERTICAL = 2;
   private final int STERN_VERTICAL = 4;
   private final int BOW_LATERAL = 3;
   private final int STERN_LATERAL = 5;

   public Act( Execution_Flags flagsRef, Data_Processing extMethRef,
                     Control controlRef ) {

      flags = flagsRef;
      data_Processing = extMethRef;
```

```
    control = controlRef;

} // end of Act constructor

public void zero_gyro_data() {
}

public void zero_surfaces() {
    if (flags.TRACE) {
        System.out.println("[Start zero_surfaces()]\n");
    }
    command_rudder(0.0);
    command_planes(0.0, 0.0);
    if (flags.TRACE) {
        System.out.println("[Finish zero_surfaces()]\n");
    }
    return;
} // end of zero_surfaces()

//THIS COULD GO TO INITIALIZATION
public void initialize_adcs() {
    //REAL WORLD CODE ommitted
    if (flags.TRACE) {
        System.out.println("[Start initialize_adcs()]\n");
    }
    if (flags.LOCATIONLAB) {
        return;
    }
    if (flags.TRACE) {
        System.out.println("Finish initialize_adcs()]\n");
    }
}

public void init_tim1a() {
    //REAL WORLD CODE ommited
}

public void thruster_power() {
    if (flags.TRACE) {
        System.out.println("[Start thruster_power()]\n");
    }
    if (flags.LOCATIONLAB) {
        return;
    }
    if (flags.TRACE) {
        System.out.println("[Finish thruster_power()]\n");
    }
}

public void screw_power() {
    if (flags.TRACE) {
        System.out.println("[Start screw_power()]\n");
    }
    if (flags.LOCATIONLAB) {
        return;
    }
```

```
    if (flags.TRACE) {
        System.out.println("[Finish screw_power()]\n");
    }
}

public void command_control_surface(double angle, int surface) {
    if (flags.TRACE) {
        System.out.println("[Start command_control_surface()]\n");
        // NEEDS IN WATER CODE
    }
    if (flags.LOCATIONLAB) {
        return;
    }
    if (flags.TRACE) {
        System.out.println("[Finish command_control_surface()]\n");
    }
}

public void command_rudder(double angle) {
    if (flags.TRACE) {
        System.out.println("[Start command_rudder()]\n");
    }
    angle = Math.toRadians(angle);
    command_control_surface(angle, BOW_RUDDER_TOP);
    command_control_surface(-angle, STERN_RUDDER_TOP);
    if (flags.TRACE) {
        System.out.println("[Finish command_rudder()]\n");
    }
    return;
}

public void command_planes(double stern_angle, double bow_angle) {
    if (flags.TRACE) {
        System.out.println("[Start command_planes()]\n");
    }
    stern_angle = Math.toRadians(stern_angle);
    bow_angle = Math.toRadians(bow_angle);
    command_control_surface(bow_angle, BOW_PLANE_STBD);
    command_control_surface(stern_angle, STERN_PLANE_STBD);
    if (flags.TRACE) {
        System.out.println("[Finish command_planes()]\n");
    }
    return;
}

public void command_propellor_off() {
    if (flags.TRACE) {
        System.out.println("Start propellors_off()]\n");
    }
    command_motor(0.0, PORT_PROP);
    command_motor(0.0, STBD_PROP);
    if (flags.TRACE) {
        System.out.println("Finish propellors_off()]\n");
    }
    return;
} // end of command_propellors_off()
```

```
   public void command_thrusters_off() {
      if (flags.TRACE) {
         System.out.println("Start command_thrusters_off()]\n");
      }
      command_motor(0.0, BOW_VERTICAL);
      command_motor(0.0, STERN_VERTICAL);
      command_motor(0.0, BOW_LATERAL);
      command_motor(0.0, STERN_LATERAL);
      if (flags.TRACE) {
         System.out.println("Finish command_thrusters_off()]\n");
      }
      return;
   } // end of command_thrusters_off

   public void command_motor(double order, int motor) {
/*     motor = 0  Left  Propeller            PORT_PROP        RPM
            1  Right Propeller            STBD_PROP        RPM
            2   Bow Vertical Thruster      BOW_VERTICAL     volts
            3   Bow Lateral  Thruster      STERN_VERTICAL   volts
            4  Stern Vertical Thruster     BOW_LATERAL      volts
            5  Stern Lateral  Thruster     STERN_LATERAL    volts
*/

    /* use local variables to permit clamping without side effects          */

      int dac_value = 0;

      /* range        0..1023              */

      double propellor_rpm = order;

      /* propellors -700.. 700 rpm           */

      double thruster_volts = order;

      /* thrusters   -24..  24 volts          */

      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("[start  command_motor ()]\n");
      }
      if ((motor == PORT_PROP) || (motor == STBD_PROP)) {
         propellor_rpm = data_Processing.clamp( propellor_rpm, -700.0, 700.0,
                                                "command_motor (): propellor_rpm" );
      }
      if (motor == PORT_PROP) {
         dac_value = control.port_speed_control(propellor_rpm / 60.0);
      }
      if (motor == STBD_PROP) {
         dac_value = control.stbd_speed_control(propellor_rpm / 60.0);
      }
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         if (motor == PORT_PROP) {
            System.out.println("[PORT ");
         }
         else if (motor == STBD_PROP) {
```

```
            System.out.println("[STBD ");
        }
        System.out.println("propellor_rpm = " + propellor_rpm
            + " dac_value = " + dac_value + "]\n");
    }
    else if ((motor == BOW_VERTICAL) || (motor == STERN_VERTICAL)
        || (motor == BOW_LATERAL) || (motor == STERN_LATERAL)) {

        thruster_volts = data_Processing.clamp(thruster_volts, -24.0, 24.0,
            "command_motors (): thruster_volts");
        dac_value = (int)((thruster_volts + 24.0) * 1023.0 / 48.0);
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[thruster_volts = " + thruster_volts
                + " dac_value = " + dac_value + "]\n");
        }
    }
    else {
        /* erroneous motor number selected */

        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[command_motor (): erroneous order/motor ("
                + order + "/" + motor + ")]\n");
        }
        return;
    }
    send_dac2b(dac_value, motor);
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("[finish command_motor ()]\n");
    }
    return;
} // end command_motor ()

public void send_dac2b(int s, int ch) {
    if (flags.LOCATIONLAB) {
        return;
    }
} // end of send_dac2b()

public void test_alive() {
    if (flags.TRACE) {
        System.out.println("[Test alive method called]\n");
    }
}

public void init_pia() {
    if (flags.LOCATIONLAB) {
        return;
    }
}

public void write_tim1a() {
    if (flags.LOCATIONLAB) {
        return;
    }
} //
```

```
    public void send_dac1() {
        if (flags.LOCATIONLAB) {
            return;
        }
    } // end of send_dac1()

    public int get_adc1(int n) {
        int return_value = 0;
        if (flags.LOCATIONLAB) {
            return_value = 0;
        }
        return return_value;
    }

    public void get_adc2(int n, int g) {
        if (flags.LOCATIONLAB) {
            return;
        }
    } // end of send_adc2()
} // end of Act class
```

D. CONTROL.JAVA

```
/* ____________________________________________________________________________________
  Title:          Control.java
  Description:    This class has the control methods for the navigation of
                  the vehicle.  It also has the control loop method which is
                  the heart of vehicle control.
  Date:           May 22, 2002
  Project:        Execution Java Software for Autonomous Underwater Vehicle,
                  Naval Postgraduate School, Monterey, CA
  Compiler:       JDK 1.3.1
  Author:         Miguel A. Ayala
  Version:        1.0
  ____________________________________________________________________________________
 */

package mil.navy.nps.auvAries.execution.control;

import mil.navy.nps.auvAries.execution.globals.Execution_Flags;
import mil.navy.nps.auvAries.execution.vehicle.Vehicle;
import mil.navy.nps.auvAries.execution.hardware.*;
import mil.navy.nps.auvAries.execution.data_processing.Kalman;
import mil.navy.nps.auvAries.execution.data_processing.Data_Processing;
import mil.navy.nps.auvAries.execution.act.Act;
import mil.navy.nps.auvAries.execution.sense.Sense;
import mil.navy.nps.auvAries.execution.input_output.Fmt;
import mil.navy.nps.auvAries.execution.input_output.IO;
import mil.navy.nps.auvAries.execution.act.Act;
import mil.navy.nps.auvAries.execution.control.Control_Coefficients;
import mil.navy.nps.auvAries.execution.network.Network_Connection;
import mil.navy.nps.auvAries.execution.decide.Parser;


/** Contains all the methods that are used for dynamic control of the vehicle.
*/
public class Control {
   /** Maximum AUV rpms */
   private final double MAX_RPM;

   /** Maximum AUV thrusters */
   private final double MAX_THRUSTER;

   /** Maximum plane degrees allowed */
   private final double MAX_PLANE; // degrees

   /** Maximum rudder in degrees allowed */
   private final double MAX_RUDDER; // degrees

   /**
    *  Simulation time step.  It represents the control loop refresh cycle rate
    * set at 10 Hz or 0.1 seconds due to hardware control constraints.
    */
   public final double TIMESTEP; // seconds

   /** Delta time for mission playback.  It takes the same value as TIMESTEP */
```

```
public static double dt;

/** Vehicle current rpm */
public static double rpm;

/**  */
public static double range_from_recovery_pt;

/**  */
public static double waypoint_distance;

/**  */
public static double waypoint_angle;

/**  */
public static double track_angle;

/**  */
public static double along_track_distance;

/**  */
public static double cross_track_distance;

/**  */
public static double docking_standoff_distance;

/**  */
public static double death_spiral_radius;

/**  */
public static double depth_error; // in the code is empty????
// for target control

/**  */
public static double target_x;

/**  */
public static double target_y;

/**  */
public static double target_z;

/**  */
public static double target_bearing;

/**  */
public static double target_range;

/**  */
public static double last_range_from_left;

/**  */
public static double range_from_left;

/**  */
public static double range_from_right;
```

```
/** */
public static double side_range_error;

/** */
public static double side_range_rate;

/** */
public static double range_from_end;

/** */
public boolean new_target_update;

/** */
public double psi_command_tgt;

/** */
public double time_last_target_update;

/** */
public static double time_int_control_on;

/** */
public static double depth_error_integral;

/** */
public static double ST1000_range_kal;
static double previous_range;
static double start_bearing;
static double end_bearing;
static double range_accumulator;
static int valid_return_count;
static int no_return_count;
static boolean scan_onto_target;
static boolean update_target_data;
double new_target_bearing;
double new_target_range;
double commanded_bearing_error;
double sonar_return_x;
double sonar_return_y;

/** */
public static double auv_oceancurrent_x;

/** */
public static double auv_oceancurrent_y;

/** */
public static double auv_oceancurrent_z;

/** */
public static double current_magnitude;
//from dynamics

/** */
public static int cross_sections;
```

```
/**  */
public static double[] cross_body_flow_u;

/**  */
public static double[] cross_body_flow_v;

/**  */
public static int cross_body_control_mode;

/**  */
public static boolean waiting_for_tgt_update;

/**  */
public static double cos_tgt_brg;

/**  */
public static double sin_tgt_brg;
public static double distance_term_factor;
// THIS COULD BE UNDER ACT
static double cos_tgt_brg_cmd;
static double sin_tgt_brg_cmd;
static double commanded_psi_tgt;
//    double distance_term_factor;
// for use in followlight controls
static boolean look_around_in_progress;
static boolean firstLoopInitialized;
static double previous_psi;
static double psiLightReference;
static boolean ZeroLookAroundFlag;
static double psiStartLookAround;
static double xLookAround;
static double yLookAround;
static double zLookAround;
static boolean heading_reference;
static final double FEET_PER_METER = 3.28084; // number of feet per meter
static double CSpeed;
static double DeltaPsi;
static double DeltaZ;
//used in port_speed_control( double n_com){
double n_com;

/* revolutions per second */

double Km_ls;
double e_n;
double v_ls _spc;
double eta_ls;
double phi_ls;
double Km_rs;
double v_rs_spc;
double eta_rs;
double phi_rs;
// coordinates of the center of the light source in the astyanax coordinate
// system
double Light_Source_center_X;
```

```
double Light_Source_center_Y;
double Light_Source_center_Z;
double distanceAUVLight;
// USE by the Thrusters Speed Controller Routines
double Int_rs;
double Int_ls;
//THIS POSSIBLE WILL GO TO A HARDWARE SPECIFIC
//DAC VALUES BEING SENT %) PROPS AND THRUSTERS
int v_dls;
int v_drs;
int v_dblt;
int v_dslt;
int v_dbvt;
int v_dsvt;
boolean end_test;
private double delta_rudder;
final int PORT_PROP;
final int STBD_PROP;
final int BOW_VERTICAL;
final int STERN_VERTICAL;
final int BOW_LATERAL;
final int STERN_LATERAL;
public double cos_psi;
public double sin_psi;
public double cos_phi;
public double time_next_command;
private double psi_im1;
public final double SONAR_HEADING_STEP = 0.0; // used in step_ST1000_sonar
private Execution_Flags flags;
private Vehicle vehicle;
private ST725_sonar ST725;
private ST1000_sonar ST1000;
private Kalman kalman;
private Data_Processing Data_Processing;
private Act act;
private Control_Coefficients control_coefficients;
private Sense sense;
private IO io;
private Network_Connection network;
//COMING FROM ACT
public static double time_gps_complete;
public static double time_postgps_dive;
public static double psi_command;
public static double previuos_psi;
public static double psi_command_hover;
public static double theta_command;
public static double x_command;
public static double y_command;
public static double z_command;
public static double stbd_rpm_command;
public static double port_rpm_command;
public static double planes_command_stern;
public static double planes_command_bow;
public static double rudder_command;
public static double rotate_command;
public static double lateral_command;
```

```
public static double bow_lateral_thruster_command;
public static double stern_lateral_thruster_command;
public static double bow_vertical_thruster_command;
public static double stern_vertical_thruster_command;
public static double previous_x_command;
public static double previous_y_command;
public static double previous_z_command;
public static double target_range_command;
public static double target_bearing_command;
public static double ST1000_command;
public static double computer_voltage;
public static double motor_voltage;
public static double dt_time;


/**
 *  Constructor for Control class
 *  @param: Execution_Flags object, Vehicle object, Sense object,
 *  @param: Data_Processing object, Kalman object,
 *      @param:             Control_Coefficients object, IO object,
 *      @param:             Network_Connection object
 *  @return: None
 */
public Control(Execution_Flags flagsObj, Vehicle vehicleObj, Sense senseRef,
   Data_Processing Data_ProcessingRef, Kalman kalmanRef,
   Control_Coefficients control_coefficientsRef, IO ioRef,
   Network_Connection networkRef) {

      //*********************************
      //**          Dependency classes    **
      //*********************************
      flags = flagsObj;
      vehicle = vehicleObj;
      sense = senseRef;
      Data_Processing = Data_ProcessingRef;
      kalman = kalmanRef;
      control_coefficients = control_coefficientsRef;
      io = ioRef;
      network = networkRef;
      MAX_RPM = 700.0;
      MAX_THRUSTER = 0.0;
      MAX_PLANE = 22.5; // degrees
      MAX_RUDDER = 22.5; // degrees
      TIMESTEP = 0.10; // seconds
      PORT_PROP = 0;
      STBD_PROP = 1;
      BOW_VERTICAL = 2;
      STERN_VERTICAL = 4;
      BOW_LATERAL = 3;
      STERN_LATERAL = 5;
      cos_psi = 0.0;
      sin_psi = 0.0;
      cos_phi = 0.0;
      rpm = 0.0;
      psi_im1 = 0.0;
      depth_error_integral = 0;
```

```
cross_sections = 15;
cross_body_flow_u = new double[cross_sections];
cross_body_flow_v = new double[cross_sections];
cross_body_control_mode = 2;
time_int_control_on = 1000000.0; //  give PD a chance to get close
range_from_end = 10.0;
last_range_from_left = 0.0;
new_target_update = false;
psi_command_tgt = 0.0;
time_last_target_update = 0.0;
waiting_for_tgt_update = false;
cos_tgt_brg = 0.0;
sin_tgt_brg = 0.0;
cos_tgt_brg_cmd = 0.0;
sin_tgt_brg_cmd = 0.0;
commanded_psi_tgt = 0.0;
// to be used in followloight controls
look_around_in_progress = true;
firstLoopInitialized = false;
previous_psi = 0;
psiLightReference = 0;
ZeroLookAroundFlag = false;
psiStartLookAround = 0;
xLookAround = 0;
yLookAround = 0;
zLookAround = 0;
heading_reference = false;
CSpeed = 0.0;
DeltaPsi = 0.0;
DeltaZ = 0.0;
n_com = 0.0;

/* revolutions per second */

Km_ls = 0.6589;
e_n = 0.0;
v_ls_spc = 0.0;
eta_ls = 10.0;
phi_ls = 5.0;
Km_rs = 0.6156;
e_n = 0.0;
v_rs_spc = 0.0;
eta_rs = 10.0;
phi_rs = 5.0;
Light_Source_center_X = 0.0;
Light_Source_center_Y = 0.0;
Light_Source_center_Z = 0.0;
distanceAUVLight = 0.0;
time_int_control_on = 0.0;
//THIS POSSIBLE WILL GO TO A HARDWARE SPECIFIC
//DAC VALUES BEING SENT %) PROPS AND THRUSTERS
v_dls = 512;
v_drs = 512;
v_dblt = 512;
v_dslt = 512;
v_dbvt = 512;
```

```
        v_dsvt = 512;
        // USE by the Thrusters Speed Controller Routines
        Int_rs = 0.0;
        Int_ls = 0.0;
        end_test = false;
        previous_range = 0.0;
        start_bearing = 0.0;
        end_bearing = 0.0;
        range_accumulator = 0.0;
        valid_return_count = 0;
        no_return_count = 0;
        scan_onto_target = false;
        update_target_data = false;
        new_target_bearing = 0.0;
        new_target_range = 0.0;
        commanded_bearing_error = 0.0;
        sonar_return_x = 0.0;
        sonar_return_y = 0.0;
        ////////////////////////////////
        port_rpm_command = 700;
        stbd_rpm_command = 700;
        dt_time = 0;

} // end of control constructor

/**
 *  Computes hover control
 *  @param:      None
 *  @return:     None
 */
void compute_hover_controls() { // VERIFIED
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("[begin compute_hover_controls]\n");
    }
    compute_vertical_thrusters();
    // Distant hoverpoint uses waypoint control until closer
    if ((waypoint_distance > control_coefficients.get_standoff_distance()
        + 20.0) && (detect_death_spiral() == false)) {

        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[Hoverpoint switch to WAYPOINTCONTROL]\n");
        }
        port_rpm_command = 700;
        stbd_rpm_command = 700;
        flags.WAYPOINTCONTROL = true;
        flags.DEADSTICKRUDDER = false;
        flags.DEADSTICKPLANES = false;
        compute_waypoint_controls();
    }
    // close hoverpoint uses hover control
    else {
        flags.WAYPOINTCONTROL = false;
        psi_command = psi_command_hover;
        // report STABLE to tactical level once hoverpoint reached
        if ((flags.HOVERCONTROL) && (flags.REPORTSTABLE) &&
            ((waypoint_distance < control_coefficients.get_standoff_distance()) &&
```

```
   (Math.abs(depth_error) < control_coefficients.get_standoff_distance()) &&
   (Math.abs(Data_Processing.normalize2(vehicle.get_psi() - psi_command)) < 2.5))) {
      if ((flags.TACTICAL) && (flags.GPSFIXINPROGRESS == false)) {
         flags.REPORTSTABLE = false;
         //  strcpy (buffer, "STABLE HOVER");
         if (flags.DISPLAYSCREEN) System.out.println("\nSTABLE HOVER");
         // send_buffer_to_tactical_socket (); // message
      }
}
// Compute Control Settings /
waypoint_angle = Data_Processing.normalize
   (Data_Processing.degrees(Data_Processing.atan2z(y_command -
   vehicle.get_y(), x_command - vehicle.get_x())));

track_angle = Data_Processing.normalize(waypoint_angle - vehicle.get_psi());

along_track_distance = Math.cos(Math.toRadians(track_angle)) *
   waypoint_distance;

cross_track_distance = -Math.sin(Math.toRadians(track_angle)) *
   waypoint_distance;

port_rpm_command = control_coefficients.get_k_propeller_hover() *
   along_track_distance - control_coefficients.get_k_propeller_current() *
   auv_oceancurrent_x * Math.cos(vehicle.get_psi()) -
   control_coefficients.get_k_propeller_current() *
   auv_oceancurrent_y * Math.sin(vehicle.get_psi()) -
   control_coefficients.get_k_surge_hover() * vehicle.get_u();

stbd_rpm_command = port_rpm_command;

if (flags.TRACE && flags.DISPLAYSCREEN) {
   System.out.println("\n HOVERCONTROL: \n");
   System.out.println("psi_command = " + psi_command);
   System.out.println("x = " + vehicle.get_x() + ", y = " +
      vehicle.get_y()); // this are from vehicle
   System.out.println("waypoint_distance = " + waypoint_distance +
      ", track_angle = " + track_angle);
   System.out.println("along_track_distance = " + along_track_distance);
   System.out.println("cross_track_distance = " + cross_track_distance);
   System.out.println("port_rpm & stbd_rpm = " + vehicle.get_port_rpm());
}

vehicle.set_auv_bow_lateral(-(-control_coefficients.get_k_thruster_psi() *
   Data_Processing.normalize2(vehicle.get_psi() - psi_command) -
   control_coefficients.get_k_thruster_r() * vehicle.get_r()) +
   control_coefficients.get_k_thruster_hover() * cross_track_distance -
   control_coefficients.get_k_thruster_current() * auv_oceancurrent_x *
   Math.sin(vehicle.get_psi()) +
   control_coefficients.get_k_thruster_current() * auv_oceancurrent_y *
   Math.sin(vehicle.get_psi()) +
   control_coefficients.get_k_sway_hover() * vehicle.get_v() +
   control_coefficients.get_k_thruster_current() *
   cross_body_flow_v[1]);

vehicle.set_auv_stern_lateral((-control_coefficients.get_k_thruster_psi() *
```

```
        Data_Processing.normalize2(vehicle.get_psi() - psi_command) -
        control_coefficients.get_k_thruster_r() * vehicle.get_r())
        + control_coefficients.get_k_thruster_hover() * cross_track_distance -
        control_coefficients.get_k_thruster_current() * auv_oceancurrent_x *
        Math.sin(vehicle.get_psi()) +
        control_coefficients.get_k_thruster_current() * auv_oceancurrent_y *
        Math.sin(vehicle.get_psi()) +
        control_coefficients.get_k_sway_hover() * vehicle.get_v() +
        control_coefficients.get_k_thruster_current() *
        cross_body_flow_v[12]);
    } // end of else of the if (( waypoint_distance > standoff_distance + 20.0 ) block

    // Extend time till next command if not at hover point yet /
    if ((flags.HOVERCONTROL) && (flags.GPSFIXINPROGRESS == false) &&
       ((waypoint_distance > control_coefficients.get_standoff_distance()) ||
       (Math.abs(depth_error) > control_coefficients.get_standoff_distance()) ||
       (Math.abs(Data_Processing.normalize2(vehicle.get_psi() - psi_command)) > 10.0 ))) {
          // still not at the hoverpoint
          if (flags.TRACE && flags.DISPLAYSCREEN)
             System.out.println("[HOVERCONTROL cylinder test]");
          // continue until hoverpt reached without further script orders
          time_next_command = vehicle.get_time() + 2.0 * dt;
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
       System.out.println("[end compute_hover_controls]\n");
    }
    return;
} // end of public void compute_hover_controls()

/**
 *  Compute docking control
 *  @param:      None
 *  @return:     None
 */
public void compute_docking_controls() { // VERIFIED
    if (flags.TRACE && flags.DISPLAYSCREEN) {
       System.out.println("\n[begin compute_docking_controls \n");
    }
    compute_vertical_thrusters();
    // Distant hoverpoint uses waypoint control until closer
    if ((waypoint_distance > control_coefficients.get_standoff_distance() +
       20.0) && (detect_death_spiral() == false)) {
       if (flags.TRACE && flags.DISPLAYSCREEN) {
          System.out.println("[Dockingpoint switch to WAYPOINTCONTROL]\n");
       }
       port_rpm_command = 700;
       stbd_rpm_command = 700;
       flags.WAYPOINTCONTROL = true;
       flags.DEADSTICKRUDDER = false;
       flags.DEADSTICKPLANES = false;
       compute_waypoint_controls();
    }
    else {
       flags.WAYPOINTCONTROL = false;
       psi_command = psi_command_hover;
       // report STABLE to tactical level once hoverpoint reached
```

```
if ((flags.DOCKINGCONTROL) && (flags.REPORTSTABLE) &&
   ((waypoint_distance < docking_standoff_distance) &&
   (Math.abs(depth_error) < docking_standoff_distance) &&
   (Math.abs(Data_Processing.normalize2(vehicle.get_psi() -
   psi_command)) < 2.5))) {
      if ((flags.TACTICAL) && (flags.GPSFIXINPROGRESS == false)) {
         flags.REPORTSTABLE = false;
         if (flags.DISPLAYSCREEN) System.out.println("\nSTABLE HOVER");
         // send_buffer_to_tactical_socket (); // message
         network.write_to_tactical("STABLE HOVER");
      }
}
// Compute Control Settings
waypoint_angle = Data_Processing.normalize
   (Data_Processing.degrees(Data_Processing.atan2z(y_command -
   vehicle.get_y(), x_command - vehicle.get_x())));
track_angle = Data_Processing.normalize(waypoint_angle - vehicle.get_psi());
along_track_distance = Math.cos(Math.toRadians(track_angle)) * waypoint_distance;
cross_track_distance = -Math.sin(Math.toRadians(track_angle)) * waypoint_distance;
port_rpm_command = control_coefficients.get_k_propeller_hover() * along_track_distance -
   control_coefficients.get_k_propeller_current() * auv_oceancurrent_x *
   Math.cos(vehicle.get_psi()) -
   control_coefficients.get_k_propeller_current() * auv_oceancurrent_y *
   Math.sin(vehicle.get_psi()) -
   control_coefficients.get_k_surge_hover() * vehicle.get_u();

stbd_rpm_command = port_rpm_command;

if (flags.TRACE && flags.DISPLAYSCREEN) {
   System.out.println("\n DOCKINGCONTROL: \n");
   System.out.println("psi_command = " + psi_command);
   System.out.println("x = " + vehicle.get_x() + ", y = " +
      vehicle.get_y()); // this are from vehicle
   System.out.println("waypoint_distance = " + waypoint_distance +
      ", track_angle = " + track_angle);
   System.out.println("along_track_distance = " + along_track_distance);
   System.out.println("cross_track_distance = " + cross_track_distance);
   System.out.println("port_rpm & stbd_rpm = " + vehicle.get_port_rpm());
}
vehicle.set_auv_bow_lateral(-(-control_coefficients.get_k_thruster_psi() *
   Data_Processing.normalize2(vehicle.get_psi() - psi_command) -
   control_coefficients.get_k_thruster_r() * vehicle.get_r())
   + control_coefficients.get_k_thruster_hover() * cross_track_distance -
   control_coefficients.get_k_thruster_current() * auv_oceancurrent_x *
   Math.sin(vehicle.get_psi()) +
   control_coefficients.get_k_thruster_current() * auv_oceancurrent_y *
   Math.cos(vehicle.get_psi()) +
   control_coefficients.get_k_sway_hover() * vehicle.get_v() +
   control_coefficients.get_k_thruster_current() *
   cross_body_flow_v[1]);

vehicle.set_auv_stern_lateral((-control_coefficients.get_k_thruster_psi() *
   Data_Processing.normalize2(vehicle.get_psi() - psi_command) -
   control_coefficients.get_k_thruster_r() * vehicle.get_r())
   + control_coefficients.get_k_thruster_hover() * cross_track_distance -
   control_coefficients.get_k_thruster_current() * auv_oceancurrent_x *
```

```
            Math.sin(vehicle.get_psi()) +
            control_coefficients.get_k_thruster_current() * auv_oceancurrent_y *
            Math.cos(vehicle.get_psi()) +
            control_coefficients.get_k_sway_hover() * vehicle.get_v() +
            control_coefficients.get_k_thruster_current() *
            cross_body_flow_v[12]);

    } // end of else of the if (( waypoint_distance > standoff_distance + 20.0 ) block
    // Extend time till next command if not at hover point yet
    if ((flags.DOCKINGCONTROL) && (flags.GPSFIXINPROGRESS == false) &&
        ((waypoint_distance > docking_standoff_distance) ||
        (Math.abs(depth_error) > docking_standoff_distance) ||
        (Math.abs(Data_Processing.normalize2(vehicle.get_psi() -
        psi_command)) > 10.0))) { // cylinder test
            // still not at the hoverpoint
            if (flags.TRACE && flags.DISPLAYSCREEN) {
                System.out.println("[HOVERCONTROL cylinder test]");
            }
            // continue until dockpt reached without further script orders
            time_next_command = vehicle.get_time() + 2.0 * dt;
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("[end compute_docking_controls]\n");
    }
    return;
} // end of compute_docking_controls ()

/**
 *  Compute recovery controls
 *  @param:      None
 *  @return:     None
 */
public void compute_recovery_controls() { // VERIFIED
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("[Begin compute_recovery_controls]\n");
    }
    last_range_from_left = 0.0;
    range_from_end = 10.0;
    range_from_recovery_pt -= vehicle.get_u() * dt;
    // if both sonars are not in place, hover in place
    if ((Data_Processing.normalize(vehicle.get_auv_ST1000_bearing()) > 286.0) ||
        (Data_Processing.normalize(vehicle.get_auv_ST1000_bearing()) < 285.0) ||
        (Data_Processing.normalize2(vehicle.get_auv_ST725_bearing()) > 0.5) ||
        (Data_Processing.normalize2(vehicle.get_auv_ST725_bearing()) < -0.5)) {
            if (flags.TRACE && flags.DISPLAYSCREEN) {
                System.out.println("[Using hover control until sonar in place]\n");
                System.out.println("[auv_ST1000_bearing] = " +
                    vehicle.get_auv_ST1000_bearing());
                System.out.println("[auv_ST725_bearing] = " +
                    vehicle.get_auv_ST725_bearing());
            }
            compute_hover_controls();
    }
    else {
        kalman.kalman_sonar725(vehicle.get_auv_ST725_range());
        range_from_left = Math.abs(Math.sin(Math.toRadians(
```

```
        vehicle.get_auv_ST1000_bearing()))) * kalman.ST1000_range_kal;
    range_from_end = kalman.ST725_range_kal + ST725.auv_ST725_x_offset;
    side_range_error = range_from_left - 1.5;
    if (flags.NEWRECOVERYCOMMAND) {
        side_range_rate = 0.0;
        flags.NEWRECOVERYCOMMAND = false;
    }
    else {
        side_range_rate = (range_from_left - last_range_from_left) / dt;
    }
    last_range_from_left = range_from_left;
    compute_vertical_thrusters();
} // end of else of if (( normalize ( vehicle.get_auv_ST1000_bearing() )

// Compute required propeller power
port_rpm_command = control_coefficients.get_k_propeller_hover() *
    (range_from_end - 5.0) -
    control_coefficients.get_k_propeller_current() * auv_oceancurrent_x
        * Math.cos(vehicle.get_psi()) -
    control_coefficients.get_k_propeller_current() * auv_oceancurrent_y
        * Math.sin(vehicle.get_psi()) -
    control_coefficients.get_k_surge_hover() * vehicle.get_u();

stbd_rpm_command = port_rpm_command;

// Compute lateral thruster power /
vehicle.set_auv_bow_lateral(-(-control_coefficients.get_k_thruster_psi() *
    Data_Processing.normalize2(vehicle.get_psi() - psi_command) -
    control_coefficients.get_k_thruster_r() *
    vehicle.get_r()) + control_coefficients.get_k_thruster_hover() *
    side_range_error - control_coefficients.get_k_thruster_current() *
    auv_oceancurrent_x * Math.sin(vehicle.get_psi()) +
    control_coefficients.get_k_thruster_current() * auv_oceancurrent_y *
    Math.cos(vehicle.get_psi()) +
    control_coefficients.get_k_sway_hover() * side_range_rate +
    control_coefficients.get_k_thruster_current() * cross_body_flow_v[1]);

vehicle.set_auv_stern_lateral((-control_coefficients.get_k_thruster_psi() *
    Data_Processing.normalize2(vehicle.get_psi() - psi_command) -
    control_coefficients.get_k_thruster_r() *
    vehicle.get_r()) + control_coefficients.get_k_thruster_hover() *
    side_range_error - control_coefficients.get_k_thruster_current() *
    auv_oceancurrent_x * Math.sin(vehicle.get_psi()) +
    control_coefficients.get_k_thruster_current() * auv_oceancurrent_y *
    Math.cos(vehicle.get_psi()) +
    control_coefficients.get_k_sway_hover() * side_range_rate +
    control_coefficients.get_k_thruster_current() *
    cross_body_flow_v[12]);

if (flags.TRACE && flags.DISPLAYSCREEN) {
    System.out.println("[Recovery Control Being Used]\n");
    System.out.println("[Range From End  : " + range_from_end);
    System.out.println("[Surge          : " + vehicle.get_u());
    System.out.println("[Range From Left : " + range_from_left);
    System.out.println("[Side Range Rate : " + side_range_rate);
    System.out.println("[Side Range Error: " + side_range_error);
```

```
    }
    if ((flags.REPORTSTABLE) && (range_from_end < 5.0)) {
        if (flags.TACTICAL) {
            flags.REPORTSTABLE = false;
            if (flags.DISPLAYSCREEN)
                System.out.println("\nSTABLE RECOVERY\n");
            network.write_to_tactical("STABLE RECOVERY");
        }
    }
    // Extend time till next command if not at hover point yet
    // report STABLE to tactical level once hoverpoint reached
    if (range_from_end > 5.0) {
        // still not at the recovery point
        if (flags.TRACE && flags.DISPLAYSCREEN)
            System.out.println("[RECOVERY cylinder test]\n");
        // continue until recoverpt reached without further script orders
        time_next_command = vehicle.get_time() + 2.0 * dt;
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("[COMPUTED RECOVERY PARAMETERS AND CONTROLS]\n");
        System.out.println("[Range From Final Pt: " + range_from_recovery_pt);
        System.out.println("[Range From Right: " + range_from_right);
        System.out.println("[Range From LEFT: " + range_from_left);
        System.out.println("[Range Error: " + side_range_error);
        System.out.println("[Side Range Rate: " + side_range_rate);
        System.out.println("[Stbd Propeller: " + stbd_rpm_command);
        System.out.println("[Port Propeller: " + port_rpm_command);
        System.out.println("[AUV Bow Lateral: " + vehicle.get_auv_bow_lateral());
        System.out.println("[AUV Stern Lateral: " + vehicle.get_auv_stern_lateral());
        System.out.println("[End compute_recovery_controls]\n");
    }
    return;
} // end of compute_recovery_controls ()

/**
 *  Computes target control
 *  @param:      None
 *  @return:     None
 */
public void compute_target_controls() { // VERIFIED
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("[Begin compute_target_controls]\n");
    }
    waiting_for_tgt_update = false;
    distance_term_factor = 1.0;
    if ((new_target_update) || (flags.NEWTARGETSTATION)) {
        cos_tgt_brg = Math.cos(Math.toRadians(target_bearing));
        sin_tgt_brg = Math.sin(Math.toRadians(target_bearing));
        cos_tgt_brg_cmd = Math.cos(Math.toRadians(Data_Processing.normalize(
            180.0 + target_bearing_command)));
        sin_tgt_brg_cmd = Math.sin(Math.toRadians(Data_Processing.normalize(
            180.0 + target_bearing_command)));
        // Compute World Space Location of Station Point
        x_command = vehicle.get_x() + cos_tgt_brg * target_range +
            cos_tgt_brg_cmd * target_range_command;
        y_command = vehicle.get_y() + sin_tgt_brg * target_range +
```

```
        sin_tgt_brg_cmd * target_range_command;
    if (flags.NEWTARGETSTATION) {
        commanded_psi_tgt = psi_command;
    }
    flags.NEWTARGETSTATION = false;
    if (new_target_update) {
        waiting_for_tgt_update = false;
    }
    new_target_update = false;
}
if (waiting_for_tgt_update == false) {
    if (flags.TARGETPOINTING) {
        psi_command_tgt = Data_Processing.degrees
            (Data_Processing.atan2z((target_y - vehicle.get_y()),
            (target_x - vehicle.get_x())));
    }
    else {
        psi_command_tgt = commanded_psi_tgt;
    }
}
// if it has been a while since the last target update, wait here for next one
if ((waiting_for_tgt_update == false) && (((flags.TARGETEDGETRACK) &&
    (time_last_target_update + 5.0 <= vehicle.get_time())) ||
    ((flags.TARGETEDGETRACK == false) &&
    (time_last_target_update + 7.5 <= vehicle.get_time()))))) {
        if (flags.DISPLAYSCREEN) {
            System.out.println("Hovering Until New Target Update\n");
            System.out.println("X: " + x_command + " Y: " + y_command +
                " Psi: " + psi_command_hover + " \n");
        }
        x_command = vehicle.get_x();
        y_command = vehicle.get_y();
        psi_command_hover = vehicle.get_psi();
        waiting_for_tgt_update = true;
}
// Re-calculate waypoint distance and depth error
// to account for possibly moving target
waypoint_distance = Math.sqrt((vehicle.get_x() - x_command) *
    (vehicle.get_x() - x_command) +
    (vehicle.get_y() - y_command) * (vehicle.get_y() - y_command));
// If waiting for target update, use hovercontrol to hold posit
if (waiting_for_tgt_update) {
    compute_hover_controls();
    if (flags.REPORTSTABLE) {
        time_next_command = vehicle.get_time() + 2.0 * dt;
    }
    return;
}
waypoint_angle = Data_Processing.normalize
    (Data_Processing.degrees(Data_Processing.atan2z(y_command -
    vehicle.get_y(), x_command - vehicle.get_x())));
track_angle = Data_Processing.normalize(waypoint_angle - vehicle.get_psi());
along_track_distance = Math.cos(Math.toRadians(track_angle)) * waypoint_distance;
cross_track_distance = -Math.sin(Math.toRadians(track_angle)) * waypoint_distance;
port_rpm_command = control_coefficients.get_k_propeller_hover() * along_track_distance -
    control_coefficients.get_k_propeller_current() * auv_oceancurrent_x *
```

```
   Math.cos(vehicle.get_psi()) - control_coefficients.get_k_propeller_current()
   * auv_oceancurrent_y * Math.sin(vehicle.get_psi()) -
   control_coefficients.get_k_surge_hover() / 2.5 * vehicle.get_u();

stbd_rpm_command = port_rpm_command;

if (flags.TRACE && flags.DISPLAYSCREEN) {
   System.out.println("\nTARGETCONTROL:\n");
   System.out.println("Station Point: " + target_bearing_command +
      " Degrees at " + target_range_command + " Feet\n");
   System.out.println("psi_command = " + psi_command_tgt + "\n");
   System.out.println("Current Point: " + target_bearing +
      " Degrees at " + target_range + " Feet\n");
   System.out.println("Computed Station Point: " + x_command +
      ", " + y_command + ", " + z_command + "\n");
   System.out.println("waypoint_distance = " + waypoint_distance +
      ", track_angle = " + track_angle + "\n");
   System.out.println("along_track_distance = " + along_track_distance);
   System.out.println("cross_track_distance = " + cross_track_distance);
   System.out.println("port_rpm & stbd_rpm  = " + vehicle.get_port_rpm());
}
vehicle.set_auv_bow_lateral(-(-control_coefficients.get_k_thruster_psi() / 3.0 *
   Data_Processing.normalize2(vehicle.get_psi() - psi_command_tgt) -
   control_coefficients.get_k_thruster_r() *
   vehicle.get_r()) + control_coefficients.get_k_thruster_hover() / 1.5
   * cross_track_distance + control_coefficients.get_k_sway_hover() / 2.0
   * vehicle.get_v() - control_coefficients.get_k_thruster_current() *
   auv_oceancurrent_x * Math.sin(vehicle.get_psi()) +
   control_coefficients.get_k_thruster_current() * auv_oceancurrent_y *
   Math.cos(vehicle.get_psi()) +
   control_coefficients.get_k_thruster_current() * cross_body_flow_v[1]);

vehicle.set_auv_stern_lateral((-control_coefficients.get_k_thruster_psi() / 3.0 *
   Data_Processing.normalize2(vehicle.get_psi() - psi_command_tgt) -
   control_coefficients.get_k_thruster_r() *
   vehicle.get_r()) + control_coefficients.get_k_thruster_hover() / 1.5
   * cross_track_distance + control_coefficients.get_k_sway_hover() / 2.0
   * vehicle.get_v() - control_coefficients.get_k_thruster_current() *
   auv_oceancurrent_x * Math.sin(vehicle.get_psi()) +
   control_coefficients.get_k_thruster_current() * auv_oceancurrent_y *
   Math.cos(vehicle.get_psi()) +
   control_coefficients.get_k_thruster_current() * cross_body_flow_v[12]);

depth_error = (z_command - kalman.z_kal);

// constrain depth_error to +- 15.0 feet to prevent going vertical
//      and enable stable pitch angle even on large depth changes
depth_error = Data_Processing.clamp(depth_error, -15.0, 15.0, "depth_error"); // feet
compute_vertical_thrusters();
// If we are not at the station point yet, continue
if ((flags.REPORTSTABLE) && ((time_last_target_update +
   1.0 < vehicle.get_time()) || (waypoint_distance > 0.5) ||
   (Math.abs(Data_Processing.normalize2(vehicle.get_psi() -
   psi_command_tgt)) > 5.0))) {
      time_next_command = vehicle.get_time() + 2.0 * dt;
}
```

```
    else {
        if ((flags.TACTICAL) && (flags.REPORTSTABLE)) {
            network.write_to_tactical("STABLE TARGET STATION");
            if (flags.DISPLAYSCREEN)
                System.out.println("\nSTABLE TARGERT STATION\n");
        }
        flags.REPORTSTABLE = false;
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("[End compute_target_controls]\n");
    }
    return;
} // end compute_target_controls ()

/**
 *   Computes waypoint controls
 *  @param:      None
 *  @return:     None
 */
public void compute_waypoint_controls() { // VERIFIED
    if (flags.TRACE && flags.DISPLAYSCREEN)
        System.out.println("[begin compute_waypoint_controls]\n");
    if ((port_rpm_command < 200.0) || (stbd_rpm_command < 200)) {
        if (flags.TRACE && flags.DISPLAYSCREEN)
            System.out.println("[WAYPOINTCONTROL rpm too low, reset to 400.0]\n");
        port_rpm_command = 400.0; // boost it higher, 200-400 are OK
        stbd_rpm_command = 400.0;
    }
    waypoint_angle = Data_Processing.atan2z(y_command - vehicle.get_y() +
        auv_oceancurrent_y * dt,
        x_command - vehicle.get_x() + auv_oceancurrent_x * dt);
    waypoint_angle = Data_Processing.normalize(Data_Processing.degrees(waypoint_angle));
    psi_command = waypoint_angle;
    if (flags.THRUSTERCONTROL)
        compute_lateral_thrusters();
    // If the auv is closer to the waypoint than this value, there
    // is a danger that it could enter a death spiral
    death_spiral_radius = Math.abs(Math.sin(Math.toRadians(
        Data_Processing.normalize2(waypoint_angle - vehicle.get_psi()))))
        * (rpm / 700.0) * 15.0;
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("death_spiral_radius =  " + death_spiral_radius);
        System.out.println("WAYPOINTCONTROL psi_command =  " + psi_command);
        System.out.println("x = " + vehicle.get_x() + ", y = " + vehicle.get_y());
    }
    //Waypoint not reached, continue
    if ((flags.FOLLOWWAYPOINTMODE) && (flags.HOVERCONTROL == false) &&
        (detect_death_spiral() == false) && // check it
        (((waypoint_distance > control_coefficients.get_standoff_distance()) &&
        (waypoint_distance > death_spiral_radius))
        || (Math.abs(depth_error) > control_coefficients.get_standoff_distance()))) {
            if (flags.TRACE && flags.DISPLAYSCREEN)
                System.out.println("\n[FOLLOWWAYPOINTMODE cylinder test]");
            // continue until WAYPOINT reached without further script orders
            time_next_command = vehicle.get_time() + 2.0 * dt;
    }
```

```
    // Waypoint Reached
    else if ((Math.abs(depth_error) <= control_coefficients.get_standoff_distance()) &&
      ((waypoint_distance <= control_coefficients.get_standoff_distance()) ||
      (waypoint_distance <= death_spiral_radius) || (detect_death_spiral()))) { // check it
        flags.WAYPOINTCONTROL = false;
        flags.FOLLOWWAYPOINTMODE = false;
        if (flags.TRACE && flags.DISPLAYSCREEN)
          System.out.println("\n[FOLLOWWAYPOINTMODE success, WAYPOINT reached]");
        if (flags.HOVERCONTROL == false) flags.DEATH_SPIRAL_RESET = true;
        // report STABLE to tactical level once waypoint received
        if ((flags.TACTICAL) && (flags.REPORTSTABLE) && (flags.HOVERCONTROL == false)
&&
          (flags.GPSFIXINPROGRESS == false)) {
            flags.REPORTSTABLE = false;
            //  strcpy (buffer, "STABLE WAYPOINT");
            if (flags.DISPLAYSCREEN)
              System.out.println("\nSTABLE WAYPOINT");
            network.write_to_tactical("STABLE WAYPOINT");
        }
    }
    if (flags.TRACE && flags.DISPLAYSCREEN)
      System.out.println("[end compute_waypoint_controls]\n");
    return;
  } // end compute_waypoint_controls ()

  /**
   *  Computes lateral controls
   *  @param:      None
   *  @return:     None
   */
  public void compute_lateral_controls() { //  VERIFIED
    if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[begin compute_lateral_controls]\n");;
    }
    compute_vertical_thrusters();
    vehicle.set_auv_bow_lateral(-control_coefficients.get_k_thruster_lateral()
      * lateral_command);
    vehicle.set_auv_stern_lateral(vehicle.get_auv_bow_lateral());
    psi_command = vehicle.get_psi();
    if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[end compute_lateral_controls]\n");
    }
    return;
  } // end of void compute_lateral_controls ()

  /**
   *  Computate rotate controls
   *  @param:      None
   *  @return:     None
   *  @exception:   None
   */
  public void compute_rotate_controls() { //  VERIFIED
    if (flags.TRACE && flags.DISPLAYSCREEN)
      System.out.println("[begin compute_rotate_controls]\n");
    compute_vertical_thrusters();
    if (flags.TRACE && flags.DISPLAYSCREEN) {
```

```
            System.out.println("( ROTATECONTROL == TRUE )\n");
        }
        vehicle.set_auv_stern_lateral(control_coefficients.get_k_thruster_rotate()
            * rotate_command);
        vehicle.set_auv_bow_lateral(-vehicle.get_auv_stern_lateral()); // negative
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[end compute_rotate_controls]\n");
        }
    } // end of void compute_rotate_controls (

    /**
     *  Computes lateral thrusters control
     *  @param:      None
     *  @return:     None
     */
    public void compute_lateral_thrusters() { //  VERIFIED
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[begin compute_lateral_thrusters]\n");
        }
        compute_vertical_thrusters();
        vehicle.set_auv_stern_lateral(-control_coefficients.get_k_thruster_psi() *
            Data_Processing.normalize2(vehicle.get_psi() - psi_command) -
            control_coefficients.get_k_thruster_r() * vehicle.get_r());
        vehicle.set_auv_bow_lateral(-vehicle.get_auv_stern_lateral());
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[end compute_lateral_thrusters]\n");
        }
        return;
    } // end of public void compute_lateral_thrusters ()

    /**
     *  Computes vertical thrusters control
     *  @param:      None
     *  @return:     None
     */
// NEEDS VERIFICATION ON AUV_bow_vertical AND AUV_stern_vertical
    public void compute_vertical_thrusters() {
        depth_error_integral = 0.0;
        if (flags.TRACE && flags.DISPLAYSCREEN)
            System.out.println("[begin compute_vertical_thrusters]\n");
        if ((flags.INTEGRALDEPTHCONTROL == 0) &&
            (vehicle.get_time() >= time_int_control_on)) {
            flags.INTEGRALDEPTHCONTROL = 1;
        }
        // Only use error in INTEGRAL CONTROL only, clamp to avoid saturation
        depth_error_integral = (depth_error_integral + (kalman.z_kal - z_command) *
            flags.TIMESTEP) *
            flags.INTEGRALDEPTHCONTROL;
        depth_error_integral = Data_Processing.clamp(depth_error_integral, -2.0, 2.0,
                                        "depth_error_integral");
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[z_kal = " + kalman.z_kal);
            System.out.println("[z_dot_kal = " + kalman.z_dot_kal);
            System.out.println("[z_command = " + z_command);
            System.out.println("[depth error = " + (kalman.z_kal - z_command));
            System.out.println("[Integral Depth Error = " + depth_error_integral);
```

```
    }
    vehicle.set_auv_bow_vertical(-control_coefficients.get_k_thruster_z() *
        (kalman.z_kal - z_command) -
        control_coefficients.get_k_thruster_w() * kalman.z_dot_kal - 5.0 *
        depth_error_integral);

    vehicle.set_auv_stern_vertical(vehicle.get_auv_bow_vertical());

    if (flags.TRACE && flags.DISPLAYSCREEN)
        System.out.println("[Pre Pitch Thruster Values: " +
        vehicle.get_auv_bow_vertical());
    // include pitch control when hovering or tracking a target
    // requires reverification in water, apparent sign error problem...

    if ((flags.HOVERCONTROL) || (flags.TARGETCONTROL) || (flags.RECOVERYCONTROL)) {

        double AUV_bow_vertical = vehicle.get_auv_bow_vertical() +
            (-control_coefficients.get_k_thruster_theta() * (vehicle.get_theta()
            - theta_command) - control_coefficients.get_k_thruster_theta() *
            vehicle.get_q() * (2.0));

        double AUV_stern_vertical = vehicle.get_auv_bow_vertical() +
            (control_coefficients.get_k_thruster_theta() * (vehicle.get_theta()
            - theta_command) + control_coefficients.get_k_thruster_theta() *
            vehicle.get_q() * (2.0));

        vehicle.set_auv_bow_vertical(AUV_bow_vertical);
        vehicle.set_auv_stern_vertical(AUV_stern_vertical);
    }
    return;
} // end of public void compute_vertical_thrusters (

/**
 *  Computes fin controls
 *  @param:      None
 *  @return:     None
 */
public void compute_fin_controls() { // VERIFIED
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("[begin compute_fin_controls]\n");
    }
    // Report Stable Course to Tactical Level if Required
    if ((flags.REPORTSTABLE) && (Math.abs(Data_Processing.normalize2(
        vehicle.get_psi() - psi_command)) < 2.5)) { // degrees */ ))
        if ((flags.TACTICAL) && (flags.HOVERCONTROL == false) &&
            (flags.WAYPOINTCONTROL == false)) {
            flags.REPORTSTABLE = false;
            network.write_to_tactical("STABLE COURSE");
            if (flags.DISPLAYSCREEN)
                System.out.println("\nSTABLE COURSE\n");
        }
    }
    // Simplified PD rudders/planes control rules: - - - - - - - - - - - - -
    // calculate rudders - - - - - - - - - - - - - - - - - - - - - - - - - -
    vehicle.set_delta_rudder(control_coefficients.get_k_psi() *
        Data_Processing.normalize2(vehicle.get_psi() - psi_command)
```

```
    + (control_coefficients.get_k_r() * vehicle.get_r()) +
    (control_coefficients.get_k_v() * vehicle.get_v()));

  if (flags.SLIDINGMODECOURSE) {
    control_coefficients.set_sigma(control_coefficients.get_k_sigma_r() *
        vehicle.get_r() + control_coefficients.get_k_sigma_psi() *
        Data_Processing.normalize2(vehicle.get_psi() - psi_command));

    vehicle.set_delta_rudder((3.1403 * vehicle.get_r()) + 81.9712 *
        control_coefficients.get_eta_steering() *
        Data_Processing.dtanh(control_coefficients.get_sigma()));
    if (flags.DISPLAYSCREEN) {
        System.out.println("SLIDINGMODECOURSE sigma = " +
            control_coefficients.get_sigma() + ", delta_rudder = " +
            vehicle.get_delta_rudder() + "\n");
    }
  }
  // reduce ordered rudder if excessive roll occurs, may work for many UUVs
  vehicle.set_delta_rudder(vehicle.get_delta_rudder() *
    Math.cos(vehicle.get_phi()) * Math.cos(vehicle.get_phi()));

  if (flags.DISPLAYSCREEN && (Math.pow(Math.cos(vehicle.get_phi()), 2.0) < 0.98)
    && (flags.DEADSTICKRUDDER == false) &&
    (flags.DEADSTICKPLANES == false)) {
        System.out.println("\nrudder/planes reduction factor due to roll phi = "
            + (Math.pow(Math.cos(vehicle.get_phi()), 2.0)));
  }
  // calculate planes - - - - - - - - - - - - - - - - - - - - - - - - - - -
  vehicle.set_delta_planes_stern((control_coefficients.get_k_z() * depth_error) +
    (control_coefficients.get_k_theta() * vehicle.get_theta()) +
    (control_coefficients.get_k_q() * vehicle.get_q()) -
    (control_coefficients.get_k_w() * kalman.z_dot_kal));

  vehicle.set_delta_planes_bow(-vehicle.get_delta_planes_stern());

  if (flags.TRACE) {
    System.out.println("delta_planes_stern = " +
        vehicle.get_delta_planes_stern() + "\n");
    System.out.println("delta_planes_bow  = " +
        vehicle.get_delta_planes_bow() + "\n");
    System.out.println("depth_error = " + depth_error + ", product = " +
        control_coefficients.get_k_z() * depth_error + "\n");
    System.out.println("theta = " + vehicle.get_theta() +
        ", product = " + control_coefficients.get_k_theta() *
        vehicle.get_theta() + "\n");
    System.out.println("q = " + vehicle.get_q() + ", product = " +
        control_coefficients.get_k_q() *
        vehicle.get_q() + "\n");
    System.out.println("z_command = " + z_command + ", z_kal = " + kalman.z_kal + "\n");
    System.out.println("z_dot_kal = " + kalman.z_dot_kal + ", product = "
        + -control_coefficients.get_k_w() *
        kalman.z_dot_kal + "\n");
  }
  // temporary fix to incorrect delta_planes polarity in boat
  if (flags.LOCATIONLAB == false) {
    if (flags.DISPLAYSCREEN) {
```

```
            System.out.println(" [reversing polarity delta_planes_stern/bow, delta_rudder]");
        }
        vehicle.set_delta_planes_stern(-vehicle.get_delta_planes_stern());
        vehicle.set_delta_planes_bow(-vehicle.get_delta_planes_bow());
        vehicle.set_delta_rudder(-vehicle.get_delta_rudder());
    }
    // reduce ordered planes if excessive roll occurs, may work for many UUVs
    vehicle.set_delta_planes_stern(vehicle.get_delta_planes_stern() *
        Math.pow(Math.cos(vehicle.get_phi()), 2.0));
    vehicle.set_delta_planes_bow(vehicle.get_delta_planes_bow() *
        Math.pow(Math.cos(vehicle.get_phi()), 2.0));
    // Dead stick means no open loop control of rudders/planes - - - - - - -
    if (flags.DEADSTICKRUDDER) {
        vehicle.set_delta_rudder(rudder_command);
    }
    if (flags.DEADSTICKPLANES) {
        vehicle.set_delta_planes_stern(planes_command_stern);
        vehicle.set_delta_planes_bow(planes_command_bow);
    }
    // constrain plane/rudder orders +/- MAX_RUDDER degrees, don't normalize!
    delta_rudder = Data_Processing.clamp(delta_rudder, -MAX_RUDDER, MAX_RUDDER,
        "delta_rudder"); // degrees
    if (Math.abs(rpm) < 0.6 * MAX_RPM) { // low    speed plane limits
        vehicle.set_delta_planes_stern(Data_Processing.clamp
            (vehicle.get_delta_planes_stern(), -MAX_PLANE, MAX_PLANE,
            "delta_planes_stern"));
        vehicle.set_delta_planes_bow(Data_Processing.clamp
            (vehicle.get_delta_planes_bow(), -MAX_PLANE, MAX_PLANE, "delta_planes_bow"));
    }
    else if (Math.abs(rpm) < 0.7 * MAX_RPM) { // medium speed plane limits
        vehicle.set_delta_planes_stern(Data_Processing.clamp
            (vehicle.get_delta_planes_stern(), -MAX_PLANE / 1.5,
            MAX_PLANE / 1.5, "delta_planes_stern"));
        vehicle.set_delta_planes_bow(Data_Processing.clamp
            (vehicle.get_delta_planes_bow(), -MAX_PLANE / 1.5,
            MAX_PLANE / 1.5, "delta_planes_bow"));
    }
    else if (Math.abs(rpm) < MAX_RPM) { // high speed plane limits
        vehicle.set_delta_planes_stern(Data_Processing.clamp
            (vehicle.get_delta_planes_stern(), -10.0, -10.0, "delta_planes_stern"));
        vehicle.set_delta_planes_bow(Data_Processing.clamp
            (vehicle.get_delta_planes_bow(), -10.0, -10.0, "delta_planes_bow"));
    }
    else { // max+++  speed plane limits
        vehicle.set_delta_planes_stern(Data_Processing.clamp
            (vehicle.get_delta_planes_stern(), -5.0, 5.0, "delta_planes_stern"));
        vehicle.set_delta_planes_bow(Data_Processing.clamp(
            vehicle.get_delta_planes_bow(), -5.0, 5.0, "delta_planes_bow"));
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("[end compute_fin_controls]\n");
    }
    return;
} // end compute_fin_controls ()

/**
```

```
 *  Computes followlight control
 *  @param:      None
 *  @return:     None
 */
void compute_followlight_controls() {
   if ((flags.TRACE) && flags.DISPLAYSCREEN) {
      System.out.println("\n[begin compute_followlight_controls]\n");
   }
   if (firstLoopInitialized == false) {
      previous_psi = vehicle.get_psi();
      psiLightReference = vehicle.get_psi();
      psiStartLookAround = vehicle.get_psi();
      firstLoopInitialized = true;
      xLookAround = vehicle.get_x();
      yLookAround = vehicle.get_y();
      zLookAround = vehicle.get_z();
   }
   if (look_around_in_progress) {
      if (flags.DISPLAYSCREEN) {
         System.out.println("\n[LOOK-AROUND]\n");
      }
      flags.DEADSTICKPLANES = true;
      rotate_command = 12.0;
      compute_rotate_controls();
      lateral_command = 0.0;
      x_command = xLookAround;
      y_command = yLookAround;
      z_command = zLookAround;
      if (flags.DISPLAYSCREEN) {
         System.out.println("\n[Ray-Tracing]\n");
      }
      //**********************************
      //What is this and where is it defined
      //*************************************************
      // RenderCalculation ( vehicle.get_x(), vehicle.get_y(),
      //                  vehicle.get_z(), vehicle.get_psi()
      //                                                    , vehicle.get_theta(), vehicle.get_phi() );
      //***********************************
      //What is this and where is it defined
      //*************************************************
      //                           psiLightReference = CalculatePsiLightReference();
      // !!! try normalize instead of fabs:
      if ((Math.abs(previous_psi - vehicle.get_psi())) > 180.0) {
         ZeroLookAroundFlag = true;
      }
      if ((ZeroLookAroundFlag == true) &&
         (vehicle.get_psi() > psiStartLookAround)) {
         look_around_in_progress = false;
      }
      previous_psi = vehicle.get_psi();
   }
   if ((!look_around_in_progress) && (heading_reference == false)) {
      if (flags.DISPLAYSCREEN) {
         System.out.println("\n[HEADING REFERENCE]\n");
      }
      flags.REPORTSTABLE = true;
```

```
        time_int_control_on = vehicle.get_time() + 10.0; // give PD control 10 seconds
        flags.ST725SCANMODE = flags.SONARSCANSWATH; // Forward Scan
        flags.DEADSTICKRUDDER = true;
        flags.DEATH_SPIRAL_RESET = true;
        rudder_command = 0.0;
        x_command = xLookAround;
        y_command = yLookAround;
        z_command = zLookAround;
        psi_command = Data_Processing.normalize(
           Data_Processing.normalize2(psiLightReference));
        psi_command_hover = psi_command;
        compute_hover_controls();
        if (flags.DISPLAYSCREEN) {
           System.out.println("\n[Ray-Tracing]\n");
        }
        // RenderCalculation ( vehicle.get_x(), vehicle.get_y(), vehicle.get_z(),
        //                                                  vehicle.get_psi(),
        //                                                  vehicle.get_theta(), vehicle.get_phi() );
        //************************
        // WHAT IS THIS
        //************************
        //                        SaveImage();

        if ((Math.floor(10 * vehicle.get_psi())) == (Math.floor(10 * psiLightReference))) {
           heading_reference = true; //vehicle is at its heading of reference
        }
     }
     if ((!look_around_in_progress) && (heading_reference == true)) {
        if (flags.DISPLAYSCREEN) {
           System.out.println("\n[Light-Following]\n");
           System.out.println("\n[Ray-Tracing\\n");
        }
        // RenderCalculation( vehicle.get_x(), vehicle.get_y(), vehicle.get_z(),
        //                                                  vehicle.get_psi(),
        //                                                  vehicle.get_theta(), vehicle.get_phi() );

/*                      distanceAuvLight = Math.sqrt (( vehicle.get_x() -

           Light_Source_center_Z * FEET_PER_METER )
                                                                                                  * (
vehicle.get_x()
                        - Light_Source_center_Z * FEET_PER_METER )
                                                                                                  + (
vehicle.get_y()
                        - Light_Source_center_X * FEET_PER_METER )
                                                                                                  * (
vehicle.get_y()
                        - Light_Source_center_X * FEET_PER_METER )
                                                                                                  + (
vehicle.get_z()
                        + Light_Source_center_Y * FEET_PER_METER )
                                                                                                  * (
vehicle.get_z()
                        + Light_Source_center_Y * FEET_PER_METER ));

                        System.out.println ( "distanceAuvLight = " + distanceAuvLight + "\n" );
```

```
                    if ( CalculateCspeedDpsiDz ( &CSpeed, &DeltaPsi, &DeltaZ )
                                                    && ( distanceAuvLight < 6 )) {

        flags.FOLLOWLIGHTCONTROL = false;

                    if ( flags.DISPLAYSCREEN ) {
                                System.out.println ( "\n[end followlight ]" );
                    }
                    }

*/

        flags.DEADSTICKPLANES = true;
        flags.DEADSTICKRUDDER = false;
        flags.TARGETPOINTING = false;
        flags.REPORTSTABLE = true;
        port_rpm_command = 400 * CSpeed;
        stbd_rpm_command = 400 * CSpeed;
        time_int_control_on = vehicle.get_time() + 10.0; // give PD control 10 seconds
        z_command = vehicle.get_z() + DeltaZ;
        compute_lateral_thrusters();
        psi_command = Data_Processing.normalize((Data_Processing.normalize2(
           vehicle.get_psi() + DeltaPsi)));
        psi_command_hover = psi_command;
        rotate_command = 0.0;
        lateral_command = 0.0;
      }
      time_next_command = vehicle.get_time() + 2.0 * dt;
      if ((flags.TRACE) && flags.DISPLAYSCREEN) {
        System.out.println("[complete compute_followlight]\n");
      }
   } // end compute_followlight_controls ()

   /**
    *  Computes port speed control
    *  @param:      Double
    *  @return:     Int
    */

/* The following four functions were added on 12 Dec 95 */

/* They are from Dave Marco's execution code and are used */

/* for speed control of the port propellors */

   public int port_speed_control(double n_com) { // VERIFIED
      Km_ls = 0.6589;
      eta_ls = 10.0;
      phi_ls = 5.0;
      if (Math.abs(n_com) < 0.25) {
        Int_ls = 0.0;
      }
      e_n = n_com - sense.read_port_motor_rpm() / 60.0;
      Int_ls = Int_ls + Data_Processing.dtanh(e_n / phi_ls) * dt;
      v_ls_spc = (1.0 / Km_ls) * (n_com + eta_ls * Int_ls);
```

```
    v_dls = (int)((1023.0 / 48.0) * (v_ls_spc) + 511.5);
    if (v_dls < 0) {
        v_dls = 0;
    }
    if (v_dls > 1023) {
        v_dls = 1023;
    }
    return (v_dls);
}

/* end port_speed_control () */

/**
 *  Computes stbd speed control
 *  @param:      Double
 *  @return:     Int
 */
public int stbd_speed_control(double n_com) { // VERIFIED
    Km_rs = 0.6156;
    eta_rs = 10.0;
    phi_rs = 5.0;
    if (Math.abs(n_com) < 0.25) {
        Int_rs = 0.0;
    }
    e_n = n_com - sense.read_stbd_motor_rpm() / 60.0;
    Int_rs = Int_rs + Data_Processing.dtanh(e_n / phi_rs) * dt;
    v_rs_spc = (1.0 / Km_rs) * (n_com + eta_rs * Int_rs);
    v_drs = (int)((1023.0 / 48.0) * (v_rs_spc) + 511.5);
    if (v_drs < 0) {
        v_drs = 0;
    }
    if (v_drs > 1023) {
        v_drs = 1023;
    }
    return (v_drs);
} // end stbd_speed_control ()

/**
 *  @param:      None
 *  @return:     None
 */
void updateCrossBodyArray() {
    int ix;
    switch (cross_body_control_mode) {
            /* This Mode does not use data */

        case 0:
            for (ix = cross_sections - 1; ix >= 0; ix--) {
                cross_body_flow_u[ix] = 0.0;
                cross_body_flow_v[ix] = 0.0;
            }
            break;

                        /* This Mode uses raw data at front thrusters only */

        case 1:
```

```
            for (ix = cross_sections - 1; ix > 0; ix--) {
               cross_body_flow_u[ix] = 0.0;
               cross_body_flow_v[ix] = 0.0;
            } // THESE COMES FROM DYNAMICS HOW DO I GET THEM??
            cross_body_flow_u[1] = vehicle.get_doppler_stw_u() -
               vehicle.get_doppler_sog_u();
            cross_body_flow_v[1] = vehicle.get_doppler_stw_v() -
               vehicle.get_doppler_sog_v();
            break;
            // This Mode uses raw data smartly, array is updated for fwd progress
            // but we need to improve it to correspond to timestep and dead-reckon
            // progress in the x direction...
         case 2:
            for (ix = cross_sections - 1; ix > 0; ix--) {
               cross_body_flow_u[ix] = cross_body_flow_u[ix - 1];
               cross_body_flow_v[ix] = cross_body_flow_v[ix - 1];
            }
            cross_body_flow_u[0] = vehicle.get_doppler_stw_u() -
               vehicle.get_doppler_sog_u();
            cross_body_flow_v[0] = vehicle.get_doppler_stw_v() -
               vehicle.get_doppler_sog_v();
            break;
         default:
/* Do Nothing with Data */
            break;
      }
      return;
   }

   /**
    *  Detect death spiral
    *  @param:      None
    *  @return:     Boolean.  True if death spiral detected, False otherwise.
    */
   boolean detect_death_spiral() {
      double turn_direction = 0.0;
      double psi_old = 0.0;
      double cumulative_turn = 0.0;
      if (flags.TRACE && flags.DISPLAYSCREEN)
         System.out.println("[start detect_death_spiral ()]\n");
      // reset static variables; don't check for spiral
      if (flags.DEATH_SPIRAL_RESET) {
         turn_direction = 0;
         cumulative_turn = 0.0;
         flags.DEATH_SPIRAL_RESET = false;
         if (flags.TRACE && flags.DISPLAYSCREEN)
            System.out.println("[finish detect_death_spiral ()]\n");
         return false;
      }
      // Turn direction changed, reset static variables, or not at depth yet
      if ((Data_Processing.dsign(vehicle.get_psi_dot()) != turn_direction) ||
         (turn_direction == 0) ||
         (Math.abs(depth_error) > control_coefficients.get_standoff_distance())) {

            turn_direction = Data_Processing.dsign(vehicle.get_psi_dot());
            cumulative_turn = 0.0;
```

```
        psi_old = vehicle.get_psi();
        if (flags.TRACE && flags.DISPLAYSCREEN)
           System.out.println("[finish detect_death_spiral ()]\n");
        return false;
    }
    // Same turn direction, check for full circle
    cumulative_turn += Data_Processing.normalize2(vehicle.get_psi() - psi_old);
    psi_old = vehicle.get_psi();
    // Full 360 Degree Constant Direction Turn Means Death Spiral
    if ((cumulative_turn >= 360) || (cumulative_turn <= -360)) {
       if (flags.TRACE && flags.DISPLAYSCREEN) {
          System.out.println("[Death Spiral Detected]\n");
          System.out.println("[finish detect_death_spiral ()]\n");
       }
       return true;
    }
    // No Spiral Detected
    if (flags.TRACE && flags.DISPLAYSCREEN)
       System.out.println("[finish detect_death_spiral ()]\n");

    return false;
}

/* end int detect_death_spiral () */

/**
 *  Control loop method. It is responsible for the continuous
 *  sense-decide-act loop at a rate of 10 Hz.
 *  @param:      ST1000_sonar object, ST725_sonar object, Act object,
 *                                Network_Connection object, Parser object
 *  @return: None
 */
public void control_loop(ST1000_sonar st1000, ST725_sonar st725, Act act,
    Network_Connection network, Parser parser) { // executed each time step

    double volts_per_dac = 0.046875;
    double lateralMult; // multiple for lateral thruster voltage
    int dt_range1, dt_range2;
    if (flags.TRACE && flags.DISPLAYSCREEN) {
       System.out.println("[start closed_loop_control_module]\n");
    }
    // shutdown checks
    if ((flags.LOCATIONLAB == false) && (flags.BENCHTEST == false) &&
       (flags.HALTSCRIPT == false)) {

       System.out.println("Start checking the systems");

     if (( computer_voltage = sense.read_computer_battery_voltage() ) < 20.0) {

          flags.HALTSCRIPT = true;
          if ( flags.DISPLAYSCREEN ) {
                    System.out.println ( "Low Computer Voltage Detected: " +
             computer_voltage + " \n" );
                    }
          }
          if (( motor_voltage = sense.read_motor_gyro_battery_voltage() ) < 20.0) {
```

```
        flags.HALTSCRIPT = true;
        if ( flags.DISPLAYSCREEN )
                System.out.println ( "Low Motor Voltage Detected: " + motor_voltage + "\n" );
        }
        if ( sense.leak_check () ) {

        flags.HALTSCRIPT = true;
        if ( flags.DISPLAYSCREEN ) System.out.println ( "Leak Detected\n" );
        }
        if ( kalman.z_kal > 6.0 ) {

        flags.HALTSCRIPT = true;
        if ( flags.DISPLAYSCREEN ) System.out.println ( "Depth Exceeded\n" );
        }
        if ( flags.DIVETRACKER && ( dt_time + 30.0 <= vehicle.get_time() )) {

        flags.HALTSCRIPT = true;
        if ( flags.DISPLAYSCREEN )
                System.out.println ( "Loss of Dive Tracker for 30 Seconds\n" );
        }
        if (( flags.ST1000INSTALLED ) &&
        ( flags.ST1000SCANMODE == flags.SONARSCANSWATH ) &&
        ( ST1000_range_kal > 0.0 ) && ( ST1000_range_kal < 3.0 )) {

        flags.HALTSCRIPT = true;
        if ( flags.DISPLAYSCREEN )
                System.out.println ( "Collision Avoidance Invoked: ST1000_range_kal = " +
                                                        ST1000_range_kal + "\n" );
                }
                }


    System.out.println("Finish checking systems");

  if (flags.HALTSCRIPT) {
    System.out.println("Execute shutdown procedure");
    execute_shutdown_script();
  }
  // Read Sensors and Communicate with Virtual World //*******************
  st1000.control_ST1000_sonar();
  st725.control_ST725_sonar();
  vehicle.set_speed(sense.read_speed());
  rpm = ((port_rpm_command + stbd_rpm_command) / 2.0);
  rpm = (Data_Processing.clamp(rpm, MAX_RPM, -MAX_RPM, "rpm"));
  if (flags.TRACE && flags.DISPLAYSCREEN) {
    System.out.println("[clamp (& rpm, MAX_RPM, -MAX_RPM, \"rpm\") complete]\n");
  }
  // Main_Motor RPM Control //*********************************************
  // note thruster use does not preclude propeller use
  if (flags.LOCATIONLAB) { // rpm model assumes instantaneous response
    vehicle.set_port_rpm(port_rpm_command);
    vehicle.set_stbd_rpm(stbd_rpm_command);
  }
  else { // in water => propeller rpms are controlled so read actual value
    vehicle.set_port_rpm(sense.read_port_motor_rpm());
```

```
   vehicle.set_stbd_rpm(sense.read_stbd_motor_rpm());
}
// if using virtual world dynamics, network is source of values <<<<<<<<
vehicle.set_phi(sense.read_roll_angle()); // read roll angle
cos_phi = Math.cos(Math.toRadians(vehicle.get_phi()));
vehicle.set_theta(sense.read_pitch_angle()); // read pitch angle
vehicle.set_psi(sense.read_psi()); // Read psi/heading
sin_psi = Math.sin(Math.toRadians(vehicle.get_psi()));
cos_psi = Math.cos(Math.toRadians(vehicle.get_psi()));
vehicle.set_p(sense.read_roll_rate_gyro()); // read roll rate
vehicle.set_q(sense.read_pitch_rate_gyro()); // read pitch rate
vehicle.set_r(Data_Processing.normalize2(vehicle.get_psi() - psi_im1) / dt);
psi_im1 = vehicle.get_psi();
//     r    = read_yaw_rate_gyro   ();    // Read yaw rate
vehicle.set_z(sense.read_depth()); // Read depth
kalman.kalman_z(vehicle.get_z());
if (flags.TRACE && flags.DISPLAYSCREEN) {
   System.out.println("[z = " + vehicle.get_z() + ", z_kal = " +
      kalman.z_kal + "]\n");
}
if (Math.abs(kalman.z_kal) < 0.0001) {
   kalman.z_kal = 0.0;
}
if (Math.abs(kalman.z_dot_kal) < 0.0001) {
   kalman.z_dot_kal = 0.0;
}
if (Math.abs(kalman.z_ddot_kal) < 0.0001) {
   kalman.z_ddot_kal = 0.0;
}
vehicle.set_z_dot(kalman.z_dot_kal);
vehicle.set_w(kalman.z_dot_kal); // look out!! <<<<
// note:  in laboratory using virtual world, sensed values are superceded
// Update Cross-Body flow sensor array
if (flags.LOCATIONLAB) {
   updateCrossBodyArray();
}
// estimate X and Y with Mathematical Model or Dead Reckoning
if (!flags.LOCATIONLAB) { // in-water, perform a valid dead reckon
   XY_model_est((v_dls - 512) * volts_per_dac, (v_drs - 512) *
      volts_per_dac, vehicle.get_auv_bow_lateral(),
      vehicle.get_auv_stern_lateral(), auv_oceancurrent_x, auv_oceancurrent_y, true);
}
else { // virtual world providing sensor inputs
   vehicle.set_x(vehicle.get_x() + (vehicle.get_speed() * dt * cos_psi));
   if (Math.abs(vehicle.get_x()) <= 0.0001) {
      vehicle.set_x(0.0);
   }
   vehicle.set_y(vehicle.get_y() + (vehicle.get_speed() * dt * sin_psi));
   if (Math.abs(vehicle.get_y()) <= 0.0001) {
      vehicle.set_y(0.0);
   }
   vehicle.set_x(vehicle.get_x() + auv_oceancurrent_x * dt);
   vehicle.set_y(vehicle.get_y() + auv_oceancurrent_y * dt);
}
if (flags.TRACE && flags.DISPLAYSCREEN) {
   Fmt.printf("[AUV_oceancurrent_x = %3.1f,", auv_oceancurrent_x);
```

```
        Fmt.printf(" AUV_oceancurrent_y = %3.1f,", auv_oceancurrent_y);
        Fmt.printf(" AUV_oceancurrent_z = %3.1f]\n", auv_oceancurrent_z);
      }
      // Control laws ****  NOTE:  all k_ constants  must be (+) positive ****
      //#ifdef os9
      // Update Dive Tracker Ranges

/*    if ( flags.DIVETRACKER && ( CLReaddmod ( &dt_range1, &dt_range2 ) == NEW_DATA )
      && ( dt_range1 < 10000 ) && ( dt_range2 < 10000 ) &&
      ( dt_range1 > 0 ) && ( dt_range2 > 0 )) {

      divetracker_range1 = (double) dt_range1 / 12.0;
      divetracker_range2 = (double) dt_range2 / 12.0;
      dt_time = vehicle.get_time();
      if ((TRACE) && (DISPLAYSCREEN))
        printf("Divetracker Ranges:  %f %f %f\n",t,divetracker_range1,divetracker_range2);
    }
    else
    {
      if (z_kal <= 2.0) dt_time = vehicle.get_time();
    }
*/

      //#endif

      waypoint_distance = Math.sqrt((vehicle.get_x() - x_command) *
        (vehicle.get_x() - x_command) +
        (vehicle.get_y() - y_command) * (vehicle.get_y() - y_command));

      /// calculate depth error OK prior to death spiral check
      depth_error = (z_command - kalman.z_kal);
      // constrain depth_error to +- 15.0 feet to prevent going vertical
      //      and enable stable pitch angle even on large depth changes
      depth_error = Data_Processing.clamp(depth_error, -15.0, 15.0, "depth_error");
      // Zero thruster commands
      vehicle.set_auv_bow_vertical(0.0);
      vehicle.set_auv_stern_vertical(0.0);
      vehicle.set_auv_bow_lateral(0.0);
      vehicle.set_auv_stern_lateral(0.0);
      vehicle.set_delta_rudder(0.0);
      vehicle.set_delta_planes_stern(0.0);
      vehicle.set_delta_planes_bow(0.0);

      // Recompute new thruster commands depending on control mode *************
      if (flags.HOVERCONTROL) {
        compute_hover_controls();
      }
      else if (flags.FOLLOWLIGHTCONTROL) {
        compute_followlight_controls();
      }
      else if (flags.DOCKINGCONTROL) {
        compute_docking_controls();
      }
      else if (flags.TARGETCONTROL) {
        compute_target_controls();
      }
```

```
else if (flags.WAYPOINTCONTROL) {
   compute_waypoint_controls();
}
else if (flags.LATERALCONTROL) {
   compute_lateral_controls();
}
else if (flags.ROTATECONTROL) {
   compute_rotate_controls ();
}
else if (flags.RECOVERYCONTROL) {
   compute_recovery_controls();
}
else if (flags.THRUSTERCONTROL) {
   compute_lateral_thrusters();
}
if (flags.TRACE && flags.DISPLAYSCREEN) {
   System.out.println("Pre-sqrt thruster control calculated values:\n");
   Fmt.printf("AUV_bow_vertical   = %6.3f\n", vehicle.get_auv_bow_vertical());
   Fmt.printf("AUV_stern_vertical = %6.3f\n", vehicle.get_auv_stern_vertical());
   Fmt.printf("AUV_bow_lateral    = %6.3f\n", vehicle.get_auv_bow_lateral());
   Fmt.printf("AUV_stern_lateral  = %6.3f\n", vehicle.get_auv_stern_lateral());
}
// convert to signed sqrt to account for volts-to-thrust relationship
// different multiple required between lab and auv because of polarity
// discrepancy between virtual world and actual auv   <<< FIX! <<<<
if (flags.LOCATIONLAB) {
   lateralMult = 2.0;
}
else {
   lateralMult = -2.0;
}
vehicle.set_auv_bow_vertical(4.8989 *
   Data_Processing.dsign(vehicle.get_auv_bow_vertical()) *
   Math.sqrt(Math.abs(vehicle.get_auv_bow_vertical())));
vehicle.set_auv_stern_vertical(4.8989 *
   Data_Processing.dsign(vehicle.get_auv_stern_vertical()) *
   Math.sqrt(Math.abs(vehicle.get_auv_stern_vertical())));
vehicle.set_auv_bow_lateral(lateralMult * 2.449 *
   Data_Processing.dsign(vehicle.get_auv_bow_lateral()) *
   Math.sqrt(Math.abs(vehicle.get_auv_bow_lateral())));
vehicle.set_auv_stern_lateral(lateralMult * 2.449 *
   Data_Processing.dsign(vehicle.get_auv_stern_lateral()) *
   Math.sqrt(Math.abs(vehicle.get_auv_stern_lateral())));
if (flags.TRACE && flags.DISPLAYSCREEN) {
   System.out.println("Post-sqrt thruster control calculated values:\n");
   Fmt.printf("AUV_bow_vertical   = %6.3f\n", vehicle.get_auv_bow_vertical());
   Fmt.printf("AUV_stern_vertical = %6.3f\n", vehicle.get_auv_stern_vertical());
   Fmt.printf("AUV_bow_lateral    = %6.3f\n", vehicle.get_auv_bow_lateral());
   Fmt.printf("AUV_stern_lateral  = %6.3f\n", vehicle.get_auv_stern_lateral());
}
// fins reset except in TARGETCONTROL and RECOVERYCONTROL modes
if ((flags.TARGETCONTROL == false) && (flags.RECOVERYCONTROL == false)) {
   compute_fin_controls();
}
// constrain thruster orders  +/- 24.0 volts == 3820 rpm no-load
// constrain propeller orders +/- 700 rpm no-load
```

```
        vehicle.set_auv_bow_vertical(Data_Processing.clamp(
           vehicle.get_auv_bow_vertical(), -24.0, 24.0, "AUV_bow_vertical"));

        vehicle.set_auv_stern_vertical(Data_Processing.clamp
           (vehicle.get_auv_stern_vertical(), -24.0, 24.0, "AUV_stern_vertical"));

        vehicle.set_auv_bow_lateral(Data_Processing.clamp(
           vehicle.get_auv_bow_lateral(), -24.0, 24.0, "AUV_bow_lateral"));

        vehicle.set_auv_stern_lateral(Data_Processing.clamp
           (vehicle.get_auv_stern_lateral(), -24.0, 24.0, "AUV_stern_lateral"));

        port_rpm_command = Data_Processing.clamp(
           port_rpm_command, -MAX_RPM, MAX_RPM, "port_rpm_command");

        stbd_rpm_command = Data_Processing.clamp(
           stbd_rpm_command, -MAX_RPM, MAX_RPM, "stbd_rpm_command");

        // Record Control Orders to Orders File
        if ((flags.NOSCRIPT == false) && ((flags.HOVERCONTROL) ||
           (flags.TARGETCONTROL) || (flags.WAYPOINTCONTROL) ||
           (flags.ROTATECONTROL) || (flags.THRUSTERCONTROL) ||
           (flags.FOLLOWLIGHTCONTROL))) {
              System.out.println(" Save to file");

/*      fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f %5.1f\n",
           t, psi_command, x_command, y_command, z_command,
           port_rpm_command, stbd_rpm_command,
           rudder_command,   planes_command_stern,
// insert planes_command_bow, and modify *.gnu plot scripts ****
           AUV_bow_vertical, AUV_stern_vertical,
           AUV_bow_lateral, AUV_stern_lateral);
 */
        }
        // command thruster and propellor orders
        act.command_motor(vehicle.get_auv_bow_vertical(), BOW_VERTICAL);
        act.command_motor(vehicle.get_auv_stern_vertical(), STERN_VERTICAL);
        act.command_motor(vehicle.get_auv_bow_lateral(), BOW_LATERAL);
        act.command_motor(vehicle.get_auv_stern_lateral(), STERN_LATERAL);
        act.command_motor(port_rpm_command, PORT_PROP);
        act.command_motor(stbd_rpm_command, STBD_PROP);

        // Send commands to rudders and planes ********************************
        act.command_rudder(delta_rudder);
        act.command_planes(vehicle.get_delta_planes_stern(), vehicle.get_delta_planes_bow());

        // send telemetry to tactical level and data recording files - - - - - -
        ////////////////////////////////////////////////////
        System.out.println("Save information");
        io.record_data(vehicle, network, parser);

        // read commands from tactical level - - - - - - - - - - - - - - - - - -
        // if (TACTICAL) read_parallel_port ();[old code] now uses socket
        // update simulation clock "t" - - - - - - - - - - - - - - - - - - - - -
        vehicle.set_time(vehicle.get_time() + dt);
```

```
      //     fflush (stdout); // force completion of screen write
      System.out.println("Update clock");
      //     currentloopclock = clock ();

/*
    if ( flags.TRACE && flags.REALTIME && flags.DISPLAYSCREEN ) {
       Fmt.println ( "[Unused Loop Time: %5.4f\n",
            (float) ( nextloopclock - currentloopclock ) / (float) CLOCKS_PER_SEC );
    }
    if (( flags.REALTIME ) &&( currentloopclock < nextloopclock )) {

        if ( flags.TRACE && flags.DISPLAYSCREEN ) {

          Fmt.println ( "currentloopclock = %ld, ", currentloopclock );
          Fmt.println ( "nextloopclock = %1d\n", nextloopclock );

             Fmt.println ("timestep dt = %5.3lf seconds (corresponding clock ticks = %d)\n",
                                dt, (int)(dt * (double) CLOCKS_PER_SEC));
          Fmt.println ( "Busy wait until system clock reaches simulation clock, ");
          Fmt.println ("loop duration = %5.3lf\n", ((double) currentloopclock -
                                (double) nextloopclock) / CLOCKS_PER_SEC);
        }
        while  ( currentloopclock < nextloopclock ) {

           currentloopclock = clock (); // %%%%% busy wait %%%%%
        }
        if ( flags.TRACE && flags.DISPLAYSCREEN ) {
        {
           Fmr.println ( "Busy wait complete, loop + wait duration = %5.3lf, ",
                 ( (double) currentloopclock - (double) nextloopclock )/ CLOCKS_PER_SEC);
           Fmt.println ( "current clock () = %ld\n", currentloopclock );
        }
    }
    else if (( flags.REALTIME == false ) && flags.LOCATIONLAB && flags.DISPLAYSCREEN &&
flags.TRACE ) {

           Fmt.println ("No busy wait, loop duration = %5.3lf, ",
                 ( (double) currentloopclock - (double) nextloopclock ) / CLOCKS_PER_SEC);
           Fmt.println ( "current clock () = %ld\n", currentloopclock );
        }

    nextloopclock = clock () + (int) ( dt * (double) CLOCKS_PER_SEC );
*/

      //   determine if closed_loop_control_module () is done; if so,  - - - -
      // repeat call to closed_loop_control_module () for each time step   - -
      // this needs to be 3-way logic file | tactical | keyboard <<<<<<<<<<<<<
      if (flags.REPLAY == true) {
         return;
      }
      else if ((flags.KEYBOARDINPUT) || (flags.TACTICAL)) {
         System.out.println("Parse command here");
         //     parse_mission_script_commands ();    // get next script orders read
         // ignore failure
         // we are reading from the mission script file in these cases
      }
```

```
/*     else if ( auvdatafile == NULL ) {

        end_test = true;            // file never opened, loop and open it //
    }
    else if ( flags.NOSCRIPT ) {  // scriptfile not yet closed, read more

        // ignore script, do not parse_mission_script_commands ();
    }
    else if ( feof (auvscriptfile) && (t > time_next_command)) {                  // all done

        if ( flags.TRACE &&flags. DISPLAYSCREEN )
              System.out.println ( "end_test NOSCRIPT == TRUE, set TRUE\n" );
        end_test = true;
    }
 */

      else if (vehicle.get_time() > time_next_command) { // scriptfile not yet closed, read more
         if (flags.TRACE && flags.DISPLAYSCREEN)
            System.out.println("\n[read more from parse_mission_script_commands]\n");
         //  parse_mission_script_commands ();    // get next script orders read
         // ignore failure
      }
      // else not done executing current script command, continue/don't block
      if (flags.TRACE && flags.DISPLAYSCREEN)
      //     System.out.println ( "\n[time_next_command = %5.1f]\n", time_next_command );
         System.out.println("Next command at time =");
      if (flags.TRACE && flags.DISPLAYSCREEN)
         System.out.println("\n[finish closed_loop_control_module ()]\n");
      if (end_test) {
         act.command_motor(0.0, BOW_VERTICAL);
         act.command_motor(0.0, STERN_VERTICAL);
         act.command_motor(0.0, BOW_LATERAL);
         act.command_motor(0.0, STERN_LATERAL);
         act.command_motor(0.0, PORT_PROP);
         act.command_motor(0.0, STBD_PROP);
      }
      return;
   } // end closed_loop_control_module ()

   // HOW DOES THE FOLLOWING METHOD WORKS???  Could be under ACT
/**
*
*/
   public void execute_shutdown_script() {
      int phase = 1;
      double time_next_phase = 0.0;
      if (flags.TRACE) {
         System.out.println("[Shutdown Script Entered]\n");
      }
      switch (phase) {
         case 1:
            if (flags.TRACE) {
               System.out.println("[Starting phase 1 of Shutdown Script]\n");
            }
            flags.THRUSTERCONTROL = true;
```

```
    flags.ROTATECONTROL = false;
    flags.LATERALCONTROL = false;
    flags.FOLLOWWAYPOINTMODE = false;
    flags.WAYPOINTCONTROL = false;
    flags.HOVERCONTROL = false;
    flags.GPSFIXINPROGRESS = false;
    x_command = vehicle.get_x();
    y_command = vehicle.get_y();
    z_command = vehicle.get_z();
    psi_command = vehicle.get_psi();
    rpm = 0;
    port_rpm_command = 0.0;
    stbd_rpm_command = 0.0;
    time_next_phase = vehicle.get_time() + 20.0;
    phase = 2;
    break;
  case 2:
    if (vehicle.get_time() >= time_next_phase) {
      if (flags.TRACE) {
        System.out.println("[Starting phase 2 of Shutdown Script]\n");
      }
      flags.THRUSTERCONTROL = false;
      time_next_phase = vehicle.get_time() + 1.0;
      phase = 3;
    }
    break;
  case 3:
    if (vehicle.get_time() >= time_next_phase) {
      if (flags.TRACE) {
        System.out.println("[Starting phase 3 of Shutdown Script]\n");
      }
      flags.LOOPFOREVER = false;
      System.out.println("SEND TO VIRTUAL WORLD 'KILL' ");
      if (flags.TRACE) {
        System.out.println("\n[end_test set TRUE]\n");
      }
      end_test = true;
      vehicle.set_x_dot(0.0);
      vehicle.set_y_dot(0.0);
      vehicle.set_z_dot(0.0);
      vehicle.set_phi_dot(0.0);
      vehicle.set_theta_dot(0.0);
      vehicle.set_psi_dot(0.0);
      vehicle.set_speed(0.0);
      vehicle.set_u(0.0);
      vehicle.set_v(0.0);
      vehicle.set_w(0.0);
      vehicle.set_p(0.0);
      vehicle.set_q(0.0);
      vehicle.set_r(0.0);
      vehicle.set_delta_planes_stern(0.0);
      vehicle.set_delta_planes_bow(0.0);
      vehicle.set_delta_rudder(0.0);
      vehicle.set_port_rpm(0.0);
      vehicle.set_stbd_rpm(0.0);
    }
```

```
            break;
        } // end of switch
    } // end of execute_shutdown_script()

    // Mathematical Model for Estimating X and Y
    public void XY_model_est(double v_ls, double v_rs, double v_blt, double v_slt, double X_dot_c,
        double Y_dot_c, boolean update_vel) {
            //   boolean  update_vel;
            double alpa_x = 0.0025, alpa_y = 0.004, b_x = 1.33, b_y = 17.0;
            double M_x = (435.0 + 43.5) / 32.2, M_y = (435.0 + 348.0) / 32.2;
            double f_ls, f_rs, f_blt, f_slt, F_x, F_y;
            double u_ddot, v_ddot, r_ddot;
            f_ls = alpa_x * (v_ls * v_ls) * Data_Processing.dsign(v_ls);
            f_rs = alpa_x * (v_rs * v_rs) * Data_Processing.dsign(v_rs);
            f_blt = alpa_y * (v_blt * v_blt) * Data_Processing.dsign(v_blt);
            f_slt = alpa_y * (v_slt * v_slt) * Data_Processing.dsign(v_slt);
            F_x = f_ls + f_rs;
            F_y = f_blt + f_slt;

                /* Use speed sensor OR mathematical model to estimate speed */

                /* depending on previous speed and speed sensor value      */

            u_ddot = (F_x - b_x * vehicle.get_u() * Math.abs(vehicle.get_u())) / M_x;
            if ((vehicle.get_u() > 0.2) && (vehicle.get_speed() >= 0.2)) {
                vehicle.set_u(vehicle.get_speed());
            }
            else if ((vehicle.get_u() < -0.2) && (vehicle.get_speed() >= 0.2)) {
                vehicle.set_speed(-vehicle.get_speed());
                vehicle.set_u(vehicle.get_speed());
            }
            else {
                vehicle.set_u(vehicle.get_u() + dt * (u_ddot));
                if (vehicle.get_u() > 0.24) vehicle.set_u(0.24);
                else if (vehicle.get_u() < -0.24) vehicle.set_u(-0.24);
                vehicle.set_speed(vehicle.get_u());
            }
            v_ddot = (F_y - b_y * vehicle.get_v() * Math.abs(vehicle.get_v())) / M_y;
            vehicle.set_v(vehicle.get_v() + dt * (v_ddot));
            if (!update_vel) {
                vehicle.set_u(vehicle.get_x_dot() * cos_psi + vehicle.get_y_dot() * sin_psi);
                vehicle.set_v(-vehicle.get_x_dot() * sin_psi + vehicle.get_y_dot() * cos_psi);
            }

                /* modify state vector values based on dead reckoning */

            vehicle.set_x_dot(vehicle.get_u() * cos_psi - vehicle.get_v() * sin_psi + X_dot_c);
            vehicle.set_y_dot(vehicle.get_u() * sin_psi + vehicle.get_v() * cos_psi + Y_dot_c);
            vehicle.set_x(vehicle.get_x() + dt * vehicle.get_x_dot());
            vehicle.set_y(vehicle.get_y() + dt * vehicle.get_y_dot());
            return;
    } // end of        public void XY_model_est ()
} // end of Control class
```

E. IO.JAVA

```
/* ____________________________________________________________________________________
  Title:          IO.java
  Description:    This class is responsible for all the input and output of
                  files and information.  It opens the the file where the
                  telemetry would be saved, the control constant coefficients
                  to be used, it creates the telemetry string and reads the
                  mission script.
  Date:           April 16, 2002
  Project:        Execution Java Software for Autonomous Underwater Vehicle,
                  Naval Postgraduate School, Monterey, CA
  Compiler:       JDK 1.3.1
  Author:         Miguel A. Ayala
  Version:        1.0
  ____________________________________________________________________________________
*/

package mil.navy.nps.auvAries.execution.input_output;

import mil.navy.nps.auvAries.execution.input_output.Commands_Queue;
import mil.navy.nps.auvAries.execution.vehicle.Vehicle;
import mil.navy.nps.auvAries.execution.control.Control_Coefficients;
import mil.navy.nps.auvAries.execution.globals.Execution_Flags;
import mil.navy.nps.auvAries.execution.network.Network_Connection;
import java.io.*;
import java.util.*;
import java.util.StringTokenizer;
import mil.navy.nps.auvAries.execution.decide.Parser;

/**
 *        This class is responsible for all the input and output of files and information.
 *  It opens the the file where the
 *  telemetry would be saved, the control constant coefficients
 *  to be used, it creates the telemetry string and reads the mission script.
 */
public class IO {
   /**  */
   public String AUVINFOFILENAME;

   /**  */
   public String AUVINITFILENAME;

   /**  */
   public String AUVSCRIPTFILENAME;

   /**  */
   public String AUVORDERSFILENAME;

   /**  */
   public String AUVDATAFILENAME;

   /**  */
   public String AUVTEXTFILENAME;
```

```
/**  */
public String ST1000DATAFILE;

/**  */
public String TARGETDATAFILE;

/**  */
//    public String AUVEMAILFILENAME;

/**  */
public String DEFAULTCONTROLCONSTANTSINPUTNAME;

/**  */
public String CONTROLCONSTANTSOUTPUTNAME;

/**  */
public String VIRTUAL_WORLD_REMOTE_HOST_NAME;

/**  */
public String TELEMETRY_FILE_NAME;

/**  */
int index;

/**  */
int numKeywords;

/**  */
FileInputStream fileIn;

/**  */
DataInputStream input;

/**  */
BufferedReader buffer;

/**  */
DataOutputStream output;

/**  */
PrintWriter fileOut;

/**  */
String commandLine;

/**  */
String valid_keywords1, valid_keywords2, valid_keywords3, valid_keywords4,
    valid_keywords5, valid_keywords6, valid_keywords7, valid_keywords8,
    valid_keywords9, valid_keywords10, valid_keywords11;

/**  */
private String[] vaild_keywords;

/**  */
private String telemetryString;
```

```
/**  */
private String telemetry_from_dynamics;

/**  */
private StringTokenizer tokens;

/**  */
private boolean read_another_line;

/**  */
private boolean comment1, comment2;

/**  */
private Control_Coefficients control_coefficients;

/**  */
private Execution_Flags flags;

/**
 *  Constructor
 *  @param:                 None
 *  @return:                None
 *  @exception:  None
 */
public IO(Control_Coefficients control_coefficientsRef, Execution_Flags flagsRef) {

   control_coefficients = control_coefficientsRef;
   flags = flagsRef;
   // FILES TO BE USED
   AUVINFOFILENAME = "mission.info";
   AUVINITFILENAME = "mission.init";
   AUVSCRIPTFILENAME = "mission.script";
   AUVORDERSFILENAME = "mission.output.orders";
   AUVDATAFILENAME = "mission.output.telemetry";
   AUVTEXTFILENAME = "mission.output.1_second";
   ST1000DATAFILE = "st1000datafile";
   TARGETDATAFILE = "targetdatafile";
   //       AUVEMAILFILENAME = "mission.output.email";
   DEFAULTCONTROLCONSTANTSINPUTNAME = "control.constants.input";
   CONTROLCONSTANTSOUTPUTNAME = "control.constants.output";
   TELEMETRY_FILE_NAME = null;
   index = 0;
   numKeywords = 0;
   commandLine = null;
   telemetryString = null;
   valid_keywords1 = "help, trace|notrace, loopforever|looponce, loop-forever|loop_once,";
   valid_keywords2 = "wait #, time #, time step (0.0 .. 5.0), mission,";
   valid_keywords3 = "keyboard|keyboard-off, quiet, kill,";
   valid_keywords4 = "rpm, course, depth, thrusters|thrusters-off, ";
   valid_keywords5 = "loopfilebackup, entercontrolconstants, rotate,";
   valid_keywords6 = "position|location|fix, orientation, gps|gps-fix,";
   valid_keywords7 = "gps-complete|gps-fix-complete, sonartrace|sonartraceoff,";
   valid_keywords8 = "sonarinstalled, trace|trace-off, parallelporttrace,";
   valid_keywords9 = "remotehoste hostname, realtime|nopause, pause,";
   valid_keywords10 = "controlconstantsfile, silent, e-mail|no-email,";
   valid_keywords11 = "waypoint, seastate (0-9)";
```

```
        read_another_line = true;
    }

    /**
     *  Print the valid keyword use as parameters.
     *  @param:                 None
     *  @return:                None
     *  @exception:  None
     */
    public void print_valid_keywords() {
        String[] valid_keywords = { valid_keywords1, valid_keywords2, valid_keywords3, valid_keywords4,
                                                                                    valid_keywords5,
valid_keywords6, valid_keywords7, valid_keywords8,
                                                                                    valid_keywords9,
valid_keywords10, valid_keywords11 };

        numKeywords = valid_keywords.length;
        for (index = 0; index < numKeywords; index++) {
            System.out.println("Number of keywords = " + valid_keywords[index]);
        }
    }

    /**
     *  Builds the header of the telemetry file
     *  @param:                 PrintWriter object that contains the file to be used
     *  @return:
     *  @exception:             IOException
     */
    public void build_telemtry_file_header(PrintWriter fileOut) {
        try {
            fileOut.println("#  auvdatafile  mission.output.telemetry  shows  42  state  vector  variables  at  0.10
second intervals.");
            fileOut.println("# state                                    paddle                                    phi
theta psi   delta  delta port  stbd  bow_  stern bow_  stern  _ST1000 sonar__     _ST725  sonar__    Dive
Dive");
             fileOut.println("# vector   t     x     y     z    phi   theta psi   speed u    v    w    p    q    r    x_dot
y_dot z_dot _dot  _dot  _dot  rudder plane rpm   rpm   vrtcl vrtcl latrl latrl  bng   range  dB    bng   range
dB   Trk1  Trk2 SOG_u SOG_v STW_u STW_v altitude ");
            fileOut.flush();
        }
        catch (Exception e) {
            System.out.println("Exception " + e + " occurred while building the telemetry file header");
        }
    }

    /**
     *  Save the Telemetry vector to a file
     *  @param:
     *  @return:
     *  @exception:   IOException
     */
    public void save_to_file(String telemetryString) {
        try {
            fileOut.println(telemetryString);
            fileOut.flush();
            System.out.println( "Saving to file " + telemetryString );
```

```
        //
    }
    catch (Exception e) {
        System.out.println("Exception " + e + " occurred while saving telemetry");
    }
}

/**
 *  Close opened files
 *  @param:
 *  @return:
 *  @exception:   IOException
 */
public void close_files() {
    fileOut.close();

}

public void open_telemetry_saved_file(String fileName) {
    try {
        fileOut = new PrintWriter(new BufferedWriter(new FileWriter(fileName)));
        build_telemtry_file_header(fileOut);
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("File " + fileName + " is open");
        }
    }
    catch (Exception e) {
        System.out.println("Exception " + e + " occurred while trying to open "
            + fileName);
    }
    return;
}

/**
 *  @param:
 *  @return:
 *  @exception:   IOException
 */
public BufferedReader open_File(String fileName) { // need to modify to create if no exist
    try {
        fileIn = new FileInputStream(fileName);
        input = new DataInputStream(fileIn);
        buffer = new BufferedReader(new InputStreamReader(fileIn));
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("File " + fileName + " is open");
        }
    }
    catch (Exception e) {
        System.out.println("Exception " + e + " occurred while trying to open "
            + fileName);
    }
    return buffer;
}

/**
```

```
 *  Opens the file specified which contains the mission commands. Reads the file and save the command
lines in a queue.
 *  @param:      An empty queue object.
 *  @return:     Modified queue with the command lines saved.
 *  @exception:   IOException
 */
public void readMissionScript(Commands_Queue queue, BufferedReader buffer) {
    try {
        commandLine = buffer.readLine();
        while (commandLine != null) {
            comment1 = commandLine.startsWith("#");
            comment2 = commandLine.startsWith("*");
            if (!comment1) {
                if (commandLine.length() != 0) {
                    queue.push(commandLine);
                }
                if (commandLine.compareToIgnoreCase("quit") == 0) {
                    read_another_line = false;
                }
            } // end of  if (!comment1){
            commandLine = buffer.readLine();
        } //  end of while ( )

        // CLOSE FILE, DONE USING IT
        fileIn.close();
    } // end of try
    catch (Exception e) {
        System.out.println("Exception " + e);
    }
}

/**
 *       To build the telemetry string for saving to the file
 *  @param:                    Vehicle object
 *  @return:      String, telemetry string
 */
public String build_telemetry_string_to_file(Vehicle aries) {
    // 42 state variables
    telemetryString = "auv_state" + " " +
        Double.toString(aries.get_time()) + " " +
        Double.toString(aries.get_x()) + " " +
        Double.toString(aries.get_y()) + " " +
        Double.toString(aries.get_z()) + " " +
        Double.toString(aries.get_phi()) + " " +
        Double.toString(aries.get_theta()) + " " +
        Double.toString(aries.get_psi()) + " " +
        Double.toString(aries.get_speed()) + " " +
        Double.toString(aries.get_u()) + " " +
        Double.toString(aries.get_v()) + " " +
        Double.toString(aries.get_w()) + " " +
        Double.toString(aries.get_p()) + " " +
        Double.toString(aries.get_q()) + " " +
        Double.toString(aries.get_r()) + " " +
        Double.toString(aries.get_x_dot()) + " " +
        Double.toString(aries.get_y_dot()) + " " +
        Double.toString(aries.get_z_dot()) + " " +
```

```
        Double.toString(aries.get_phi_dot()) + " " +
        Double.toString(aries.get_theta_dot()) + " " +
        Double.toString(aries.get_psi_dot()) + " " +
        Double.toString(aries.get_delta_rudder()) + " " +
        Double.toString(aries.get_delta_planes_stern()) + " " +
        // Double.toString( aries.get_delta_planes_bow() ) + " " +
        Double.toString(aries.get_port_rpm()) + " " +
        Double.toString(aries.get_stbd_rpm()) + " " +
        Double.toString(aries.get_auv_bow_vertical()) + " " +
        Double.toString(aries.get_auv_stern_vertical()) + " " +
        Double.toString(aries.get_auv_bow_lateral()) + " " +
        Double.toString(aries.get_auv_stern_lateral()) + " " +
        Double.toString(aries.get_auv_ST1000_bearing()) + " " +
        Double.toString(aries.get_auv_ST1000_range()) + " " +
        Double.toString(aries.get_auv_ST1000_strength()) + " " +
        Double.toString(aries.get_auv_ST725_bearing()) + " " +
        Double.toString(aries.get_auv_ST725_range()) + " " +
        Double.toString(aries.get_auv_ST725_strength()) + " " +
        Double.toString(aries.get_divetracker_range1()) + " " +
        Double.toString(aries.get_divetracker_range2()) + " " +
        Double.toString(aries.get_doppler_sog_u()) + " " +
        Double.toString(aries.get_doppler_sog_v()) + " " +
        Double.toString(aries.get_doppler_stw_u()) + " " +
        Double.toString(aries.get_doppler_stw_v()) + " " +
        Double.toString(aries.get_doppler_altitude()) ;

    return telemetryString;

}       // end of build-telemetry_string_to_file



/**
 *      To build the telemetry string out of the telemetry of the vehicle object
 *  @param:                 Vehicle object
 *  @return:    String, telemetry string
 */
public String build_telemetry_string(Vehicle aries) {
    // 42 state variables
    telemetryString = "auv_state" + " " +
        Double.toString(aries.get_time()) + " " +
        Double.toString(aries.get_x()) + " " +
        Double.toString(aries.get_y()) + " " +
        Double.toString(aries.get_z()) + " " +
        Double.toString(aries.get_phi()) + " " +
        Double.toString(aries.get_theta()) + " " +
        Double.toString(aries.get_psi()) + " " +
        Double.toString(aries.get_speed()) + " " +
        Double.toString(aries.get_u()) + " " +
        Double.toString(aries.get_v()) + " " +
        Double.toString(aries.get_w()) + " " +
        Double.toString(aries.get_p()) + " " +
        Double.toString(aries.get_q()) + " " +
        Double.toString(aries.get_r()) + " " +
        Double.toString(aries.get_x_dot()) + " " +
        Double.toString(aries.get_y_dot()) + " " +
```

```
        Double.toString(aries.get_z_dot()) + " " +
        Double.toString(aries.get_phi_dot()) + " " +
        Double.toString(aries.get_theta_dot()) + " " +
        Double.toString(aries.get_psi_dot()) + " " +
        Double.toString(aries.get_delta_rudder()) + " " +
        Double.toString(aries.get_delta_planes_stern()) + " " +
        // Double.toString( aries.get_delta_planes_bow() ) + " " +
        Double.toString(aries.get_port_rpm()) + " " +
        Double.toString(aries.get_stbd_rpm()) + " " +
        Double.toString(aries.get_auv_bow_vertical()) + " " +
        Double.toString(aries.get_auv_stern_vertical()) + " " +
        Double.toString(aries.get_auv_bow_lateral()) + " " +
        Double.toString(aries.get_auv_stern_lateral()) + " " +
        Double.toString(aries.get_auv_ST1000_bearing()) + " " +
        Double.toString(aries.get_auv_ST1000_range()) + " " +
        Double.toString(aries.get_auv_ST1000_strength()) + " " +
        Double.toString(aries.get_auv_ST725_bearing()) + " " +
        Double.toString(aries.get_auv_ST725_range()) + " " +
        Double.toString(aries.get_auv_ST725_strength()) + " " +
        Double.toString(aries.get_divetracker_range1()) + " " +
        Double.toString(aries.get_divetracker_range2()) + " " +
        Double.toString(aries.get_doppler_sog_u()) + " " +
        Double.toString(aries.get_doppler_sog_v()) + " " +
        Double.toString(aries.get_doppler_stw_u()) + " " +
        Double.toString(aries.get_doppler_stw_v()) + " " +
        Double.toString(aries.get_doppler_altitude()) + "\n";

    System.out.println("Telemetry String in IO: " + telemetryString);
    return telemetryString;
} // end of build_telemetry_string()

/**  */
public void get_Control_Coefficients() {
    if (flags.TRACE) {
        System.out.println("[Start get_Control_Coefficients]");
        System.out.println("File name: " + DEFAULTCONTROLCONSTANTSINPUTNAME);
    }
    int counter = 0;
    String fileName = DEFAULTCONTROLCONSTANTSINPUTNAME;
    String lineRead;
    open_File(fileName);
    try {
        lineRead = buffer.readLine();
        counter++;
        comment1 = lineRead.startsWith(" ");
        comment2 = lineRead.startsWith("_");
        while (lineRead != null) {
            tokens = new StringTokenizer(lineRead);
            if ((comment1 == true)) {
                if (lineRead.length() != 0) {
                    switch (counter) {
                        case 9: // k_psi   k_r    k_v    k_z    k_w    k_theta     k_q
                            control_coefficients.set_k_psi(
                                        Double.parseDouble((String)tokens.nextToken()));
                            control_coefficients.set_k_r(
                                Double.parseDouble((String)tokens.nextToken()));
```

```
    control_coefficients.set_k_v(
        Double.parseDouble((String)tokens.nextToken()));
    control_coefficients.set_k_z(
        Double.parseDouble((String)tokens.nextToken()));
    control_coefficients.set_k_w(
        Double.parseDouble(
        (String)tokens.nextToken()));
    control_coefficients.set_k_theta(
        Double.parseDouble(
        (String)tokens.nextToken()));
    control_coefficients.set_k_q(
        Double.parseDouble((String)tokens.nextToken()));
    if (flags.TRACE) {
        System.out.println("\n*** CONTROL COEFFICIENTS ARE: ***");
        System.out.println("[k_psi = " +
            control_coefficients.get_k_psi() + ", k_r = " +
            control_coefficients.get_k_r() + ", k_v = " +
            control_coefficients.get_k_v() +
            ", k_z = " + control_coefficients.get_k_z() +
            ", k_w = " + control_coefficients.get_k_w() +
            ", k_theta = " + control_coefficients.get_k_theta() +
            ", k_q = " + control_coefficients.get_k_q() + "]");
    }
    break;
case 15: // k_thruster_psi     k_thruster_r    k_thruster_rotate
    control_coefficients.set_k_thruster_psi(
                Double.parseDouble((String)tokens.nextToken()));
    control_coefficients.set_k_thruster_r(
        Double.parseDouble((String)tokens.nextToken()));
    control_coefficients.set_k_thruster_rotate(
        Double.parseDouble((String)tokens.nextToken()));
    if (flags.TRACE) {
        System.out.println("[k_thruster_psi = " +
            control_coefficients.get_k_thruster_psi() +
            ", k_thruster_r = " + control_coefficients.get_k_thruster_r() +
            ", k_thruster_rotate = " +
            control_coefficients.get_k_thruster_rotate() + "]");
    }
    break;
case 21: // k_thruster_z       k_thruster_w    k_thruster_theta
    control_coefficients.set_k_thruster_z(
                Double.parseDouble((String)tokens.nextToken()));
    control_coefficients.set_k_thruster_w(
        Double.parseDouble((String)tokens.nextToken()));
    control_coefficients.set_k_thruster_theta(
        Double.parseDouble((String)tokens.nextToken()));
    if (flags.TRACE) {
        System.out.println("[k_thruster_z = " +
            control_coefficients.get_k_thruster_z() +
            ", k_thruster_w = " + control_coefficients.get_k_thruster_w() +
            ", k_thruster_theta = " +
            control_coefficients.get_k_thruster_theta() + "]");
    }
    break;
case 27: // k_propeller_hover   k_surge_hover   k_propeller_current
    control_coefficients.set_k_propeller_hover(
```

```
                Double.parseDouble((String)tokens.nextToken()));
                control_coefficients.set_k_surge_hover(
                   Double.parseDouble((String)tokens.nextToken()));
                control_coefficients.set_k_propeller_current(
                   Double.parseDouble((String)tokens.nextToken()));
                if (flags.TRACE) {
                   System.out.println("[k_propeller_hover = " +
                      control_coefficients.get_k_propeller_hover() +
                      ", k_surge_hover = " + control_coefficients.get_k_surge_hover() +
                      ", k_propeller_current = " +
                      control_coefficients.get_k_propeller_current() + "]");
                }
                break;
             case 33: // k_thruster_hover    k_sway_hover    k_thruster_current
                control_coefficients.set_k_thruster_hover(
                Double.parseDouble((String)tokens.nextToken()));
                control_coefficients.set_k_sway_hover(
                   Double.parseDouble((String)tokens.nextToken()));
                control_coefficients.set_k_thruster_current(
                   Double.parseDouble((String)tokens.nextToken()));
                if (flags.TRACE) {
                   System.out.println("[k_thruster_hover = " +
                      control_coefficients.get_k_thruster_hover() +
                      ", k_sway_hover = " + control_coefficients.get_k_sway_hover() +
                      ", k_thruster_current = " +
                      control_coefficients.get_k_thruster_current() + "]");
                }
                break;
             case 39: // k_thruster_lateral  standoff_distance
                control_coefficients.set_k_thruster_lateral(
                Double.parseDouble((String)tokens.nextToken()));
                control_coefficients.set_standoff_distance(
                   Double.parseDouble((String)tokens.nextToken()));
                if (flags.TRACE) {
                   System.out.println("[k_thruster_lateral = "
                      + control_coefficients.get_k_thruster_lateral() +
                      ", standoff_distance = " +
                      control_coefficients.get_standoff_distance() + "]\n");
                }
                break;
             } // end of switch case
          }
       } // end of  if (!comment1){
       lineRead = buffer.readLine();
       counter++;
    } // // end of while ( )
    fileIn.close();
 } // end of try
 catch (Exception e) {
    System.out.println("Exception " + e + " occurred ");
 }
 if (flags.TRACE) {
    System.out.println("[End get_Control_Coefficients]\n");
 }
 return;
}
```

```
  /**  */
  public void record_data(Vehicle aries, Network_Connection network, Parser parser) {

    save_to_file(build_telemetry_string_to_file(aries));
    if (flags.TACTICALPARSE == false) {
      network.write_Telemetry_to_dynamics(build_telemetry_string(aries));
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("\nsending to virtual world");
      System.out.println(build_telemetry_string(aries));
    }
    if (flags.DISPLAYSCREEN) {
      if (flags.LOCATIONLAB == false) {

      }
      //        ++count;
      if (flags.LOCATIONLAB && flags.DISPLAYSCREEN) {
        System.out.println("sent telemetry to virtual world " +
          aries.get_time() + " " + aries.get_x() + " " +
          aries.get_y() + " " + aries.get_z() + " " + aries.get_phi() +
          " " + aries.get_theta() +
          " " + aries.get_psi());
      }
    }
    if (flags.LOCATIONLAB) {
      telemetry_from_dynamics = network.read_Telemetry_from_dynamics();
      if (flags.REPLAY) {
        //    read_telemetry_string();                    //p80 execution and p9 extern_funct
        build_telemetry_string(aries);
      }
      else {
        parser.parse_telemtery_string(telemetry_from_dynamics);
      }
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[finish record_data ()");
    }
  }
} // end of IO class
```

F. COMMANDS_QUEUE.JAVA

```
/* ____________________________________________________________________________________
  Title:          Commands_Queue.java
  Description:    This is a queue data structure used to read all the commands
                  from the mis sion script file.  Once the commands are read and
                  storaged in the queue, they can be read and popped out of the
                  queue.
                  NOTE: This queue implementation is done using a linked list.
  Date:           April 7, 2002
  Project:        Execution Java Software for Autonomous Underwater Vehicle,
                  Naval Postgraduate School, Monterey, CA
  Compiler:       JDK 1.3.1
  Author:         Miguel A. Ayala
  Version:        1.0
  ____________________________________________________________________________________
*/

package mil.navy.nps.auvAries.execution.input_output;

/**
 * This class is a queue data structure implemented using a linked list.
 * It will be the holder of all the commands for the AUV once read from the command mission script.
 */
public class Commands_Queue {
   private ListNode firstNode;
   private ListNode lastNode;
   private String name;
   private int commandsCount;

   /**
    *  Overloaded contructor. Construct an empty Stack with parameter "s" as the name.
    *  @param:      String s.  The name of the linked-list.
    *  @return:     None
    */
   public Commands_Queue(String s) {
      name = s;
      firstNode = lastNode = null;
      commandsCount = 0;
   }

   /**
    *  Constructor. Construct an empty Stack with "Commands" as the name.
    *  @param:      None
    *  @return:     None
    */
   public Commands_Queue() {
      this("Commands");
   }

   /**
    *       Stack operator responsible to add objects to the stack.
    *  @param:      Object insertItem.  Object to be inserted into the stack
    *  @return:     None
    */
```

```
public synchronized void push(Object insertItem) {
   if (isEmpty()) {
      firstNode = lastNode = new ListNode(insertItem);
      commandsCount++;
   }
   else {
      lastNode = lastNode.next = new ListNode(insertItem);
      commandsCount++;
   }
}

/**
 *  Stack operator responsible to remove the first object from the stack.
 *  @param:      None
 *  @return:     None
 */
public synchronized Object pop() {
   Object removeItem = null;
   if (isEmpty()) {
      System.out.println(name + " stack is empty!!\n");
   }
   else {
      removeItem = firstNode.data; // retrieve the data
      // reset the firstNode and lastNode references
      if (firstNode.equals(lastNode))
         firstNode = lastNode = null;
      else
         firstNode = firstNode.next;
      commandsCount--;
   }
   return (String)removeItem;
}

/**
 *  Return true if the Stack is empty.
 *  @param:      None
 *  @return:     None
 */
public synchronized boolean isEmpty() {
   return firstNode == null;
}

/**
 *       Output the Stack contents.
 *  @param:      None
 *  @return:     None
 */
public synchronized void print() {
   if (isEmpty()) {
      System.out.println("Empty " + name);
      return;
   }
   System.out.print("\nThe " + name + " are " + commandsCount + " : ");
   ListNode current = firstNode;
   while (current != null) {
      System.out.print(current.data.toString() + ", ");
```

```
            current = current.next;
        }
        System.out.println("\n");
    }

    /**
     *       Returns the number of commands currently in the stack.
     *  @param:      None
     *  @return:     Number of commands in the stack
     */
    public int get_commandsCount() {
        return commandsCount;
    }
} // end of public class Commands_Queue
```

G. PARSER.JAVA

```
/* ____________________________________________________________________________________
  Title:          Parser.java
  Description:    The purpose of this class is to parse the mission commands.
                  It parse the command one by one from a queue.  Every time a
                   new command is needed it is poped from the queue.  The
                  avantage of using a queue for holding all the commands is
                  that instead of reading it all from the script and not
                  knowing what command would be next, using the queue will
                  allow the auv to know exactly what is the next command.
                  Another queue could be implemented for commands already
                  executed so the auv needs to cancel a command it will know
                  what was the previous one.
  Date:           April 10, 2002
  Project:        Execution Java Software for Autonomous Underwater Vehicle,
                  Naval Postgraduate School, Monterey, CA
  Compiler:       JDK 1.3.1
  Author:         Miguel A. Ayala
  Version:        1.0
  ____________________________________________________________________________________
 */

package mil.navy.nps.auvAries.execution.decide;

import java.io.*;
import java.util.StringTokenizer;
import mil.navy.nps.auvAries.execution.globals.Execution_Flags;
import mil.navy.nps.auvAries.execution.control.*;
import mil.navy.nps.auvAries.execution.vehicle.Vehicle;
import mil.navy.nps.auvAries.execution.act.Act;
import mil.navy.nps.auvAries.execution.data_processing.Data_Processing;
import mil.navy.nps.auvAries.execution.input_output.IO;
import mil.navy.nps.auvAries.execution.input_output.Commands_Queue;

/**
 * The purpose of this class is to be a parser for the mission commands and flags.
 * The commands are obtained from a queue that already has the commands.  The queue
 * is then use to pop commands and identify them.  The flags are read from the
 * command line arguments imputted by the user. These flags are parsed one by one
 * and passed to the decide class object which will match them against pre-defined
 * flags.  If a macth is succesful that that flag is set.  The method
 * match_command_line_flags( ) is used for this task.  That method returns the
 * object flags with all the flags updated.
 */
public class Parser {
   /** The count of tokens found in the command string line read from
    * the mission script file. */
   private int tokenCount;

   /**
    * String containing the command string line.  This string is composed of
    * the command and parameters pertaining that command if applicable.
    */
   private String command_line;
```

```
/** Tokenize command extracted from the command string line */
private String command;

/** Parameters of the command.  Only applicable if the command has
 * parameters associated with it. */
private String[] parameters;

/** Parameters of the command.  Only applicable if the command has
 * parameters associated with it. */
private String[] parsed_command;

/**
 * A java.util class that allows you to break down a string into individual
 * componenets.  The object tokens is an instance of the class StringTokenizer.
 */
private StringTokenizer tokens;

/** Instance of the Decide class. */
private Decide decide;

/** Instance of Execution_Flags class. */
private Execution_Flags flags;

/** Instance of IO class. */
private IO io;

/** Instance of Commands_Queue class. */
private Commands_Queue queue;

/** Instance of Control class. */
private Control control;

/** Instance of Control_Coefficients class. */
private Control_Coefficients control_coefficients;

/** Instance of Vehicle class. */
private Vehicle vehicle;

/** Instance of Act class. */
private Act act;

/** Instance of Data_Processing class. */
private Data_Processing Data_Processing;

boolean first_line;

/**
 *  Constructor.  It receives an instance of the Execution_Flags class
 *  containing the flags for the AUV operations.
 *  @param:      Flag object containing the all the flags to be set.
 *  @return:     None
 */
public Parser(Execution_Flags flagsObj, IO ioObj, Control controlObj,
   Decide decideObj, Commands_Queue queueObj, Act actObj,
   Data_Processing Data_ProcessingRef, Vehicle vehicleRef) {
```

```
        command = "";
        command_line = "";
        tokenCount = 0;
        vehicle = vehicleRef;
        flags = flagsObj;
        control = controlObj;
        decide = decideObj;
        queue = queueObj;
        act = actObj;
        Data_Processing = Data_ProcessingRef;
        command_line = (String)queue.pop();
        tokens = new StringTokenizer(command_line);
        tokenCount = tokens.countTokens();
        parsed_command = new String[tokenCount];
        first_line = true;
}

/**
 * Method responsible to parse the commands from the mission script file.
 * Once parsed the command and parameters are sent to the match_commands()
 * method as arguments so they can be matched against a set of specific commands and act accordingly.
 * @param:  None
 * @return: None
 */
public void parse_mission_script_commands() {
    if (!first_line) {
        command_line = (String)queue.pop(); // get the first command
    }
    tokens = new StringTokenizer(command_line);
    command = null;
    while (tokens.hasMoreTokens()) {
        tokenCount = tokens.countTokens();
        command = (String)tokens.nextToken();
        System.out.println("\ncommand: " + command);
        parameters = new String[tokenCount];
        for (int i = 1; i < tokenCount; i++) {
            parameters[i] = tokens.nextToken();
            System.out.println("parameter[" + i + "]: " + parameters[i]);
        }
        decide.match_commands(command, parameters);
        first_line = false;
    } // end of while  (tokens.hasMoreTokens())
}

/**
 * Parse the telemetry string into its components.  Checks if the telemetry
 * vector is ok, if so the respective variables from the telemetry
 * are updated with the values parsed from the telemetry string received.
 * @param:      String.  Telemetry string vector.
 * @return:     None
 */
public void parse_telemtery_string(String telemetry_string) {
    int variables_parsed;
    String keyword;
    tokens = new StringTokenizer(telemetry_string);
    if (flags.TRACE && flags.DISPLAYSCREEN) {
```

```
    System.out.println("[begin parse_telemetry_string ()]\n");
}
if (flags.TRACE && flags.DISPLAYSCREEN) {
    System.out.println("from telemetry buffer: " + telemetry_string + "\n");
}
keyword = tokens.nextToken();
variables_parsed = tokens.countTokens(); // 42 state variables
if (flags.TRACE && flags.DISPLAYSCREEN) {
    System.out.println("variables_parsed = " + ( variables_parsed + 1 ) +
        ", STATEVECTORSIZE = " + vehicle.get_STATEVECTORSIZE() + "\n");
}

                    // transfer OK, keep new values
if ((variables_parsed) == vehicle.get_STATEVECTORSIZE() - 1 ) {
    vehicle.set_time(Double.parseDouble(tokens.nextToken()));
    vehicle.set_x(Double.parseDouble(tokens.nextToken()));
    vehicle.set_y(Double.parseDouble(tokens.nextToken()));
    vehicle.set_z(Double.parseDouble(tokens.nextToken()));
    vehicle.set_phi(Double.parseDouble(tokens.nextToken()));
    vehicle.set_theta(Double.parseDouble(tokens.nextToken()));
    vehicle.set_psi(Double.parseDouble(tokens.nextToken()));
    vehicle.set_speed(Double.parseDouble(tokens.nextToken()));
    vehicle.set_u(Double.parseDouble(tokens.nextToken()));
    vehicle.set_v(Double.parseDouble(tokens.nextToken()));
    vehicle.set_w(Double.parseDouble(tokens.nextToken()));
    vehicle.set_p(Double.parseDouble(tokens.nextToken()));
    vehicle.set_q(Double.parseDouble(tokens.nextToken()));
    vehicle.set_r(Double.parseDouble(tokens.nextToken()));
    vehicle.set_x_dot(Double.parseDouble(tokens.nextToken()));
    vehicle.set_y_dot(Double.parseDouble(tokens.nextToken()));
    vehicle.set_z_dot(Double.parseDouble(tokens.nextToken()));
    vehicle.set_phi_dot(Double.parseDouble(tokens.nextToken()));
    vehicle.set_theta_dot(Double.parseDouble(tokens.nextToken()));
    vehicle.set_psi_dot(Double.parseDouble(tokens.nextToken()));
    vehicle.set_delta_rudder(Double.parseDouble(tokens.nextToken()));
    vehicle.set_delta_planes_stern(Double.parseDouble(tokens.nextToken()));
  //vehicle.set_delta_planes_bow ( Double.parseDouble( tokens.nextToken() ));
    vehicle.set_port_rpm(Double.parseDouble(tokens.nextToken()));
    vehicle.set_stbd_rpm(Double.parseDouble(tokens.nextToken()));
    vehicle.set_auv_bow_vertical(Double.parseDouble(tokens.nextToken()));
    vehicle.set_auv_stern_vertical(Double.parseDouble(tokens.nextToken()));
    vehicle.set_auv_bow_lateral(Double.parseDouble(tokens.nextToken()));
    vehicle.set_auv_stern_lateral(Double.parseDouble(tokens.nextToken()));
    vehicle.set_auv_ST1000_bearing(Data_Processing.normalize(
        Double.parseDouble(tokens.nextToken())));
    vehicle.set_auv_ST1000_range(Double.parseDouble(tokens.nextToken()));
    vehicle.set_auv_ST1000_strength(Double.parseDouble(tokens.nextToken()));
    vehicle.set_auv_ST725_bearing(Data_Processing.normalize(
        Double.parseDouble(tokens.nextToken())));
    vehicle.set_auv_ST725_range(Double.parseDouble(tokens.nextToken()));
    vehicle.set_auv_ST725_strength(Double.parseDouble(tokens.nextToken()));
    vehicle.set_divetracker_range1(Double.parseDouble(tokens.nextToken()));
    vehicle.set_divetracker_range2(Double.parseDouble(tokens.nextToken()));
    vehicle.set_doppler_sog_u(Double.parseDouble(tokens.nextToken()));
    vehicle.set_doppler_sog_v(Double.parseDouble(tokens.nextToken()));
    vehicle.set_doppler_stw_u(Double.parseDouble(tokens.nextToken()));
```

```
    vehicle.set_doppler_stw_v(Double.parseDouble(tokens.nextToken()));
    vehicle.set_doppler_altitude(Double.parseDouble(tokens.nextToken()));
}
else if ((variables_parsed != vehicle.get_STATEVECTORSIZE() -1 ) &&
    (variables_parsed != -1)) {

    parse_mission_script_commands();
    if (flags.DISPLAYSCREEN)
        System.out.println("\nGarble problem in buffer_received !!! variables parsed = " +
        variables_parsed + "\n");
    flags.TRACE = true;
}
if (flags.TRACE && flags.DISPLAYSCREEN) {
    System.out.println("\nfrom telemetry buffer state variables: ");
    System.out.println("\n" + keyword + " time = " + vehicle.get_time() +
        " x = " + vehicle.get_x() + " y = " + vehicle.get_y() +
        " z = " + vehicle.get_z());
    System.out.println("phi = " + vehicle.get_phi() + " theta = " +
        vehicle.get_theta() + " psi = " + vehicle.get_psi());
    System.out.println("paddlewheel speed = " + vehicle.get_speed());
    System.out.println("u = " + vehicle.get_u() + " v = " + vehicle.get_v() +
        " w = " + vehicle.get_w() + " p = " + vehicle.get_p() +
        " q = " + vehicle.get_q() + " r = " + vehicle.get_r());
    System.out.println("x_dot = " + vehicle.get_x_dot() + " y_dot = " +
        vehicle.get_y_dot() + " z_dot = " + vehicle.get_z_dot());
    System.out.println("phi_dot = " + vehicle.get_phi_dot() +
        " theta_dot = " + vehicle.get_theta_dot() +
        " psi_dot = " + vehicle.get_psi_dot());
    System.out.println("delta_rudder = " + vehicle.get_delta_rudder() +
        " delta_planes_stern = " +
        vehicle.get_delta_planes_stern());
    System.out.println("port_rpm = " + vehicle.get_port_rpm() +
        " stbd_rpm = " + vehicle.get_stbd_rpm());
    System.out.println("bow_vertical = " + vehicle.get_auv_bow_vertical() +
        " stern_vertical = " + vehicle.get_auv_stern_vertical());
    System.out.println("bow_lateral = " + vehicle.get_auv_bow_lateral() +
        " stern_lateral = " + vehicle.get_auv_stern_lateral() );
    System.out.println("ST1000 b/r/s = " + vehicle.get_auv_ST1000_bearing() +
        " " + vehicle.get_auv_ST1000_range() + " " +
        vehicle.get_auv_ST1000_strength() );
    System.out.println("ST725 b/r/s = " + vehicle.get_auv_ST725_bearing() +
        " " + vehicle.get_auv_ST725_range() + " " +
        vehicle.get_auv_ST725_strength() );
    System.out.println("divetracker_range1 = " +
        vehicle.get_divetracker_range1() + " divetracker_range2 = " +
        vehicle.get_divetracker_range2() );
    System.out.println("doppler_sog_u = " + vehicle.get_doppler_sog_u() +
        " doppler_sog_v = " + vehicle.get_doppler_sog_v() );
    System.out.println("doppler_stw_u = " + vehicle.get_doppler_stw_u() +
        " doppler_stw_v = " + vehicle.get_doppler_stw_v() );
    System.out.println("doppler_altitude = " + vehicle.get_doppler_altitude() );

}
// keep all telemetry variables in degrees
vehicle.set_phi(Data_Processing.normalize2(vehicle.get_phi()));
vehicle.set_theta(Data_Processing.normalize2(vehicle.get_theta()));
```

```
        vehicle.set_psi(Data_Processing.normalize(vehicle.get_psi()));
        vehicle.set_phi_dot(Data_Processing.normalize2(vehicle.get_phi_dot()));
        vehicle.set_theta_dot(Data_Processing.normalize2(vehicle.get_theta_dot()));
        vehicle.set_psi_dot(Data_Processing.normalize2(vehicle.get_psi_dot()));
        vehicle.set_p(Data_Processing.normalize2(vehicle.get_p()));
        vehicle.set_q(Data_Processing.normalize2(vehicle.get_q()));
        vehicle.set_r(Data_Processing.normalize2(vehicle.get_r()));
        vehicle.set_delta_rudder(Data_Processing.normalize2(vehicle.get_delta_rudder()));
        vehicle.set_delta_planes_stern(Data_Processing.normalize2(
           vehicle.get_delta_planes_stern()));
        // delta_planes_bow   = normalize2 (delta_planes_bow);
        return;
    }
} // end of Parser class
```

H. LIST_NODE.JAVA

```
/* ____________________________________________________________________________________
  Title:          Commands_Queue.java
  Description:    This is a queue data structure used to read all the commands
                   from the mission script file.  Once the commands are read and
                  storaged in the queue, they can be read and popped out of the
                   queue.
  NOTE:           This queue implementation is done using a linked list.
  Date:           April 7, 2002
  Project:        Execution Java Software for Autonomous Underwater Vehicle,
                  Naval Postgraduate School, Monterey, CA
  Compiler:        JDK 1.3.1
  Author:         Miguel A. Ayala
  Version:        1.0
  ____________________________________________________________________________________
*/

package mil.navy.nps.auvAries.execution.input_output;

/** This class is the linked list data structure. */
class ListNode {
   Object data;
   ListNode next;

   /**
    *  Constructor. Creates a ListNode that refers to Object o.
    *  @param:      Object o.  The object which will be stored in the list.
    *  @return:     None
    */
   ListNode(Object o) {
      this(o, null);
   }

   /**
    *  Constructor. Creates a ListNode that refers to Object o and to the next ListNode in the List.
    *  @param:      Object o, ListNode nextNode.  Object which will be stored in the
    *  list and the next node that will be linked to it.
    *  @return:     None
    */
   ListNode(Object o, ListNode nextNode) {
      data = o; // this node refers to Object o
      next = nextNode; // set next to refer to next
   }

   /**
    *  Return a reference to the Object in "this" node.
    *  @param:      None
    *  @return:     None
    */
   Object getObject() {
      return data;
   }

   /**
```

```
    *  Return the next node.
    *  @param:      None
    *  @return:     None
    */
   ListNode getNext() {
      return next;
   }
} // end of class ListNode
```

I. DATA_PROCESSING.JAVA

```
/* ____________________________________________________________________________________________
   Title:          Data_Processing.java
   Description:
   Date:           23 May, 2002
   Project:        Execution Java Software for Autonomous Underwater Vehicle,
                   Naval Postgraduate School, Monterey, CA
   Compiler:       JDK 1.3.1
   Author:         Miguel A. Ayala
   Version:        1.0
   ____________________________________________________________________________________________
 */

package mil.navy.nps.auvAries.execution.data_processing;

import mil.navy.nps.auvAries.execution.globals.Execution_Flags;

/**
*
*/
public class Data_Processing {

   Execution_Flags flags;
   double degs;
   double clampee;
   double absolute_min;
   double absolute_max;
   double rads;
   double x;
   double y;
   double value;
   String name;

   public Data_Processing() {

      degs = 0.0;
      clampee = 0.0;
      absolute_min = 0.0;
      absolute_max = 0.0;
      rads = 0.0;
      x = 0.0;
      y = 0.0;
      value = 0.0;
      name = "";
   }

   /**
    *  @param:      double, double, double, String
    *  @return:     double
    */
   public double clamp(double clampee, double absolute_min, double absolute_max,
      String name) {

      double new_value, local_min, local_max;
```

```
    if ((absolute_max == 0.0) && (absolute_min == 0.0)) return clampee; // no clamp
    if (absolute_max >= absolute_min) { // ensure min & max used in proper order
        local_min = absolute_min;
        local_max = absolute_max;
    }
    else {
        local_min = absolute_max;
        local_max = absolute_min;
    }
    if ((clampee) > local_max) {
        new_value = local_max;
        if (flags.TRACE && flags.DISPLAYSCREEN)
            System.out.println("[clamping " + name + " from " + clampee +
            " to " + new_value + "]\n ");
        clampee = new_value;
    }
    if ((clampee) < local_min) {
        new_value = local_min;
        if (flags.TRACE && flags.DISPLAYSCREEN)
            System.out.println("[clamping " + name + " from " + clampee +
            " to " + new_value + "]\n ");
        clampee = new_value;
    }
    return clampee;
}

/**
 *  @param:      double
 *  @return:     double
 */
public double radians(double rads) {
    return rads * 180.0 / Math.PI;
}

/**
 *  @param:      double
 *  @return:     double
 */
public double normalize(double degs) {
    if (degs < -0) degs += 360.0;
    if (degs >= 360.0) degs -= 360.0;
    return degs;
}

/**
 *  @param:      double
 *  @return:     double
 */
public double normalize2(double degs) {
    if (degs <= -180.0) degs += 360.0;
    if (degs > 180.0) degs -= 360.0;
    return degs;
}

/**
 *  @param:      double, double
```

```
 *  @return:     double
 */
public double atan2z(double x, double y) {
   if (Math.abs(x) == 0.0 && Math.abs(y) == 0) return 0.0;
   return atan2(x, y);
}

/**
 *  @param:      double
 *  @return:     double
 */
public double atan2(double x, double y) {
   double return_value = 0.0;
   if (Math.abs(x) == 0.0) {
      if (y < 0.0) return_value = Math.PI / 2.0;
      else if (Math.abs(y) == 0) return_value = 0.0;
   }
   else {
      if (x < 0.0) {
         if (y < 0.0) return_value = Math.atan(x / y - Math.PI);
         else
            return_value = (Math.atan(x / y + Math.PI));
      }
      else
         return_value = (Math.atan(x / y));
   }
   return return_value;
}

/**
 *  @param:      double
 *  @return:     double
 */
public double degrees(double rads) {
   return rads * 180.0 / Math.PI;
}

/**
 *  @param:      double
 *  @return:     double
 */
public double dsign(double value) {
   double return_value = 0.0;
   if ((value == 0.0) || (value == -0.0)) return_value = 1.0;
   if (value > 0.0) return_value = 1.0;
   if (value < 0.0) return_value = -1.0;
   return return_value;
}

/**
 *  @param:      double
 *  @return:     double
 */
public double dtanh(double value) {
   if (Math.abs(value) > 1.0) {
      return (dsign(value));
```

```
      }
      else {
         return (value);
      }

            /*   double epv,emv;

              epv = exp(value);
              emv = exp(-value);

              return( (epv - emv)/(epv + emv) );
            */
   }
}           // end of data_processing class
```

J. VEHICLE.JAVA

```
/* ____________________________________________________________________________________
   Title:        Vehicle.java
   Description:        This class holds the telemetry of the vehicle.  All the
                                                 telemetry variables are private instance variables of this
                 class.  Set and Get methods have been developed in order to
                 access those instance variables
   Date:         April 16, 2002
   Project:      Execution Java Software for Autonomous Underwater Vehicle,
                 Naval Postgraduate School, Monterey, CA
   Compiler:     JDK 1.3.1
   Author:       Miguel A. Ayala
   Version:      1.0
   ____________________________________________________________________________________
*/

package mil.navy.nps.auvAries.execution.vehicle;

/**
 *          This class is reponsible of holding all the telemetry variables of the vehicle.
 * The telemetry data is saved in private instance variables accesible only by
 * public methods of this class.
 */
public class Vehicle {

   /** Size of the telemetry vector in terms of number of variables it is
    * holding
    */
   private final int STATEVECTORSIZE; // how many variables follow

   /**
    * Word that will identify the state of the calcualtion process and communication
    * between dynamics and execution.  Some of the key words are: auv_state: a good telemetry string
    * kill: the process have failed and execution and dymanics will shutdown quit: end of the mission
    */
   private String[] keyword;

   /** Mission time in seconds */
   private double time; // units are seconds

   /** x position in world coordinates in feet */
   private double x; // feet

   /** y position in world coordinates in feet */
   private double y; // feet

   /** z position in world coordinates in feet */
   private double z; // feet

   /** roll posture in world coordinates in degrees */
   private double phi; // degrees

   /** pitch posture in world coordinates in degrees */
   private double theta; // degrees
```

```
/** yaw posture in world coordinates in degrees */
private double psi; // degrees

/** Euler velocity along North-axis in feet/sec */
private double x_dot; // feet/sec

/** Euler velocity along  East-axis in feet/sec */
private double y_dot; // feet/sec

/** Euler velocity along Depth-axis in feet/sec */
private double z_dot; // feet/sec

/** Euler rotation rate about North-axis in degrees/sec */
private double phi_dot; // degrees/sec.

/** Euler rotation rate about East-axis in degrees/sec */
private double theta_dot; // degrees/sec

/** Euler rotation rate about Depth-axis in degree/sec */
private double psi_dot; // degrees/sec

/** paddlewheel speed, same as surge in feet/sec */
private double speed; // feet/sec (paddlewheel)

/** surge linear velocity along x-axis in feet/sec */
private double u; // feet/sec

/** sway linear velocity along y-axis in ft/sec */
private double v; // feet/sec

/** heave linear velocity along z-axis in ft/sec */
private double w; // feet/sec

/** roll angular velocity about x-axis in degrees/sec */
private double p; // degrees/sec

/** pitch angular velocity about y-axis in degrees/sec */
private double q; // degrees/sec

/** yaw angular velocity about z-axis in degrees/sec */
private double r; // degrees/sec

/**  inclination of the bow plane in degrees */
private double delta_planes_bow; // degrees

/** inclination of the stern plane in degrees */
private double delta_planes_stern; // degrees

/** positive is bow rudder to starboard, in degrees */
private double delta_rudder; // degrees

/**  propellor revolutions per minute port side */
private double port_rpm; // -700..700

/** propellor revolutions per minute starboard side */
```

```
private double stbd_rpm; // -700..700

/** thruster volts 24V = 3820 rpm no load */
private double auv_bow_vertical; // thruster  rpm

/**  thrusters volts 24V = 3820 rpm no load */
private double auv_stern_vertical; // thruster  rpm

/** thrusters volts 24V = 3820 rpm no load */
private double auv_bow_lateral; // thruster  rpm

/** thrusters volts 24V = 3820 rpm no load */
private double auv_stern_lateral; // thruster  rpm

/**  */
private double auv_ST1000_bearing; // ST_1000 conical pencil bearing degrees

/**  */
private double auv_ST1000_range; // ST_1000 conical pencil range   feet

/**  */
private double auv_ST1000_strength; // ST_1000 conical pencil strength dB

/**  */
private int auv_ST1000_direction; //  RELATIVE bearing = 0  TRUE bearing = 1

/**  */
private double auv_ST725_bearing; // ST_725  1 x 24 sector bearing  degrees

/**  */
private double auv_ST725_range; // ST_725  1 x 24 sector range    feet

/**  */
private double auv_ST725_strength; // ST_725  1 x 24 sector strength  dB

/**  */
private int auv_ST725_direction; //  RELATIVE bearing = 0
//  TRUE bearing = 1

/**  */
private double divetracker_range1; // feet range to divetracker unit 1

/**  */
private double divetracker_range2; // feet range to divetracker unit 2

/**  */
private double doppler_sog_u; // speed over ground in meters/sec

/**  */
private double doppler_sog_v; // speed over ground in meters/sec

/**  */
private double doppler_stw_u; // speed through water in meters/sec

/**  */
private double doppler_stw_v; // speed through water in meters/sec
```

```
/**  */
private double doppler_altitude; // altitude above bottom in meters

/**
 *        Vehicle constructor. It instantiate all the private variables composing
 * the telemetry vector.  This telemetry vector variables are updated every
 * 0.1 seconds or 10 Hz in accordance with the stability control requirements.
 *  @param:                    None
 *  @return:      None
 */
public Vehicle() {

   STATEVECTORSIZE = 42; // how many variables follow
   keyword = null;
   time = 0.0; // units are seconds
   x = 5.0; // feet
   y = 5.0; // feet
   z = 2.0; // feet
   phi = 0.0; // degrees
   theta = 0.0; // degrees
   psi = 0.0; // degrees
   x_dot = 0.0; // feet/sec
   y_dot = 0.0; // feet/sec
   z_dot = 0.0; // feet/sec
   phi_dot = 0.0; // degrees/sec
   theta_dot = 0.0; // degrees/sec
   psi_dot = 0.0; // degrees/sec
   speed = 0.0; // feet/sec (paddlewheel)
   u = 0.0; // feet/sec
   v = 0.0; // feet/sec
   w = 0.0; // feet/sec
   p = 0.0; // degrees/sec
   q = 0.0; // degrees/sec
   r = 0.0; // degrees/sec
   delta_planes_bow = 0.0; // degrees
   delta_planes_stern = 0.0; // degrees
   delta_rudder = 0.0; // degrees
   port_rpm = 0; // -700..700
   stbd_rpm = 0; // -700..700
   auv_bow_vertical = 0.0; // thruster  rpm
   auv_stern_vertical = 0.0; // thruster  rpm
   auv_bow_lateral = 0.0; // thruster  rpm
   auv_stern_lateral = 0.0; // thruster  rpm
   auv_ST1000_bearing = 0.0; // ST_1000 conical pencil bearing degrees
   auv_ST1000_range = 0.0; // ST_1000 conical pencil range   feet
   auv_ST1000_strength = -1.0; // ST_1000 conical pencil strength dB
   auv_ST1000_direction = 0; //  RELATIVE bearing = 0  TRUE bearing = 1
   auv_ST725_bearing = 0.0; // ST_725  1 x 24 sector bearing  degrees
   auv_ST725_range = 0.0; // ST_725  1 x 24 sector range    feet
   auv_ST725_strength = -1.0; // ST_725  1 x 24 sector strength  dB
   auv_ST725_direction = 0; //  RELATIVE bearing = 0  TRUE bearing = 1
   divetracker_range1 = -1.0; // feet range to divetracker unit 1
   divetracker_range2 = -1.0; // feet range to divetracker unit 2
   doppler_sog_u = 0.0; // speed over ground in meters/sec
   doppler_sog_v = 0.0; // speed over ground in meters/sec
```

```
        doppler_stw_u = 0.0; // speed through water in meters/sec
        doppler_stw_v = 0.0; // speed through water in meters/sec
        doppler_altitude = 0.0; // altitude above bottom in meters
    } // end of constructor
    // ***************************************************
    // *                    SET METHODS              *
    // ***************************************************

    /**
     * Sets the mission time every time the telemetry vector is updated. It would go in 0.1 increments starting
with 0.0.
     * @param:      double
     * @return:     None
     */
    public void set_time(double timeSetted) {
        time = timeSetted;
    }

    /**
     * Sets vehicle's x position in world coordinates in feet
     * @param:      double
     * @return:     None
     */
    public void set_x(double x_position) {
        x = x_position;
    }

    /**
     * Sets vehicle's y position in world coordinates in feet
     * @param:      double
     * @return:     None
     */
    public void set_y(double y_position) {
        y = y_position;
    }

    /**
     * Sets vehicle's z position in world coordinates in feet
     * @param:      double
     * @return:     None
     */
    public void set_z(double z_position) {
        z = z_position;
    }

    /**
     * Sets vehicle's roll posture in world coordinates in degrees
     * @param:      double
     * @return:     None
     */
    public void set_phi(double phi_degrees) {
        phi = phi_degrees;
    }

    /**
     * Sets vehicle's pitch posture in world coordinates in degrees
```

```
 * @param:      double
 * @return:     None
 */
public void set_theta(double theta_degrees) {
   theta = theta_degrees;
}

/**
 * Sets vehicle's yaw posture in world coordinates in degrees
 * @param:      double
 * @return:     None
 */
public void set_psi(double psi_degrees) {
   psi = psi_degrees;
}

/**
 * Sets vehicle's Euler velocity along North-axis in feet/sec
 * @param:      double
 * @return:     None
 */
public void set_x_dot(double x_velocity) {
   x_dot = x_velocity;
}

/**
 * Sets vehicle's Euler velocity along East-axis in feet/sec
 * @param:      double
 * @return:     None
 */
public void set_y_dot(double y_velocity) {
   y_dot = y_velocity;
}

/**
 * Sets vehicle's Euler velocity along Depth-axis in feet/sec
 * @param:      double
 * @return:     None
 */
public void set_z_dot(double z_velocity) {
   z_dot = z_velocity;
}

/**
 * Sets vehicle's Euler rotation rate about North-axis in degrees/sec
 * @param:      double
 * @return:     None
 */
public void set_phi_dot(double phi_velocity) {
   phi_dot = phi_velocity;
}

/**
 * Sets vehicle's Euler rotation rate about East-axis in degrees/sec
 * @param:      double
 * @return:     None
```

```
 */
public void set_theta_dot(double theta_velocity) {
   theta_dot = theta_velocity;
}

/**
 * Sets vehicle's Euler rotation rate about Depth-axis in degree/sec
 * @param:      double
 * @return:     None
 */
public void set_psi_dot(double psi_velocity) {
   psi_dot = psi_velocity;
}

/**
 * Sets vehicle's paddlewheel speed, same as surge in feet/sec
 * @param:      double
 * @return:     None
 */
public void set_speed(double speed_read) {
   speed = speed_read;
}

/**
 * Sets vehicle's surge linear velocity along x-axis in feet/sec
 * @param:      double
 * @return:     None
 */
public void set_u(double u_velocity) {
   u = u_velocity;
}

/**
 * Sets vehicle's sway linear velocity along y-axis in feet/sec
 * @param:      double
 * @return:     None
 */
public void set_v(double v_velocity) {
   v = v_velocity;
}

/**
 * Sets vehicle's heave linear velocity along z-axis in feet/sec
 * @param:      double
 * @return:     None
 */
public void set_w(double w_velocity) {
   w = w_velocity;
}

/**
 * Sets vehicle's roll angular velocity about x-axis in feet/sec
 * @param:      double
 * @return:     None
 */
public void set_p(double p_read) {
```

```
    p = p_read;
}

/**
 * Sets vehicle's pitch angular velocity about y-axis in feet/sec
 * @param:       double
 * @return:      None
 */
public void set_q(double q_read) {
    q = q_read;
}

/**
 * Sets vehicle's yaw angular velocity about z-axis in feet/sec
 * @param:       double
 * @return:      None
 */
public void set_r(double r_read) {
    r = r_read;
}

/**
 * Sets bow planes delta
 * @param:       double
 * @return:      None
 */
public void set_delta_planes_bow(double delta_planes_bow_read) {
    delta_planes_bow = delta_planes_bow_read;
}

/**
 * Stes stern planes delta
 * @param:       double
 * @return:      None
 */
public void set_delta_planes_stern(double delta_planes_stern_read) {
    delta_planes_stern = delta_planes_stern_read;
}

/**
 * Sets delta rudders.  Positive is bow rudder to starboard
 * @param:       double
 * @return:      None
 */
public void set_delta_rudder(double delta_rudder_read) {
    delta_rudder = delta_rudder_read;
}

/**
 * Sets port rpms
 * @param:       double
 * @return:      None
 */
public void set_port_rpm(double port_rpm_read) {
    port_rpm = port_rpm_read;
}
```

```
/**
 * Sets starboard rpms
 * @param:       double
 * @return:      None
 */
public void set_stbd_rpm(double stbd_rpm_read) {
   stbd_rpm = stbd_rpm_read;
}

/**
 * Sets bow thrusters voltage.
 * +- 24V <=> +-2 lb, + Volts moves thruster in + direction, all identical
 * @param:       double
 * @return:      None
 */
// +- 24V <=> +-2 lb, + Volts moves thruster in + direction, all identical
public void set_auv_bow_vertical(double bow_vertical_thruster) {
   auv_bow_vertical = bow_vertical_thruster;
}

/**
 * Sets stern thrusters voltage.
 * +- 24V <=> +-2 lb, + Volts moves thruster in + direction, all identical
 * @param:       double
 * @return: None
 */
public void set_auv_stern_vertical(double stern_vertical_thruster) {
   auv_stern_vertical = stern_vertical_thruster;
}

/**
 * Sets bow lateral thrusters voltage
 * @param:       double
 * @return: None
 */
public void set_auv_bow_lateral(double bow_lateral_thruster) {
   auv_bow_lateral = bow_lateral_thruster;
}

/**
 * Sets stern lateral thrusters voltage
 * @param:       double
 * @return: None
 */
public void set_auv_stern_lateral(double stern_lateral_thruster) {
   auv_stern_lateral = stern_lateral_thruster;
}

/**
 * Sets ST1000 bearing in degrees
 * @param:       double
 * @return: None
 */
public void set_auv_ST1000_bearing(double auv_ST1000_bearing_read) {
   auv_ST1000_bearing = auv_ST1000_bearing_read;
```

```
}

/**
 * Sets ST1000 range in ft
 * @param:       double
 * @return: None
 */
public void set_auv_ST1000_range(double auv_ST1000_range_read) {
   auv_ST1000_range = auv_ST1000_range_read;
}

/**
 * Sets ST1000 strength in dB
 * @param:       double
 * @return: None
 */
public void set_auv_ST1000_strength(double auv_ST1000_strength_read) {
   auv_ST1000_strength = auv_ST1000_strength_read;
}
// this one is commented on top

/**
 * Sets ST1000 direction TRUE = 1, RELATIVE = 0
 * @param:       double
 * @return: None
 */
public void set_auv_ST1000_direction(int auv_ST1000_direction_read) {
   auv_ST1000_direction = auv_ST1000_direction_read;
}

/**
 * Sets ST725 bearing in degrees
 * @param:       double
 * @return: None
 */
public void set_auv_ST725_bearing(double auv_ST725_bearing_read) {
   auv_ST725_bearing = auv_ST725_bearing_read;
}

/**
 * Sets ST725 range in feet
 * @param:       double
 * @return: None
 */
public void set_auv_ST725_range(double auv_ST725_range_read) {
   auv_ST725_range = auv_ST725_range_read;
}

/**
 * Sets ST725 strength in dB
 * @param:       double
 * @return: None
 */
public void set_auv_ST725_strength(double auv_ST725_strength_read) {
   auv_ST725_strength = auv_ST725_strength_read;
}
```

```
/**
 * Sets ST725 direction TRUE = 1, RELATIVE = 0
 * @param:       double
 * @return: None
 */
public void set_auv_ST725_direction(int auv_ST725_direction_read) {
   auv_ST725_direction = auv_ST725_direction_read;
}

/**
 * Sets divetracker unit1 range in feet
 * @param:       double
 * @return: None
 */
public void set_divetracker_range1(double divetracker_range1_read) {
   divetracker_range1 = divetracker_range1_read;
}

/**
 * Sets divetracker unit2 range in feet
 * @param:       double
 * @return: None
 */
public void set_divetracker_range2(double divetracker_range2_read) {
   divetracker_range2 = divetracker_range2_read;
}

/**
 * Sets speed over ground along x-axis in meters/sec
 * @param:       double
 * @return: None
 */
public void set_doppler_sog_u(double u_speed_over_ground_read) {
   doppler_sog_u = u_speed_over_ground_read;
}

/**
 * Sets speed over ground along y-axis in meters/sec
 * @param:       double
 * @return: None
 */
public void set_doppler_sog_v(double v_speed_over_ground_read) {
   doppler_sog_v = v_speed_over_ground_read;
}

/**
 * Sets speed through water along x-axis in meters/sec
 * @param:       double
 * @return: None
 */
public void set_doppler_stw_u(double u_speed_over_water_read) {
   doppler_stw_u = u_speed_over_water_read;
}

/**
```

```
 * Sets speed through water along y-axis in meters/sec
 * @param:      double
 * @return: None
 */
public void set_doppler_stw_v(double v_speed_over_water_read) {
   doppler_stw_v = v_speed_over_water_read;
}

/**
 * Sets altitude above bottom in meters
 * @param:      double
 * @return: None
 */
public void set_doppler_altitude(double doppler_altitude_read) {
   doppler_altitude = doppler_altitude_read;
}
// ***************************************************
// *                                GET METHODS              *
// ***************************************************

/**
 * Returns the size of the telemetry string
 * @param:      None
 * @return:     integer
 */
public int get_STATEVECTORSIZE() {
   return STATEVECTORSIZE;
}

/**
 * Returns mission time in seconds
 * @param:      None
 * @return:     double
 */
public double get_time() {
   return time;
}

/**
 * Return x position in world coordinates in feet
 * @param:      None
 * @return:     double
 */
public double get_x() {
   return x;
}

/**
 * Return y position in world coordinates in feet
 * @param:      None
 * @return:     double
 */
public double get_y() {
   return y;
}
```

```
/**
 * Return z position in world coordinates in feet
 * @param:      None
 * @return:     double
 */
public double get_z() {
   return z;
}

/**
 * Returns roll posture in world coordinates in degrees
 * @param:      None
 * @return:     double
 */
public double get_phi() {
   return phi;
}

/**
 * Returns pitch posture in world coordinates in degrees
 * @param:      None
 * @return: double
 */
public double get_theta() {
   return theta;
}

/**
 * Returns yaw posture in world coordinates in degrees
 * @param:      None
 * @return: double
 */
public double get_psi() {
   return psi;
}

/**
 * Returns Euler velocity along North-axis in ft/sec
 * @param:      None
 * @return: double
 */
public double get_x_dot() {
   return x_dot;
}

/**
 * Returns Euler velocity along East-axis in ft/sec
 * @param:      None
 * @return: double
 */
public double get_y_dot() {
   return y_dot;
}

/**
 * Returns Euler velocity along Depth-axis in ft/sec
```

```
 * @param:       None
 * @return: double
 */
public double get_z_dot() {
   return z_dot;
}

/**
 * Returns rotation rate along North-axis in degrees/sec
 * @param:       None
 * @return: double
 */
public double get_phi_dot() {
   return phi_dot;
}

/**
 * Returns rotation rate along East-axis in degrees/sec
 * @param:       None
 * @return: double
 */
public double get_theta_dot() {
   return theta_dot;
}

/**
 * Returns rotation rate along Depth-axis in degrees/sec
 * @param:       None
 * @return: double
 */
public double get_psi_dot() {
   return psi_dot;
}

/**
 * Returns paddlewheel speed in ft/sec
 * @param:       None
 * @return: double
 */
public double get_speed() {
   return speed;
}

/**
 * Returns surge linear velocity along x-axis in ft/sec
 * @param:       None
 * @return: double
 */
public double get_u() {
   return u;
}

/**
 * Returns sway linear velocity along y-axis in ft/sec
 * @param:       None
 * @return: double
```

```
 */
public double get_v() {
   return v;
}

/**
 * Returns heave linear velocity along z-axis in ft/sec
 * @param:       None
 * @return: double
 */
public double get_w() {
   return w;
}

/**
 * Returns roll angular velocity about x-axis in degrees/sec
 * @param:       None
 * @return: double
 */
public double get_p() {
   return p;
}

/**
 * Returns pitch angular velocity about y-axis in degrees/sec
 * @param:       None
 * @return: double
 */
public double get_q() {
   return q;
}

/**
 * Returns yaw angular velocity about z-axis in degrees/sec
 * @param:       None
 * @return: double
 */
public double get_r() {
   return r;
}

/**
 * Returns bow delta planes in degrees
 * @param:       None
 * @return: double
 */
public double get_delta_planes_bow() {
   return delta_planes_bow;
}

/**
 * Returns stern delta planes in degrees
 * @param:       None
 * @return: double
 */
public double get_delta_planes_stern() {
```

```
    return delta_planes_stern;
}

/**
 * Returns delta rudder in degrees. Positive is bow rudder to starboard
 * @param:       None
 * @return: double
 */
public double get_delta_rudder() {
    return delta_rudder;
}

/**
 * Returns port rpms
 * @param:       None
 * @return: double
 */
public double get_port_rpm() {
    return port_rpm;
}

/**
 * Return starboard rpms
 * @param:       None
 * @return: double
 */
public double get_stbd_rpm() {
    return stbd_rpm;
}

/**
 *
 * @param:       None
 * @return: double
 */
// +- 24V <=> +-2 lb, + Volts moves thruster in + direction, all identical
public double get_auv_bow_vertical() {
    return auv_bow_vertical;
}

/**
 * @param:       None
 * @return: double
 */
public double get_auv_stern_vertical() {
    return auv_stern_vertical;
}

/**
 * @param:       None
 * @return: double
 */
public double get_auv_bow_lateral() {
    return auv_bow_lateral;
}
```

```
/**
 * @param:       None
 * @return: double
 */
public double get_auv_stern_lateral() {
   return auv_stern_lateral;
}

/**
 * Returns ST1000 bearing in degress
 * @param:       None
 * @return: double
 */
public double get_auv_ST1000_bearing() {
   return auv_ST1000_bearing;
}

/**
 * Returns ST1000 range in ft
 * @param:       None
 * @return: double
 */
public double get_auv_ST1000_range() {
   return auv_ST1000_range;
}

/**
 * Returns ST1000 strength in dB
 * @param:       None
 * @return: double
 */
public double get_auv_ST1000_strength() {
   return auv_ST1000_strength;
}

/**
 * Returns ST1000 direction TRUE = 1, RELATIVE = 0
 * @param:       None
 * @return: double
 */
public int get_auv_ST1000_direction() {
   return auv_ST1000_direction;
}

/**
 * Returns ST725 bearing in degrees
 * @param:       None
 * @return: double
 */
public double get_auv_ST725_bearing() {
   return auv_ST725_bearing;
}

/**
 * Returns ST725 range in ft
 * @param:       None
```

```
 * @return: double
 */
public double get_auv_ST725_range() {
   return auv_ST725_range;
}

/**
 * Returns ST725 strength in dB
 * @param:       None
 * @return: double
 */
public double get_auv_ST725_strength() {
   return auv_ST725_strength;
}

/**
 * Returns ST725 direction TRUE = 1, RELATIVE = 0
 * @param:       None
 * @return: double
 */
public int get_auv_ST725_direction() {
   return auv_ST725_direction;
}

/**
 * Returns divetracker unit1 range in ft
 * @param:       None
 * @return: double
 */
public double get_divetracker_range1() {
   return divetracker_range1;
}

/**
 * Returns divetracker unit2 range in ft
 * @param:       None
 * @return: double
 */
public double get_divetracker_range2() {
   return divetracker_range2;
}

/**
 * Returns speed over ground in meters/sec about x-axis
 * @param:       None
 * @return: double
 */
public double get_doppler_sog_u() {
   return doppler_sog_u;
}

/**
 * Returns speed over ground in meters/sec about y-axis
 * @param:       None
 * @return: double
 */
```

```
    public double get_doppler_sog_v() {
        return doppler_sog_v;
    }

    /**
     * Returns speed through water in meters/sec about x-axis
     * @param:       None
     * @return: double
     */
    public double get_doppler_stw_u() {
        return doppler_stw_u;
    }

    /**
     * Returns speed through water in meters/sec about y-axis
     * @param:       None
     * @return: double
     */
    public double get_doppler_stw_v() {
        return doppler_stw_v;
    }

    /**
     * Returns altitude above bottom in meters
     * @param:       None
     * @return: double
     */
    public double get_doppler_altitude() {
        return doppler_altitude;
    }

} // end of Vehicle class
```

K. NETWORK_CONNECTION.JAVA

```
/* ____________________________________________________________________________________________
   Title:        Network_Connection.java
   Description:        This class is responsible for the network connection between
                execution and dynamics.  It reads and writes the telemetry data
                from and to dynamics through a socket connection.
   Date:        April 16, 2002
   Project:      Execution Java Software for Autonomous Underwater Vehicle,
                Naval Postgraduate School, Monterey, CA
   Compiler:     JDK 1.3.1
   Author:       Miguel A. Ayala
   Version:      1.0
   ____________________________________________________________________________________________
*/

package mil.navy.nps.auvAries.execution.network;

import java.net.*;
import java.io.*;
import mil.navy.nps.auvAries.execution.globals.Execution_Flags;
import mil.navy.nps.auvAries.execution.input_output.Parser;

/**
 *  This class is responsible for the network connection between execution and
 * dynamics.  It reads and writes the telemetry
 * data from and to dynamics through a socket connection.
 */
public class Network_Connection {
   /** Port use between dynamics and execution to connect through a network */
   private final int AUVSIM1_TCP_PORT1;

   /** Port use between tactical and execution to connect through a network */
   private final int AUVSIM1_TCP_PORT2;

   /** Port use between tactical and execution to connect through a network
    * and pass massages
    */
   private final int AUVSIM1_TCP_PORT3;

   /** Dynamic host name used in the network */
   private final String VIRTUAL_WORLD_REMOTE_HOST_NAME;

   /** tactical host name used in the network */
   public final String TACTICAL_REMOTE_HOST_NAME;

   /** Stream object use to write the telemetry to dynamics. */
   private DataOutputStream dynamics_output;

   /** Stream object use to read the telemetry from dynamics. */
   private DataInputStream dynamics_input;

   /** Stream object use to communicate with tactical. */
   private DataOutputStream tactical_output;
```

```
/** Stream object use to read from tactical. */
private DataInputStream tactical_input;

/** Telemetry string that will be read from dynamics. */
private String telemetry_from_dynamics;

/** Message string that will be read from tactical. */
private String message_from_tactical;

/** Socket to be used for the network connection between execution and dynamics. */
private Socket dynamics_network_connection;

/** Socket to be used for the network connection between execution and tactical. */
private Socket tactical_network_connection;

/** Reference to the execution flags to be used */
private Execution_Flags flags;

/**
 *  Constructor
 *  @param:      flagsRef a reference of the Execution_Flags object
 *  @return:     None
 */
public Network_Connection(Execution_Flags flagsRef) {
   AUVSIM1_TCP_PORT1 = 3211;
   AUVSIM1_TCP_PORT2 = 3212;
   AUVSIM1_TCP_PORT3 = 3213;
   VIRTUAL_WORLD_REMOTE_HOST_NAME = "localhost";
   TACTICAL_REMOTE_HOST_NAME = "localhost";
   flags = flagsRef;
   telemetry_from_dynamics = "";
   message_from_tactical = "";
} // end of the Network_Connection constructor
//*********************************************************
//*      METHODS OF NETWORK_CONNECTION CLASS            *
//*********************************************************

/**
 * Opens a network connection using VIRTUAL_WORLD_REMOTE_HOST_NAME ( localhost )
 * as host name and AUVSIM1_TCP_PORT1 ( port 3211 ) as port if the location is
 * tha lab or TACTICAL_REMOTE_HOST_NAME ( localhost ) and AUVSIM1_TCP_PORT2
 * ( port 3212 ) if using in the real world.
 * @param:      None
 * @return:     None
 */
public void open_network_connection() {
   try {
      if (flags.LOCATIONLAB) {
         // Step 1: Create a Socket to make connection.
         System.out.println("Attempting connection\n");
         dynamics_network_connection = new Socket(InetAddress.getByName(
            VIRTUAL_WORLD_REMOTE_HOST_NAME), AUVSIM1_TCP_PORT1);
         System.out.println("Connected to virtual world at: " +
            dynamics_network_connection.getInetAddress().getHostName());
         // Step 2: Get the input and output streams.
         dynamics_output = new DataOutputStream(
```

```
            dynamics_network_connection.getOutputStream());
        dynamics_output.flush();
        dynamics_input = new DataInputStream(
            dynamics_network_connection.getInputStream());
        System.out.println("\nGot I/O streams from dynamics\n");
    } // end of if ( flags.LOCATIONLAB )

    if (flags.TACTICAL) {
        // Step 1: Create a Socket to make connection.
        System.out.println("Attempting connection\n");
        tactical_network_connection = new Socket(InetAddress.getByName(
            TACTICAL_REMOTE_HOST_NAME), AUVSIM1_TCP_PORT2);
        System.out.println("Connected to tactical at: " +
            tactical_network_connection.getInetAddress().getHostName());
        // Step 2: Get the input and output streams.
        tactical_output = new DataOutputStream(
            tactical_network_connection.getOutputStream());
        tactical_output.flush();
        tactical_input = new DataInputStream(
            tactical_network_connection.getInputStream());
        System.out.println("\nGot I/O streams from tactical\n");
    } // end of if ( flags.TACTICAL )
  }
  catch (Exception e) {
    System.out.println("Execption " + e +
        " occurred while trying to connect to network.");
  }
}

/**
 *  Send the telemetry data to dynamics via the netwrok.
 *  @param:      Telemetry string to be written to dynamics.
 *  @return:     None
 */
public void write_Telemetry_to_dynamics(String telemetry_to_dynamics) { //P77 EXECUTION
  // NEED TO ADD SOMETHING ABOUT KILL???
  try {
    dynamics_output.writeBytes(telemetry_to_dynamics);
    dynamics_output.flush();
    System.out.println("Telemetry Sent: " + telemetry_to_dynamics);
  }
  catch (Exception e) {
    System.out.println("\nError writing telemetry to dynamics. Exception " +
        e + " occurred.");
  }
}

/**
 *  Read the telemetry string from dynamics through the network.
 *  @param:      None.
 *  @return:     Telemetry string read from dynamics.
 */
public String read_Telemetry_from_dynamics() { //P78 EXECUTION
  try {
    telemetry_from_dynamics = (String)dynamics_input.readLine();
    System.out.println("Telemetry form dynamics: " + telemetry_from_dynamics);
```

```
        }
        catch (Exception e) {
            System.out.println("\nError reading telemetry from dynamics. Exception " +
                e + " occurred.");
        }
        return telemetry_from_dynamics;
    }
    // NEED TO PARSE TELEMETRY FROM DYNAMICS

    /**
     *  Read the commands string from tactical through the network.
     *  @param:      None.
     *  @return:     String.  Message string read from tactical.
     */
    public String read_from_tactical() { //P78 EXECUTION
        try {
            message_from_tactical = (String)tactical_input.readLine();
            System.out.println("Telemetry form dynamics: " + message_from_tactical);
        }
        catch (Exception e) {
            System.out.println("\nError reading from tactical. Exception " + e +
                " occurred.");
        }
        return message_from_tactical;
    }

    /**
     *  Send the information to tactical via the netwrok.
     *  @param:      Message string to tactical.
     *  @return:     None
     */
    public void write_to_tactical(String tactical_message) { //P77 EXECUTION
        try {
            tactical_output.writeBytes(tactical_message);
            tactical_output.flush();
            if (flags.TRACE) {
                System.out.println("Sent to tactical: " + tactical_message);
            }
        }
        catch (Exception e) {
            System.out.println("\nError writing to tactical. Exception " + e +
                " occurred.");
        }
    }
} // end of Network_Connection class
```

L. EXECUTION_FLAGS.JAVA

```
package mil.navy.nps.auvAries.execution.globals;

/* ____________________________________________________________________________
   Title:        Execution_Flags.java
   Description:        Execution Level Flags
   Date:         April 10, 2002
   Project:      Execution Java Software for Autonomous Underwater Vehicle,
                 Naval Postgraduate School, Monterey, CA
   Compiler:     JDK 1.3.1
   Author:       Miguel A. Ayala
   Version:      1.0
   ____________________________________________________________________________
*/

/** This class has all the flags used in the Execution level software. */
public class Execution_Flags {

    /** Flag to trace the execution program  */
    public static boolean TRACE; //

    /** Flag to display on screen the execution of the program  */
    public static boolean DISPLAYSCREEN; //

    /** Loaction of AUV in lab and use of virtual world True or False  */
    public static boolean LOCATIONLAB;

    /** Tactical communication in use */
    public static boolean TACTICAL; //

    /** Repeat execution indefinitely  */
    public static boolean LOOPFOREVER; //  repeat execution indefinitely

    /** Backup files between replications  */
    public static boolean LOOPFILEBACKUP; //  backup files between replications

    /** trace each char received at port  */
    public static boolean PARALLELPORTTRACE; //  trace each char received at port

    /** Sonar ST1000 head available for query */
    public static boolean ST1000INSTALLED; //  sonar head available for query

    /** Sonar ST725 head available for query */
    public static boolean ST725INSTALLED; //  sonar head available for query

    /** Trace the sonar responses */
    public static boolean SONARTRACE; //  trace on or trace off

    public static int ST1000SCANMODE; //  WHERE IS THE ENUMERATION
    public static int ST725SCANMODE; //  WHERE IS THE ENUMERATION
    public static double SONARHEADINGSTEP; //  degrees

    /** TRUE or RELATIVE direction */
    public static int ST1000SCANDIRECTION;
```

```
/** TRUE or RELATIVE direction */
public static int ST725SCANDIRECTION;

/** degrees commanded bearing TRUE/REL */
public static double ST1000_command; //  degrees commanded bearing TRUE/REL

/** set to 3.0 in tank, 5.0 in open water */
public static double TARGETDISCRIMINATIONDIST; //

/** normal forward scan */
public static int SONARSCANSWATH; // normal forward scan

// (ST1000SCANWIDTH/2.0 degrees either side)
/** target edge scan (left or right) */
public static int SONARSCANEDGE; //

/** target tracking mode*/
public static int SONARSCANTRACK; //

/** locate target */
public static int SONARSCANLOCATE; // locate target

/** manually position */
public static int SONARSCANMANUAL; // manually position

/**
 * Control coefficients input
 * true = manual entry, false = default values */
public static boolean ENTERCONTROLCONSTANTS; //

/** true = print Control constants, false = default */
public static boolean SHOWCONTROLCONSTANTS; //

/** Load control coefficients, true = file entry, false = default values */
public static boolean LOADCONTROLCONSTANTS;

/** Pogram execition, true = real-time waits, false = no-pause */
public static boolean REALTIME;

/**  true = dive tracker being used */
public static boolean DIVETRACKER; //

/** true = dead reckon navigate, false = regular */
public static boolean DEADRECKON; //

/** true = use ordered rudder, false = control */
public static boolean DEADSTICKRUDDER; //

/** true = use ordered planes, false = control */
public static boolean DEADSTICKPLANES; //

/** 1 = use PID, 0 = use PD */
public static int INTEGRALDEPTHCONTROL; //

/** give PD a chance to get close */
```

```
public static double time_int_control_on; //  give PD a chance to get close

/** true = use sliding mode, false = control */
public static boolean SLIDINGMODECOURSE; //

/** true = use thrusters, false = use propellers */
public static boolean THRUSTERCONTROL; //

/** true = use thrusters to rotate in place */
public static boolean ROTATECONTROL; //

/** true = use thrusters for lateral motion */
public static boolean LATERALCONTROL; //

/** true = go to WAYPOINT without WAITs */
public static boolean FOLLOWWAYPOINTMODE; //

/** true = go to WAYPOINT */
public static boolean WAYPOINTCONTROL;  //

/**  true = hover at WAYPOINT */
public static boolean HOVERCONTROL; //

/** true = dock to WAYPOINT */
public static boolean DOCKINGCONTROL = false; //

/**  true = hover relative to a target */
public static boolean TARGETCONTROL; //

/** true = target is new, false = target is old */
public static boolean NEWTARGET; //

/**  true = point at target if TARGETCONTROL */
public static boolean TARGETPOINTING; //

/**  true = tracking on target edge only */
public static boolean TARGETEDGETRACK; //

/**  true = new station keeping point */
public static boolean NEWTARGETSTATION; //

/** true = recovery in progress */
public static boolean RECOVERYCONTROL; //

/**     true = command new, false = in progress */
public static boolean NEWRECOVERYCOMMAND; //

/**  true = followlight mode in progress */
public static boolean FOLLOWLIGHTCONTROL; //

/** degrees full scan, centered on bow */
public static double ST1000SCANWIDTH; //

/** degrees full scan, centered on bow */
public static double ST725SCANWIDTH; //
// 90 degrees takes ~15 seconds
```

```
// note that  MAX values clamp hydrodynamics inputs only, not vehicle orders.
//          0.0 values do not clamp
//        public static final double          MAX_RPM = 700;                // dual propellers
//        public static final double          MIN_THRUSTER = 0.0;                    // rpm
//        public static final double          MAX_PLANE = 22.5;             // degrees
//        public static final double          MAX_RUDDER = 22.5;            // degrees

/** true = water leak detected */
public static boolean LEAK; //

/** true = automatic shutdown criteria met */
public static boolean HALTSCRIPT; //

/** true = reset death spiral checker */
public static boolean DEA TH_SPIRAL_RESET; //

/** true = send e-mail, false = don't send e-mail */
public static boolean SEND_EMAIL; //

/** e-mail entered at the begining of execution */
public static boolean EMAIL_ENTERED; //

/** e-mail entered at the begining of execution */
public static boolean EMAIL; //

/** time of a single closed loop */
public static double TIMESTEP; //

/** true if benchtest in progress */
public static boolean BENCHTEST;

/** true = tactical level parsing commands */
public static boolean TACTICALPARSE; //

/** true = read commands from keyboard vice mission file */
public static boolean KEYBOARDINPUT; //

        /**  true = mission.script.HELP already shown */
public static boolean HELPFILELAUNCHED; //

/** true = wait GPS-FIX & restore z_command */
public static boolean GPSFIXINPROGRESS; //

/** true = tell when stable hover/waypt/gps */
public static boolean REPORTSTABLE; //

/** true = replay of existing telemetry file */
public static boolean REPLAY; //

/** true = replay of existing telemetry file */
public static boolean NOSCRIPT; //
//    with no accompanying script file

/** AUV estimate of SeaState */
public static int SEASTATE; //
```

```
public void Execution_Flags() {

    TRACE = true; //
    DISPLAYSCREEN = true; //
    LOCATIONLAB = true; //  virtual world is true, false is actual vehicle
    TACTICAL = false; //
    LOOPFOREVER = false; //  repeat execution indefinitely
    LOOPFILEBACKUP = true; //  backup files between replications
    PARALLELPORTTRACE = false; //  trace each char received at port
    ST1000INSTALLED = true; //  sonar head available for query
    ST725INSTALLED = true; //  sonar head available for query
    SONARTRACE = false; //  trace on or trace off
    ST1000SCANMODE = 1; //  WHERE IS THE ENUMERATION
    ST725SCANMODE = 1; //  WHERE IS THE ENUMERATION
    SONARHEADINGSTEP = 0.9; //  degrees
    ST1000SCANDIRECTION = 1; // "RIGHT";  "LEFT" = -1
    ST725SCANDIRECTION = -1;
    ST1000_command = 0.0; //  degrees commanded bearing TRUE/REL
    TARGETDISCRIMINATIONDIST = 5.0; //  set to 3.0 in tank, 5.0 in open water
    SONARSCANSWATH = 1; // normal forward scan
    // (ST1000SCANWIDTH/2.0 degrees either side)
    SONARSCANEDGE = 2; // target edge scan (left or right)
    SONARSCANTRACK = 3; // target tracking mode
    SONARSCANLOCATE = 4; // locate target
    SONARSCANMANUAL = 5; // manually position
    ENTERCONTROLCONSTANTS = false; //  true = manual entry, false = default values
    SHOWCONTROLCONSTANTS = false; //  true = print constants, false = default
    LOADCONTROLCONSTANTS = true; //  true = file entry, false = default values
    REALTIME = false; //  true = real-time waits, false = no-pause
    DIVETRACKER = false; //  true = dive tracker being used
    DEADRECKON = false; //  true = dead reckon navigate, false = regular
    DEADSTICKRUDDER = false; //  true = use ordered rudder, false = control
    DEADSTICKPLANES = false; //  true = use ordered planes, false = control
    INTEGRALDEPTHCONTROL = 0; //  1 = use PID, 0 = use PD
    time_int_control_on = 1000000.0; //  give PD a chance to get close
    SLIDINGMODECOURSE = false; //  true = use sliding mode, false = control
    THRUSTERCONTROL = false; //  true = use thrusters, false = use propellers
    ROTATECONTROL = false; //  true = use thrusters to rotate in place
    LATERALCONTROL = false; //  true = use thrusters for lateral motion
    FOLLOWWAYPOINTMODE = false; //  true = go to WAYPOINT without WAITs
    WAYPOINTCONTROL = false; //  true = go to WAYPOINT
    HOVERCONTROL = false; //  true = hover at WAYPOINT
    DOCKINGCONTROL = false; //  true = dock to WAYPOINT
    TARGETCONTROL = false; //  true = hover relative to a target
    NEWTARGET = false; //  true = target is new, 0=target is old
    TARGETPOINTING = false; //  true = point at target if TARGETCONTROL
    TARGETEDGETRACK = false; //  true = tracking on target edge only
    NEWTARGETSTATION = false; //  true = new station keeping point
    RECOVERYCONTROL = false; //  true = recovery in progress
    NEWRECOVERYCOMMAND = true; //              true = command new, false = in progress
    FOLLOWLIGHTCONTROL = false; //  true = followlight mode in progress
    ST1000SCANWIDTH = 60.0; //  degrees full scan, centered on bow
    ST725SCANWIDTH = 60.0; //  degrees full scan, centered on bow
    // 90 degrees takes ~15 seconds
    // note that  MAX values clamp hydrodynamics inputs only, not vehicle orders.
    //          0.0 values do not clamp
```

```
        LEAK = false; //  true = water leak detected
        HALTSCRIPT = false; //  true = automatic shutdown criteria met
        DEATH_SPIRAL_RESET = true; //  true = reset death spiral checker
        SEND_EMAIL = false; //  true = send e-mail, false = don't send e-mail
        EMAIL_ENTERED = false; //  e-mail entered at the begining of execution
        EMAIL = false; //  e-mail entered at the begining of execution
        TIMESTEP = 0.10; //  time of a single closed loop
        //  add code to warn if exceeded <<<<
        //  units are seconds
        BENCHTEST = false;
        TACTICALPARSE = false; //  true = tactical level parsing commands
        KEYBOARDINPUT = false; //  true = read keyboard vice mission file
        HELPFILELAUNCHED = false; //  true = mission.script.HELP already shown
        GPSFIXINPROGRESS = false; //  true = wait GPS-FIX & restore z_command
        REPORTSTABLE = false; //  true = tell when stable hover/waypt/gps
        REPLAY = false; //  true = replay of existing telemetry file
        NOSCRIPT = false; //  true = replay of existing telemetry  file
        //    with no accompanying script file
        SEASTATE = 2; //  AUV estimate of SeaState
        LEAK = false;
        HALTSCRIPT = false; // to perform shutdown procedure
        DEATH_SPIRAL_RESET = true;
        EMAIL = false;
        EMAIL_ENTERED = false;
    }
}           // end of Execution_Flags class
```

M. ST1000_SONAR.JAVA

```
/* ____________________________________________________________________________________________
  Title:          ST1000_sonar.java
  Description:
  Date:           23 May, 2002
  Project:        Execution Java Software for Autonomous Underwater Vehicle,
                  Naval Postgraduate School, Monterey, CA
  Compiler:       JDK 1.3.1
  Author:         Miguel A. Ayala
  Version:        1.0
  ____________________________________________________________________________________________
 */

package mil.navy.nps.auvAries.execution.hardware;

import mil.navy.nps.auvAries.execution.globals.Execution_Flags;
import mil.navy.nps.auvAries.execution.data_processing.Data_Processing;
import mil.navy.nps.auvAries.execution.vehicle.Vehicle;
import mil.navy.nps.auvAries.execution.kalman_filter.Kalman;
import mil.navy.nps.auvAries.execution.control.Control;

/**
*
*/
public class ST1000_sonar {
   Execution_Flags flags;
   Data_Processing data_Processing;
   Vehicle vehicle;
   Kalman kalman;
   Control control;
   private int LEFT = -1;
   private int RIGHT = 1;
   private final int TRUE = 1;
   private final int RELATIVE = 0;
   boolean SONARPINGED = false;
   public double ST1000_range_kal;
   private final double AUV_ST1000_x_offset;
   private final double AUV_ST1000_y_offset;
   private final double AUV_ST1000_z_offset;
   private boolean reset_sonar_filter;
   static double previous_range;
   static double start_bearing;
   static double end_bearing;
   static double range_accumulator;
   static int valid_return_count;
   static int no_return_count;
   static boolean scan_onto_target;
   static boolean update_target_data;
   double new_target_bearing;
   double new_target_range;
   double commanded_bearing_error;
   double sonar_return_x;
   double sonar_return_y;

   /**
```

```
 *
 */
public ST1000_sonar(Execution_Flags flagsRef, Data_Processing Data_ProcessingRef,
   Vehicle vehicleRef, Kalman kalmanRef, Control controlRef) {

      flags = flagsRef;
      vehicle = vehicleRef;
      data_Processing = Data_ProcessingRef;
      kalman = kalmanRef;
      control = controlRef;
      AUV_ST1000_x_offset = 2.875;
      AUV_ST1000_y_offset = -0.16666667;
      AUV_ST1000_z_offset = 0.33333333;
      previous_range = 0.0;
      start_bearing = 0.0;
      end_bearing = 0.0;
      range_accumulator = 0.0;
      valid_return_count = 0;
      no_return_count = 0;
      scan_onto_target = false;
      update_target_data = false;
      new_target_bearing = 0.0;
      new_target_range = 0.0;
      commanded_bearing_error = 0.0;
      sonar_return_x = 0.0;
      sonar_return_y = 0.0;
}

/**
 *
 */
public void control_ST1000_sonar() {
   if (!flags.ST1000INSTALLED) {
      return;
   }
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[Begin control_ST1000_sonar]\n");
   }

   /*
    Sonar Scan Modes
    1: SONARSCANSWATH normal forward scan     (ST1000SCANWIDTH/2.0 degrees either side)
    2: SONARSCANEDGE  target edge scan (either left or right)
    3: full target scan
    4: locate target
    5: manually position
   */
   if ((SONARPINGED) && (!flags.REPLAY)) {
      vehicle.set_auv_ST1000_range(read_ST1000_sonar());
      if ((Math.abs((vehicle.get_auv_ST1000_range() - ST1000_range_kal)) >=
         flags.TARGETDISCRIMINATIONDIST) ||
         (vehicle.get_auv_ST1000_range() <= 0.001) || (ST1000_range_kal <= 0.001))
            reset_sonar_filter = true;
                  kalman.ST1000_range_kal (  vehicle.get_auv_ST1000_range() );

      SONARPINGED = false;
```

```
        // if ( !flags.REPLAY )
        // this writes to a file
//                      fprintf (st1000datafile,"%6.1lf %6.3lf %6.3lf %6.3lf %6.3lf %6.3lf\n",
        //              vehicle.get_time(), vehicle.get_auv_ST1000_range(),
        //              ST1000_range_kal, vehicle.get_auv_ST1000_bearing(),
        //              ST1000_range_kal * Math.cos ( Math.toRadians (
        //              vehicle.get_auv_ST1000_bearing() )),
        //              ST1000_range_kal * Math.sin ( Math.toRadians (
        //              vehicle.get_auv_ST1000_bearing() )));
      }
      if (flags.ST1000SCANMODE == flags.SONARSCANSWATH) {
        /* Normal Straight Ahead Scan Mode to Detect Collisions */

        if (flags.TRACE && flags.DISPLAYSCREEN) {
          System.out.println("[Sonar using normal forward scan]\n");
        }

                                        /* See if it is time to reverse scan direction */

        if (((flags.ST1000SCANDIRECTION == RIGHT) || (flags.ST1000SCANDIRECTION == 0)) &&
          (data_Processing.normalize2(vehicle.get_auv_ST1000_bearing()) >=
          data_Processing.normalize2(flags.ST1000SCANWIDTH / 2.0)))
            flags.ST1000SCANDIRECTION = LEFT;
        else if (((flags.ST1000SCANDIRECTION == LEFT) ||
  (flags.ST1000SCANDIRECTION == 0)) &&
          (data_Processing.normalize2(vehicle.get_auv_ST1000_bearing()) <=
          -data_Processing.normalize2(flags.ST1000SCANWIDTH / 2.0)))
            flags.ST1000SCANDIRECTION = RIGHT;
        else if (flags.ST1000SCANDIRECTION == 0)
          flags.ST1000SCANDIRECTION = RIGHT;
        ping_ST1000_sonar(flags.ST1000SCANDIRECTION);
      }
      else if (flags.ST1000SCANMODE == flags.SONARSCANEDGE) {
        /* Edge Tracking Mode */

        if (flags.TRACE && flags.DISPLAYSCREEN) {
          System.out.println("[Sonar in edge tracking mode]\n");
        }

                                // Decide whether or not ping was on the target or not

        if ((ST1000_range_kal > 0.0) &&
        (Math.abs(ST1000_range_kal - previous_range) < flags.TARGETDISCRIMINATIONDIST)) {
          if (((valid_return_count == 0) && (scan_onto_target)) ||
            (scan_onto_target == false)) {
            start_bearing = data_Processing.normalize(vehicle.get_psi() +
              vehicle.get_auv_ST1000_bearing());
          }
          previous_range = ST1000_range_kal;
          no_return_count = 0;
          range_accumulator += ST1000_range_kal;
          valid_return_count++;
        }
        else {
          no_return_count++;
        }
```

```
                // Check to see if it is time to reverse the scan and compute numbers

if (scan_onto_target) {
   if ((valid_return_count > 2) || ((valid_return_count > 0) &&
      (no_return_count > 2))) {
      update_target_data = true;
      scan_onto_target = false;
   }
}
else {
   if (no_return_count > 2) {
      update_target_data = true;
      scan_onto_target = true;
   }
}
if (Math.abs(data_Processing.normalize2(data_Processing.normalize(
                vehicle.get_auv_ST1000_bearing() + vehicle.get_psi()) -
   control.target_bearing)) > 90.0) {
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("[Lost Target, Switching to Target Search Sonar Mode]\n");
      }
      scan_onto_target = true;
      flags.ST1000SCANDIRECTION = -flags.ST1000SCANDIRECTION;
      flags.ST1000SCANMODE = flags.SONARSCANLOCATE;
}
if (update_target_data) {
   if (valid_return_count > 0) {
                                                // Compute a sonar range and bearing

      control.target_bearing = start_bearing;
      new_target_range = range_accumulator / (double)valid_return_count;

                                        // Determine location of target in world coordinates

      control.target_x = vehicle.get_x() + control.cos_psi *
         AUV_ST1000_x_offset /* x position of sonar */
      - control.sin_psi * AUV_ST1000_y_offset;
      control.target_y = vehicle.get_y() + control.sin_psi *
         AUV_ST1000_x_offset /* y position of sonar */
      + control.cos_psi * AUV_ST1000_y_offset;
      control.target_x = control.target_x + Math.cos(
         Math.toRadians(control.target_bearing)) * new_target_range;
      control.target_y = control.target_y + Math.sin(
         Math.toRadians(control.target_bearing)) * new_target_range;

                                        // Determine location of vehicle center relative to target

      control.target_bearing = data_Processing.normalize
         (Math.toDegrees((data_Processing.atan2z
         ((control.target_y - vehicle.get_y()), (control.target_x -
         vehicle.get_x()))))));
      new_target_range = Math.sqrt((vehicle.get_x() - control.target_x) *
         (vehicle.get_x() - control.target_x) +
         (vehicle.get_y() - control.target_y) * (vehicle.get_y() -
         control.target_y));
```

```
            if (flags.TRACE && flags.DISPLAYSCREEN) {
               System.out.println("[Target X Location: " +
                  control.target_x + "]\n");
               System.out.println("[Target Y Location: " +
                  control.target_y + "]\n");
               System.out.println("[Target Bearing: " +
                  control.target_bearing + "]\n");
               System.out.println("[Target Range: " +
                  control.target_range + "]\n");
            }
            control.target_range = new_target_range;
            control.new_target_update = true;
            if (scan_onto_target) {
               range_accumulator = 0.0;
               valid_return_count = 0;
            }
            else {
               range_accumulator = previous_range;
               valid_return_count = 1;
            }
            no_return_count = 0;
            flags.ST1000SCANDIRECTION = -1; // reverse direction
            control.time_last_target_update = vehicle.get_time();

                                                   // this writes to a file
            //       fprintf (targetdatafile,"%lf %lf %lf %lf %lf %lf %lf\n",
            //       vehicle.get_time(), control.target_range,
            //       control.target_bearing, vehicle.get_psi(),
            //       control.target_range_command, control.target_bearing_command,
            //  control.psi_command_tgt );

            if (flags.TRACE && flags.DISPLAYSCREEN) {
               System.out.println("[Target Update:  " +
                  control.target_range + ", " + control.target_bearing + "]\n");
            }
         }
         update_target_data = false;
      }
      if ((scan_onto_target) || (no_return_count == 0))
         ping_ST1000_sonar(flags.ST1000SCANDIRECTION);
      else
         ping_ST1000_sonar(0);
   }
   else if (flags.ST1000SCANMODE == flags.SONARSCANTRACK) {
      /* Target Tracking Mode */

      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("[Sonar in target tracking mode]\n");
      }

                                       /* See if it is time to reverse scan direction */

      if ((ST1000_range_kal > 0.0) && ((ST1000_range_kal - previous_range) <= 5.0)) {
         end_bearing = data_Processing.normalize(vehicle.get_psi() +
            vehicle.get_auv_ST1000_bearing());
```

```
    if (valid_return_count == 0) {
        start_bearing = end_bearing;
    }
    previous_range = ST1000_range_kal;
    no_return_count = 0;
    range_accumulator += ST1000_range_kal;
    valid_return_count += 1;
    ping_ST1000_sonar(flags.ST1000SCANDIRECTION);
}
else {
    if (no_return_count > 3) { // scanned off end of target
        if (valid_return_count > 0) {
                                    // Compute a sonar range and bearing

            new_target_bearing = data_Processing.normalize
                (start_bearing + data_Processing.normalize2(end_bearing -
                            start_bearing) / 2.0);
            new_target_range = range_accumulator / (double)valid_return_count;

                                        // Determine location of target in world coordinates

            control.target_x = vehicle.get_x() + control.cos_psi *
                AUV_ST1000_x_offset // x position of sonar *
                -control.sin_psi * AUV_ST1000_y_offset;
            control.target_y = vehicle.get_y() + control.sin_psi *
                AUV_ST1000_x_offset // y position of sonar
                + control.cos_psi * AUV_ST1000_y_offset;
            control.target_x = control.target_x + Math.cos(
                Math.toRadians(new_target_bearing)) * new_target_range;
            control.target_y = control.target_y + Math.sin(
                Math.toRadians(new_target_bearing)) * new_target_range;

                                        // Determine location of vehicle center relative to target

            control.target_bearing = data_Processing.normalize
                (Math.toDegrees(data_Processing.atan2z
                ((control.target_y - vehicle.get_y()), (control.target_x -
                vehicle.get_x()))));
            new_target_range = Math.sqrt((vehicle.get_x() - control.target_x) *
                (vehicle.get_x() - control.target_x) + (vehicle.get_y() -
                control.target_y) *
                (vehicle.get_y() - control.target_y));
            if (flags.TRACE && flags.DISPLAYSCREEN) {
                System.out.println("[Target X Location: " +
                    control.target_x + "]\n");
                System.out.println("[Target Y Location: " +
                    control.target_y + "]\n");
                System.out.println("[Target Bearing: " +
                    control.target_bearing + "]\n");
                System.out.println("[Target Range: " +
                    control.target_range + "]\n");
            }
            control.target_range = new_target_range;
            control.new_target_update = true;
            System.out.println("[Target Update:  " + control.target_range +
                ", " + control.target_bearing + "]\n");
```

```
            // THIS WRITES to file
            //  fprintf (targetdatafile,"%lf %lf %lf %lf %lf %lf %lf\n",
            //     vehicle.get_time(), control.target_range, control.target_bearing,
            //  vehicle.get_psi(), control.target_range_command,
            //  control.target_bearing_command, control.psi_command_tgt );

            control.time_last_target_update = vehicle.get_time();
            control.new_target_update = true;
            no_return_count = 0;
            valid_return_count = 0;
            range_accumulator = 0.0;
            flags.ST1000SCANDIRECTION = -1;

            /* reverse direction */

            ping_ST1000_sonar(flags.ST1000SCANDIRECTION);
         }
         else {
            /* backtrack if scanned more than one step off target */

            ping_ST1000_sonar(flags.ST1000SCANDIRECTION);
            no_return_count = 0;
         }
      }
      else {
         ping_ST1000_sonar(0);
         ++no_return_count;

         /* try again before reversing scan */
      }
   }
}

      // Find Target Based On Tactical Range and Bearing
else if (flags.ST1000SCANMODE == flags.SONARSCANLOCATE) {
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[Sonar searching for target]\n");
   }
   previous_range = 0.0;
   control.time_last_target_update = 0.0;
   if (flags.TRACE && (ST1000_range_kal > 0.0)) {
      System.out.println("ST1000_range_kal: " + ST1000_range_kal +
         "\nSonar Bearing: " +
         data_Processing.normalize(vehicle.get_auv_ST1000_bearing() +
         vehicle.get_psi()));
      System.out.println("Target Range: " + control.target_range +
         "\nTarget Bearing: " + control.target_bearing);
   }
   if (ST1000_range_kal > 0.0) {

      // Determine location of return in world coordinates

      sonar_return_x = vehicle.get_x() + control.cos_psi *
         AUV_ST1000_x_offset /* x position of sonar */
      - control.sin_psi * AUV_ST1000_y_offset;
```

```
        sonar_return_y = vehicle.get_y() + control.sin_psi *
            AUV_ST1000_x_offset /* y position of sonar */
        + control.cos_psi * AUV_ST1000_y_offset;
        sonar_return_x += Math.cos(Math.toRadians
            (vehicle.get_psi() + vehicle.get_auv_ST1000_bearing())) *
            ST1000_range_kal;
        sonar_return_y += Math.sin(Math.toRadians
            (vehicle.get_psi() + vehicle.get_auv_ST1000_bearing())) *
            ST1000_range_kal;
        // Determine location of vehicle center relative to target *
        new_target_bearing = data_Processing.normalize
            (Math.toDegrees(data_Processing.atan2z
            ((sonar_return_y - vehicle.get_y()), (sonar_return_x -
            vehicle.get_x()))));
        new_target_range = Math.sqrt((vehicle.get_x() - sonar_return_x) *
            (vehicle.get_x() - sonar_return_x) +
            (vehicle.get_y() - sonar_return_y) * (vehicle.get_y() - sonar_return_y));
        // Determine if Target Located
        if ((Math.abs(new_target_range - control.target_range) <= 5.0) &&
            (Math.abs(data_Processing.normalize2
            (data_Processing.normalize(new_target_bearing) -
                    control.target_bearing)) <= 15.0)) {
            if (flags.TARGETEDGETRACK) {
                flags.ST1000SCANMODE = flags.SONARSCANEDGE; // use target edge scan
                // scan towards edge on side move is towards //
                flags.ST1000SCANDIRECTION = (int)data_Processing.dsign
                    (data_Processing.normalize2(control.target_bearing -
                            control.target_bearing_command));
                if (flags.DISPLAYSCREEN)
                    if (flags.ST1000SCANDIRECTION == 1)
                        System.out.println("Sonar Scanning Right To Obtain Edge\n");
                    else
                        System.out.println("Sonar Scanning LEFT To Obtain Edge\n");
            }
            else {
                flags.ST1000SCANMODE = flags.SONARSCANTRACK; // use full target track scan
            }
            previous_range = ST1000_range_kal;
            start_bearing = data_Processing.normalize(vehicle.get_psi() +
                vehicle.get_auv_ST1000_bearing());
            no_return_count = 0;
            scan_onto_target = false;
            range_accumulator = ST1000_range_kal;
            valid_return_count = 1;
            if (flags.DISPLAYSCREEN)
                // VERIFY THIS SYSTEM.OUT
                    System.out.println("[Target Located: " +
                        data_Processing.normalize(vehicle.get_psi() +
                        vehicle.get_auv_ST1000_bearing()) +
                        " degrees" + data_Processing.normalize(ST1000_range_kal) +
                        " feet]\n");
        }
    }
    ping_ST1000_sonar(flags.ST1000SCANDIRECTION);
}
else if (flags.ST1000SCANMODE == flags.SONARSCANMANUAL) {
```

```
      if (vehicle.get_auv_ST1000_direction() == TRUE) {
         commanded_bearing_error = data_Processing.normalize2
            (control.ST1000_command - vehicle.get_auv_ST1000_bearing() -
            data_Processing.normalize(vehicle.get_psi()));
      }
      else if (vehicle.get_auv_ST1000_direction() == RELATIVE) {
         commanded_bearing_error = data_Processing.normalize2
            (control.ST1000_command - vehicle.get_auv_ST1000_bearing());
      }
      else {
         if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[error in AUV_ST1000_direction, assume RELATIVE]\n");
         }
         commanded_bearing_error = data_Processing.normalize2
            (control.ST1000_command - vehicle.get_auv_ST1000_bearing());
      }
      if (commanded_bearing_error > flags.SONARHEADINGSTEP / 2.0) {
         flags.ST1000SCANDIRECTION = RIGHT;
      }
      else if (commanded_bearing_error < -flags.SONARHEADINGSTEP / 2.0) {
         flags.ST1000SCANDIRECTION = LEFT;
      }
      else
         flags.ST1000SCANDIRECTION = 0;
      if (flags.TRACE && flags.DISPLAYSCREEN) {
         System.out.println("[ST1000 sonar in manual scan mode]\n");
         System.out.println("[AUV_ST1000_bearing      = " +
            vehicle.get_auv_ST1000_bearing() + "]\n");
         System.out.println("[Command ST1000_bearing  = " +
            control.ST1000_command + "]\n");
         System.out.println("[psi                     = " +
            vehicle.get_psi() + "]\n");
         System.out.println("[normalize (psi)         = " +
            data_Processing.normalize(vehicle.get_psi()) + "]\n");
         System.out.println("[commanded_bearing_error = " +
            commanded_bearing_error + "]\n");
         System.out.println("[ST1000SCANDIRECTION     = " +
            flags.ST1000SCANDIRECTION + "]\n");
      }
      ping_ST1000_sonar(flags.ST1000SCANDIRECTION);
   }
   else // Invalid Mode
      if (flags.DISPLAYSCREEN) System.out.println("[Invalid sonar scan mode! ***]\n");
   if (flags.TRACE && flags.DISPLAYSCREEN)
      System.out.println("[End control_ST1000_sonar]\n");
   return;
} // end of control_ST1000_sonar

/**
 *
 */
public void initialize_ST1000_sonar() {
   System.out.println("ST1000 initialization");
}

/**
```

```
 *
 */
public void center_ST1000_sonar() {
   flags.TRACE = true;
   if (flags.ST1000INSTALLED == false) {
      return;
   }
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[Begin center_ST1000_sonar]\n");
   }
   vehicle.set_auv_ST1000_bearing(0.0);
   if (!flags.LOCATIONLAB) {
      System.out.println("Physical world code");
   }
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[End center_ST1000_sonar]\n");
   }
   return;
} // end of center_ST1000_sonar

/**
 *
 */
public void step_ST1000_sonar(int direction) {
   if (!flags.ST1000INSTALLED) {
      return;
   }
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[Begin step_ST1000_sonar]\n");
   }
   if (direction == LEFT) {
      vehicle.set_auv_ST1000_bearing(data_Processing.normalize(vehicle.get_auv_ST1000_bearing() -
         flags.SONARHEADINGSTEP));
   }
   else if (direction == RIGHT) {
      vehicle.set_auv_ST1000_bearing(data_Processing.normalize(vehicle.get_auv_ST1000_bearing() +
         flags.SONARHEADINGSTEP));
   }
   if (vehicle.get_auv_ST1000_bearing() < 0.9 || vehicle.get_auv_ST1000_bearing() > 359.1) {
      vehicle.set_auv_ST1000_bearing(0.0);
   }
   if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[Step direction: " + direction + "]\n");
      System.out.println("[New ST1000 Bearing: " + vehicle.get_auv_ST1000_bearing() + " ]\n");
      System.out.println("[End step_ST1000_sonar]\n");
   }
} // end of step_ST1000_sonar()

/**
 *
 */
public void ping_ST1000_sonar(int direction) {
   if (!flags.ST1000INSTALLED) {
      return;
   }
   if (flags.TRACE && flags.DISPLAYSCREEN) {
```

```
            System.out.println("[Begin ping_ST1000_sonar]\n");
        }
        // HERE GOES the Physical world code
        if ((direction == LEFT) || (direction == RIGHT) || (direction == 0)) {
            SONARPINGED = true;
            step_ST1000_sonar(direction);
        }
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[End ping_ST1000_sonar]\n");
        }
    } // end of ping_ST1000_sonar

    /**
     *
     */
    public double read_ST1000_sonar() {
        if (!flags.ST1000INSTALLED) {
            return 0.0;
        }
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[Begin read_ST1000_sonar]\n");
        }
        // HERE Physical world code
        if (vehicle.get_auv_ST1000_range() > 32.808) {
            vehicle.set_auv_ST1000_range(0.0);
        }
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[AUV_ST1000_range = " + vehicle.get_auv_ST1000_range() + "]\n");
            System.out.println("[End read_ST1000_sonar]\n");
        }
        return (vehicle.get_auv_ST1000_range());
    } // end of public double read_ST1000_sonar ()

    /**
     *
     */
    public void set_ST1000_step_size(int step_code) {
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[Begin set_ST1000_step_size()]");
        }
        if (flags.TRACE && flags.DISPLAYSCREEN) {
            System.out.println("[End set_ST1000_step_size()]");
        }
        return;
    }
} // end of ST1000_sonar class
```

N. ST725_SONAR.JAVA

```
/* ____________________________________________________________________________________
  Title:          ST725_sonar.java
  Description:
  Date:           23 May, 2002
  Project:        Execution Java Software for Autonomous Underwater Vehicle,
                  Naval Postgraduate School, Monterey, CA
  Compiler:       JDK 1.3.1
  Author:         Miguel A. Ayala
  Version:        1.0
  ____________________________________________________________________________________
 */

package mil.navy.nps.auvAries.execution.hardware;

import mil.navy.nps.auvAries.execution.globals.Execution_Flags;
import mil.navy.nps.auvAries.execution.data_processing.Data_Processing;
import mil.navy.nps.auvAries.execution.vehicle.Vehicle;

/**
*
*/
public class ST725_sonar {
   private Execution_Flags flags;
   private Data_Processing Data_Processing;
   private Vehicle vehicle;
   public double auv_ST725_x_offset;
   //    private final double SONARHEADINGSTEP = 0.9; // degrees
   private int LEFT = -1;
   private int RIGHT = 1;
   private boolean SONARPINGED = false;

   /**
    * Constructor
    */
   public ST725_sonar(Execution_Flags flagsRef, Data_Processing Data_ProcessingRef,
      Vehicle vehicleRef) {

      flags = flagsRef;
      vehicle = vehicleRef;
      Data_Processing = Data_ProcessingRef;
      flags.ST725INSTALLED = true;
      flags.TACTICAL = true;

   } // end of ST725 constructor

   /**
    */
   public void control_ST725_sonar() {
      private double start_bearing = 0.0;
      private double end_bearing = 0.0;
      private double range_accumulator = 0.0;
      private int valid_return_count = 0.0;
      private boolean scan_onto_target = false;
```

```
    private boolean update_target_data = false;
    private double new_target_bearing;
    private double new_target_range;
    private double commanded_target_error;
    private double sonar_return_x;
    private double sonar_return_y;
*/

    if (!flags.ST725INSTALLED) {
      return;
    }
    if (!flags.TACTICAL) {
      return;
    }
    if (flags.TRACE) {
      System.out.println("[Begin control ST725_sonar]\n");
    }
    if (flags.ST725SCANMODE == flags.SONARSCANSWATH) {
      /* Normal Straight Ahead Scan Mode to Detect Collisions */

      if (flags.TRACE && flags.DISPLAYSCREEN) {
        System.out.println("[ST725 sonar using normal forward scan]\n");
      }

     /* See if it is time to reverse scan direction */

      if (((flags.ST725SCANDIRECTION == RIGHT) || (flags.ST725SCANDIRECTION == 0)) &&
        (Data_Processing.normalize2(vehicle.get_auv_ST725_bearing()) >=
        Data_Processing.normalize2(flags.ST725SCANWIDTH / 2.0))) {
          flags.ST725SCANDIRECTION = LEFT;
      }
      else if (((flags.ST725SCANDIRECTION == LEFT) || (flags.ST725SCANDIRECTION == 0)) &&
        (Data_Processing.normalize2(vehicle.get_auv_ST725_bearing()) <=
        -Data_Processing.normalize2(flags.ST725SCANWIDTH / 2.0))) {
          flags.ST725SCANDIRECTION = RIGHT;
      }
      else if (flags.ST725SCANDIRECTION == 0) {
        flags.ST725SCANDIRECTION = RIGHT;
      }
      vehicle.set_auv_ST725_bearing(vehicle.get_auv_ST725_bearing() +
        flags.ST725SCANDIRECTION * flags.SONARHEADINGSTEP);
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[ST725 sonar in manual scan mode by execution level]\n");
      System.out.println("[AUV_ST725_bearing " + vehicle.get_auv_ST725_bearing() + "]\n");
      System.out.println("[ST725SCANWIDTH " + flags.ST725SCANWIDTH + "]\n");
      System.out.println("[ST725SCANDIRECTION " + flags.ST725SCANDIRECTION + "]\n");
    }
    if (flags.TRACE && flags.DISPLAYSCREEN) {
      System.out.println("[End control_ST725_sonar]\n");
    }
    return;
  } // end of control_ST725_sonar

} // end of ST725 class
```

O. KALMAN.JAVA

```
/* ____________________________________________________________________________________________
   Title:          Kalman.java
   Description:            Kalman Filter implementations
   Date:           23 May, 2002
   Project:        Execution Java Software for Autonomous Underwater Vehicle,
                   Naval Postgraduate School, Monterey, CA
   Compiler:       JDK 1.3.1
   Author:         Miguel A. Ayala
   Version:        1.0
   ____________________________________________________________________________________________
 */

package mil.navy.nps.auvAries.execution.kalman_filter;

import mil.navy.nps.auvAries.execution.globals.Execution_Flags;

/**
* This class provides kalman filters implemetations
*/
public class Kalman {
   public boolean kal_init_z;
   public double thres_z;
   public double z_kal;
   public double z_dot_kal;
   public double z_ddot_kal;
   public double ST725_range_kal;
   public double ST725_range_kal_dot;
   public double ST725_range_kal_ddot;
   public double res;
   public double yk;
   public double xk1_0, xk1_1, xk1_2;
   public double phii[] [], h[], b[], lk[];
   public double ST1000_range_kal;
   public double ST1000_range_kal_dot;
   public double ST1000_range_kal_ddot;
   public boolean reset_sonar_filter;
   private Execution_Flags flags;


   /**
    * Contructor
    * @param:      None
    * @return: None
    */
   public Kalman() {
      kal_init_z = true;
      thres_z = 1.0;
      z_kal = 0.0;
      z_dot_kal = 0.0;
      z_ddot_kal = 0.0;
      ST725_range_kal_dot = 0.0;
      ST725_range_kal_ddot = 0.0;
      ST1000_range_kal = 0.0;
      ST1000_range_kal_dot = 0.0;
```

```
    ST1000_range_kal_ddot = 0.0;
    res = 0.0;
    reset_sonar_filter = false;
}

/**
 * Performs kalman filtering for the sonar 725
 *  @param:      double
 *  @return:     None
 */
public void kalman_sonar725(double ST725_range) {
    yk = ST725_range;
    phii = new double[3] [3];
    h = new double[3];
    b = new double[3];
    lk = new double[3];

                    /* a=[0 1 0;0 0 1;0 0 0]; phii=expm(a*0.1); where dt = 0.1 */

    b[0] = 0.0;
    b[1] = 0.0;
    b[2] = 1.0;

          /* phii = [1.0    0.1    0.005
                    0.0    1.0     0.1
                    0.0    0.0     1.0] */

    phii[0] [0] = 1.0;
    phii[0] [1] = 0.1;
    phii[0] [2] = 0.005;
    phii[1] [0] = 0.0;
    phii[1] [1] = 1.0;
    phii[1] [2] = 0.1;
    phii[2] [0] = 0.0;
    phii[2] [1] = 0.0;
    phii[2] [2] = 1.0;

                    /* h = [1 0 0]; */

    h[0] = 1.0;
    h[1] = 0.0;
    h[2] = 0.0;
    if (flags.NEWRECOVERYCOMMAND) {
        ST725_range_kal = yk;
        ST725_range_kal_dot = 0.0;
        ST725_range_kal_ddot = 0.0;

                    /* xk1=xk; */

        xk1_0 = ST725_range_kal;
        xk1_1 = ST725_range_kal_dot;
        xk1_2 = ST725_range_kal_ddot;
    }

                    /* set lk = const. Slow Filter */
```

```
    lk[0] = 0.2544;
    lk[1] = 0.3727;
    lk[2] = 0.2731;

                    /* xk1(:,i)=phii*xk(:,i); */

    xk1_0 = phii[0] [0] * ST725_range_kal + phii[0] [1] * ST725_range_kal_dot
       + phii[0] [2] * ST725_range_kal_ddot;
    xk1_1 = phii[1] [0] * ST725_range_kal + phii[1] [1] * ST725_range_kal_dot
       + phii[1] [2] * ST725_range_kal_ddot;
    xk1_2 = phii[2] [0] * ST725_range_kal + phii[2] [1] * ST725_range_kal_dot
       + phii[2] [2] * ST725_range_kal_ddot;
    res = yk - (h[0] * xk1_0 + h[1] * xk1_1 + h[2] * xk1_2);

                    /*  Set res = 0.0 if larger than threshold */

    if (Math.abs(res) > thres_z) {
       res = 0.0;
    }
    ST725_range_kal = xk1_0 + lk[0] * res;
    ST725_range_kal_dot = xk1_1 + lk[1] * res;
    ST725_range_kal_ddot = xk1_2 + lk[2] * res;
    return;
 } // end of public kalman_sonar725 ( double ST725_range )

 /**
  * Performs kalman filtering for the sonar 1000
  *  @param:      double
  *  @return:     None
  */

 public void kalman_sonar1000(double ST1000_range) {
    double yk = ST1000_range;
    double xk1_0, xk1_1, xk1_2;
    double res;
    phii = new double[3] [3];
    h = new double[3];
    b = new double[3];
    lk = new double[3];

                    /* a=[0 1 0;0 0 1;0 0 0]; phii=expm(a*0.1); where dt = 0.1 */

    b[0] = 0.0;
    b[1] = 0.0;
    b[2] = 1.0;

/* phii = [1.0    0.1    0.005
           0.0    1.0     0.1
           0.0    0.0     1.0] */

    phii[0] [0] = 1.0;
    phii[0] [1] = 0.1;
    phii[0] [2] = 0.005;
    phii[1] [0] = 0.0;
    phii[1] [1] = 1.0;
    phii[1] [2] = 0.1;
```

```
    phii[2] [0] = 0.0;
    phii[2] [1] = 0.0;
    phii[2] [2] = 1.0;

                    /* h = [1 0 0]; */

    h[0] = 1.0;
    h[1] = 0.0;
    h[2] = 0.0;
    if (reset_sonar_filter) {
        ST1000_range_kal = yk;
        ST1000_range_kal_dot = 0.0;
        ST1000_range_kal_ddot = 0.0;

                            /* xk1=xk; */

        xk1_0 = ST1000_range_kal;
        xk1_1 = ST1000_range_kal_dot;
        xk1_2 = ST1000_range_kal_ddot;
        reset_sonar_filter = false;
    }

                    /* set lk = const. Slow Filter */

    lk[0] = 0.2544;
    lk[1] = 0.3727;
    lk[2] = 0.2731;

                    /* xk1(:,i)=phii*xk(:,i); */

    xk1_0 = phii[0] [0] * ST1000_range_kal + phii[0] [1] * ST1000_range_kal_dot
        + phii[0] [2] * ST1000_range_kal_ddot;
    xk1_1 = phii[1] [0] * ST1000_range_kal + phii[1] [1] * ST1000_range_kal_dot
        + phii[1] [2] * ST1000_range_kal_ddot;
    xk1_2 = phii[2] [0] * ST1000_range_kal + phii[2] [1] * ST1000_range_kal_dot
        + phii[2] [2] * ST1000_range_kal_ddot;
    res = yk - (h[0] * xk1_0 + h[1] * xk1_1 + h[2] * xk1_2);

                    /*  Set res = 0.0 if larger than threshold */

    if (Math.abs(res) > 5.0) {
        res = 0.0;
    }
    ST1000_range_kal = xk1_0 + lk[0] * res;
    ST1000_range_kal_dot = xk1_1 + lk[1] * res;
    ST1000_range_kal_ddot = xk1_2 + lk[2] * res;
    return;
}

/**
 * Performs kalman filtering for depth
 *  @param:      double
 *  @return:     None
 */
public void kalman_z(double yk) {
    double xk1_0, xk1_1, xk1_2;
```

```
 phii = new double[3] [3];
 h = new double[3];
 b = new double[3];
 lk = new double[3];

          /* a=[0 1 0;0 0 1;0 0 0]; phii=expm(a*0.1); where dt = 0.1 */

 b[0] = 0.0;
 b[1] = 0.0;
 b[2] = 1.0;

        /* phii = [1.0    0.1    0.005
                   0.0    1.0      0.1
                   0.0    0.0      1.0] */

 phii[0] [0] = 1.0;
 phii[0] [1] = 0.1;
 phii[0] [2] = 0.005;
 phii[1] [0] = 0.0;
 phii[1] [1] = 1.0;
 phii[1] [2] = 0.1;
 phii[2] [0] = 0.0;
 phii[2] [1] = 0.0;
 phii[2] [2] = 1.0;

/* h = [1 0 0]; */

 h[0] = 1.0;
 h[1] = 0.0;
 h[2] = 0.0;
 if (kal_init_z) {
    z_kal = yk;
    z_dot_kal = 0.0;
    z_ddot_kal = 0.0;

   /* xk1=xk; */

    xk1_0 = z_kal;
    xk1_1 = z_dot_kal;
    xk1_2 = z_ddot_kal;
    kal_init_z = false;
 }

/* set lk = const. Slow Filter */

 lk[0] = 0.2544;
 lk[1] = 0.3727;
 lk[2] = 0.2731;

/* xk1(:,i)=phii*xk(:,i); */

 xk1_0 = phii[0] [0] * z_kal + phii[0] [1] * z_dot_kal + phii[0] [2] * z_ddot_kal;
 xk1_1 = phii[1] [0] * z_kal + phii[1] [1] * z_dot_kal + phii[1] [2] * z_ddot_kal;
 xk1_2 = phii[2] [0] * z_kal + phii[2] [1] * z_dot_kal + phii[2] [2] * z_ddot_kal;
 res = yk - (h[0] * xk1_0 + h[1] * xk1_1 + h[2] * xk1_2);
```

```
                    /*  Set res = 0.0 if larger than threshold */

      if (Math.abs(res) > thres_z) {
         res = 0.0;
      }
      z_kal = xk1_0 + lk[0] * res;
      z_dot_kal = xk1_1 + lk[1] * res;
      z_ddot_kal = xk1_2 + lk[2] * res;
      return;
   } // end of kalman_z
} // end of class Kalman
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B COMPACT DISK

A. INTRODUCTION

This section briefly describes the contents of the compact disk (cd) appendix.

B. CD CONTENTS

The appendix cd has all the source code developed in this project. The javadoc for the source code is provided. The java.realtime extension of the Java language is provided with the respective javadoc. Finally, the thesis document is included.

The cd is organized in the following directories:

- AUV Execution Java Software
- Execution Javadoc
- Real-Time Java
- Thesis

A readme file provides an overview of the cd.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Badr, Salah M, Byrnes, Ronald B, Brutzman, Donald P., Nelson, Michael L., *Real-Time Systems*, technical report NPS-CS-92-004, Naval Postgraduate School, Monterey California, February 1992.

2. Behforooz, Ali, Hudsun, Frederick J., *Software Engineerin Fundamentals*, Oxford University Press, Oxford New York, 1996.

3. Brutzman, Donald P., *A Virtual World for an Autonomous UnderwaterVehicle*, Ph.D. Dissertation, Naval Postgraduate School, Monterey California, December 1994.

4. Burns, Michael L.,*Merging Virtual and Real Execution Level Control Software for the Phoenix Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1996.

5. Byrne, Kevin Michael, *Real-Time Modeling of Cross-Body Flow for Torpedo Tube Recoveryof the PHOENIX Autonomous Underwater Vehicle (AUV)*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1998.

6. Carnahan, Lisa, Ruark, Marcus, *Requirements for Real-time Extensions for the Java™ Platform: Report from the Requirments Group for Real-time Extensions For the Java™ Platform*, U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, September 1999.

7. Dewitz, Sandra D., *Systems Analysis and Design and the Transition to Objects*, McGraw-Hill, New York, 1996.

8. Dietel, H.M, Dietel, P.J., *Java How to Program*, third edition, Prentice Hall, New Jersey, 1999.

9. Douglass, Bruce Powel, *Doing Hard Time, Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*, Addison-Wesley, New Jersey, 1999.

10. Frakes, William B., Fox, Christopher J., Nejmeh, Brian A., *Software Engineering in the Unix/C Environment*, Prentince Hall, Englewood Cliffs, New Jersey, 1991.

11. Hofmeister, Christine, Nord, Robert, Soni, Dilip, *Applied Software Architecture*, Addison-Wesley, New Jersey, 2000.

12. Holden, Michael John, *Ada Implementation of Concurrents Execution of Multiple Tasks in the Strategic and Tactical Levels of the Rational Behavior Model for the NPS PHOENIX Autonomous Underwater Vehicle (AUV)*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1995.

13. Laplante, Phillip A., *Real-Time Systems Design and Analysis, an Engineer's Handbook*, IEEE Press, New York, 1993.

14. Larman, Craig, *Applying UML and Patterns*, Printence Hall, New Jersey, 1998.

15. Leffingwell, Dean, Widrig, Don, *Managing Software Requirements, a Unified Approach*, Addison-Wesley, New Jersey, 2000.

16. Lewis, Daniel W., *Fundamentals of Embedded Software, where C and Assembly meet*, Prentince Hall, New Jersey, 2002.

17. Marco, David, *Autonomous Control of UnderwaterVehicles and Local Area Maneuvering*, Ph.D. Dissertation, Naval Postgraduate School, Monterey California, September 1996.

18. Nilsen, Kevin, *Issues in the Design and Implementation of Real-Time Java*, Iowa State University, Ames, Iowa, 1995.

19. Nilsen, Kevin, *Embedded Real-Time Development in the Java Language*, Iowa State University, Ames, Iowa, 1998.

20. Nilsen, Kevin, *Real-Time Java*, Iowa State University: Ames, Iowa, 1996

21. Nilsen, Kevin, *Functional Requirements for Core Real-Time Extensions for the Java Platform*, Iowa State University, Ames, Iowa, 1999

22. Shaw, Alan, *Real-Time Systems and Software*, John Wiley & Sons, Inc., New York, 2001.

23. Sun Microsystems Inc., *The Java Language Overview*. Sun Microsystems, Inc., Mountain View, California, 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Dan Boger
   Naval Postgraduate School
   Monterey, California

4. RADM Winsor Whiton
   Naval Security Group Command
   Fort Mead, Maryland

5. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

6. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

7. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia

8. Marine Corps Tactical Systems Support Activity
   (Attn: Operations Officer)
   Camp Pendleton, California

9. Dr. Donald P. Brutzman Code UW/Br
   Undersea Warfare Academic Group
   Naval Postgraduate School
   Monterey, California

10. Dr. Man-Tak Shing, Code CS/Sh
    Computer Science Department
    Naval Postgraduate School
    Monterey, California

11. Dr. Anthoney J. Healey, Code ME/Hy
    Mechanical Engineering Department
    Naval Postgraduate School
    Monterey, California

12. Capt Miguel A. Ayala
Computer Science Department
Naval Postgraduate School
Monterey, California

13. Research Associate Doug Horner
MOVES Institute
Naval Postgraduate School
Monterey, California

14. Research Associate Bruce Allen
MOVES Institute
Naval Postgraduate School
Monterey, California

15. Grace Francisco
TogetherSoft
Mountain View, CA