NAVAL POSTGRADUATE SCHOOL
Monterey, California

PROJECT REPORT

THE LONWORKS TECHNOLOGY FOR A CONTROL NETWORK ONBOARD THE *PHŒNIX* AUV

By

Laurent Beguery
Alexandre David

August, 1998

Project Advisors: Dr Tony Healey
Dr Don Brutzman

Abstract

The Naval Postgraduate School is on the leading edge of Autonomous Underwater Vehicle (AUV) research and has developed its own AUV called the *Phoenix*.

This vehicle is a surveying robot used for mine countermeasures in shallow waters. It is equipped with a variety of different sensors and actuators, thus engendering strong data flow requirements. As of now, hardware devices onboard the *Phoenix* are manually configured and connected to a single computer. This approach is becoming more and more outdated and the throughput of the system is no longer satisfactory.

This project investigates the possibility of implementing LonWorks hardware and LonTalk communication protocol as a decentralized, interoperable, extensible and more reliable networked control system.

The *Phoenix* devices (thrusters, propellers, fins, rudders, sonars, gyroscopes, etc) can be modeled and installed on a prototype LonTalk network. This project demonstrates that it is possible to control motors and servos by the means of this technology, thus allowing a future use of this technology to control the AUV. A LonWorks control network would make the *Phoenix* much more failure-resistant and better able to support real-time behavior.

Table of Contents

Table of Figures

List of Acronyms

AUV	Autonomous Underwater Vehicle
A/D	Analog to Digital
CAUVR	Center for Autonomous Underwater Vehicle Research
DC	Direct Current
DGPS	Differential Global Positioning System
D/A	Digital to Analog
Flex I/O	Flexible Input Output Card (Manufactured by IEC)
GPS	Global Positioning System
IEC	Intelligent Technologies Corporation
ISO/OSI	International Organization for Standardization/Open Systems Interconnects
I/O	Input/Output
MAC	Media Access Control
NPS	Naval Postgraduate School
NV	Network Variable
PWM	Pulse Width Modulation
RBM	Rational Behavior Model
SLTA	Serial to LonTalk Adapter
SNVT	Standard Network Variable Type

Acknowledgements

First of all, we would like to thank Professor Tony Healey for the unconditional support he gave us throughout this project. If not for his perseverance and the confidence he demonstrated towards us, this project would not have been as successful.

Secondly, we wish to extend our gratitude to Dr. Don Brutzman who welcomed us to the Naval Postgraduate School and helped us a lot during this stay.

We are also very grateful to LCDR Jeff Riedel and Dr. David Marco who answered every possible question imaginable and whom support, knowledge and guidance have been a constant inspiration for us.

Also, we would like to acknowledge CDR Mike Holden for his help to find a place to stay, his enthusiasm and his sense of humor.

We owe a lot to Warren Rogers and Xiaoping Jun (EC engineering department), Eric Heatzig (Florida Atlantic University), Stewart Goldenberg (IEC Technologies) and Anup Sidhu (ATMEL Corporation) for their technical support throughout this project.

We would like to thank our landlord Christy Farrar and her son Austin with who we had lots of fun and who made our stay here a great pleasure.

We are also grateful to everybody for having supported the French team for its tremendous victory in the World Cup soccer final, and especially LCDR Todd Hawkinson who “proudly” displayed a French flag on his desk after a bad guess for the final’s winner…

Lastly, but mostly, we would like to thank our project supervisor in France, Didier Léandri, who has given us the opportunity to live this wonderful experience abroad.

I. Introduction

A. Background

As mechanical engineering students from the “Ecole Nationale d’Ingénieurs” in Tarbes, France, we have been given the opportunity to come to Monterey, California to work with the Naval Postgraduate School for our final project work.

The school is very involved in Autonomous Underwater Vehicle (AUV) research, and has therefore created a Center for Autonomous Underwater Vehicle Research (CAUVR) that is exploring many concepts in the design and control of AUVs The concepts developed have been applied to the NPS *Phoenix* AUV.

An Autonomous Underwater Vehicle is a self-contained unmanned vehicle used for missions such as surveying, pollution detection or mine countermeasure in shallow waters, which is the main area of research this project is investigating.

B. Motivation

The motivation for this project stems from the steady progression in the development of the NPS *Phoenix* AUV. At the beginning of the program, there were only a few devices onboard the vehicle and there were consequently no real requirements for an advanced control system for these devices. This resulted in the installation of a central architecture network, where the hardware devices were manually connected and configured using parallel ports, serial ports, A/D and D/A controllers, all connected to a single computer.

The current vehicle still uses this architecture, but the number of hardware devices in place on the AUV has increased at such a pace that data transmission management problems are arising. Moreover, it is becoming more and more tedious to upgrade the software or to implement a new device since that implies several modifications in the system configuration. This approach is time-consuming, poorly failure-resistant and not satisfactory anymore.

With a view to solving this problem, the implementation of a more efficient network architecture was considered. Echelon LonWorks hardware and LonTalk communication protocol constitute a flexible A/D and D/A hardware networking technology that appears to provide reliable communication, decentralized (peer-to peer) topology with no single point of failure, easy extensibility and interoperability for a wide variety of hardware devices (Echelon 97).

The reliability and throughput of *Phoenix* onboard devices can therefore be significantly improved using a LonWorks-based control system. This project demonstrates how such a configuration can be accomplished.

C. Goals for Project

The primary goal of this project is to test a prototype LonWorks network that connects the main types of *Phoenix* devices, i.e. the thrusters, the planes (rudders and fins) and the measurement devices (sonars, gyros, GPS, etc).

For this purpose, this network features:

- Thrusters modeling: Two DC brush motors coupled to D/A converters managed by an I/O board,
- Fins and rudders modeling: one servo coupled to an I/O board,
- Measurement devices modeling: a serial gateway.

To achieve the main goal, there are therefore three sub-problems to address:

- the first step is to be able to command the two brush motors through the network. In order to do so, it is necessary to understand the working of the Flex I/O printed circuit boards purchased from Intelligent Technologies Corporation (IEC).
- these I/O boards are also used to create a PWM output in order to control some servos that simulate the fins and rudders of the AUV.
- finally, the last point focuses on achieving data transmission on the network through a serial port. To achieve this goal, a serial gateway has to be used.

D. Summary of Chapters

The topic of this project is the adaptation of the AUV to increasing data processing requirements (due to the implementation of new devices onboard the vehicle) by using a distributed LonWorks control network.

The present chapter is devoted to the background, motivation and goals for this project. Chapter II presents the NPS AUV project and the virtual world used to accelerate its development, and provides as well an introduction to the network technology considered in order to improve the AUV data throughput. Chapter III states in detail the research problems addressed by this project. Operation of the LonBuilder software is explained in Chapter IV. Chapter V gives some important very tips to the user. The working of the motors and the servos is the topic for chapter VI and VII. Chapter VIII provides conclusions regarding this project and presents recommendations for future work. The appendices contain source codes, some information on control networks, a description of a LonWorks device, the complete Navigator command chart, an explanation of the most common LED behaviors on Flex I/O boards, and finally a description of the contents of our directory on the computer connected to the LonBuilder.

II. Related Work

A. Introduction

Research on Autonomous Underwater Vehicles has been an ongoing project at the Naval Postgraduate School (NPS) since 1987 [Healey 90,92] [Brutzman 96] through the *Phoenix* project.

This section provides a general overview of the *Phoenix* vehicle (hardware and software) as it is now. It will also include a presentation of the virtual world used to predict the AUV reaction in particular environments and conditions, as well as an introduction to the network technology considered to implement a distributed control network onboard the AUV.

The *Phoenix* AUV is an untethered robot submarine designed for research in adaptive control, mission planning, mission execution, and post-mission data analysis [Healey 90]. It has been developed to prove that it is possible to achieve ocean survey and mine countermeasures by the means of a low-cost unmanned submarine.

B. Phoenix AUV

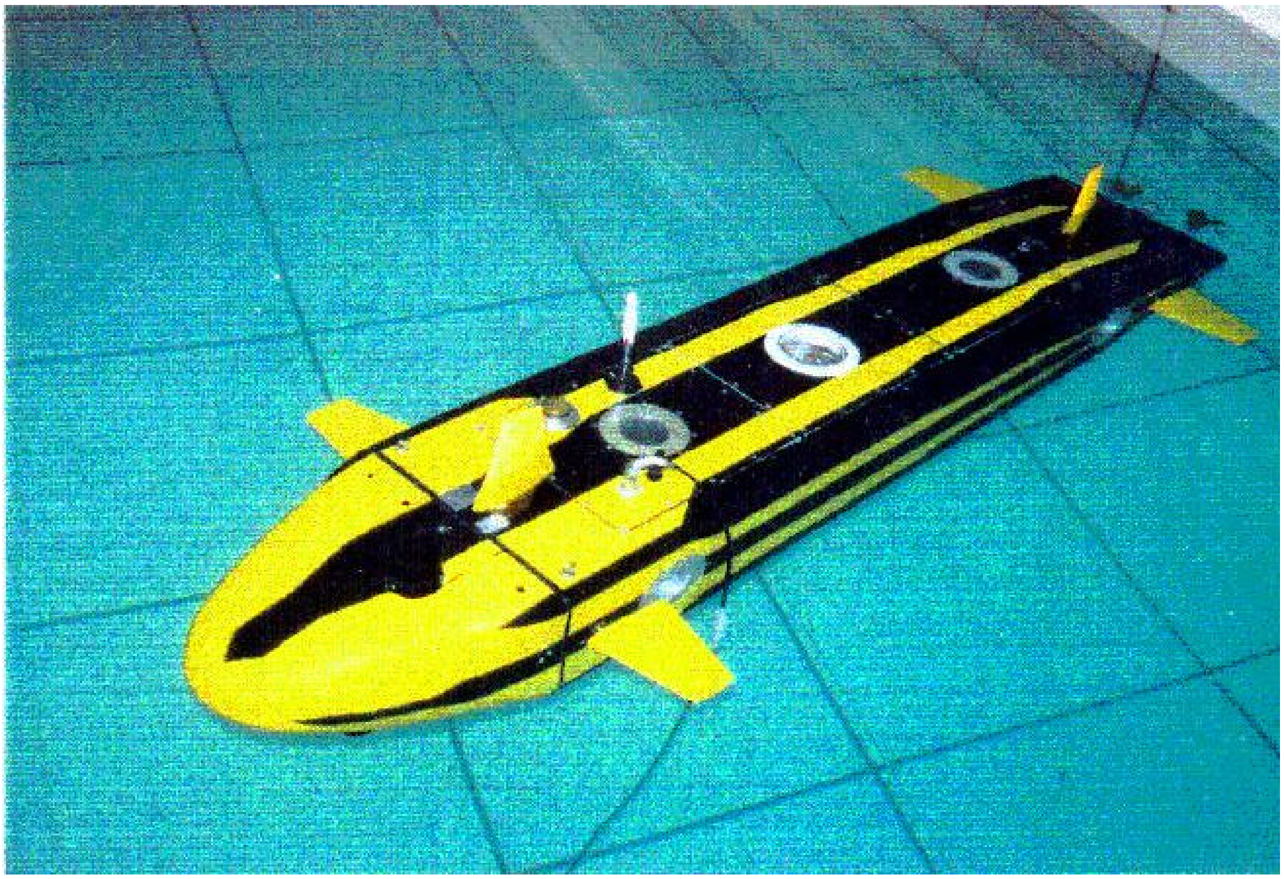


Figure 1: *Phoenix* AUV undergoing testing at the Center for AUV Research (CAUVR) laboratory te
early 1995.

1. Physical Description

The Naval Postgraduate School *Phoenix* AUV is approximately 2.4 meter long, 0.46 meter wide and 0.31 meter deep. It has the shape of a miniature submarine with two aft propellers, two vertical thrusters, two horizontal thrusters, one or two fore rudders (the upper one is sometimes removed), two aft rudders, two fore fins and two aft fins to control its movement through the water.

The AUV has a 2 psi pressurized aluminum hull with a free-flooding nose cone that houses some of the AUV's measurement devices. The vehicle is designed to be neutrally buoyant at three hundred and eighty seven pounds with a design depth at twenty feet.

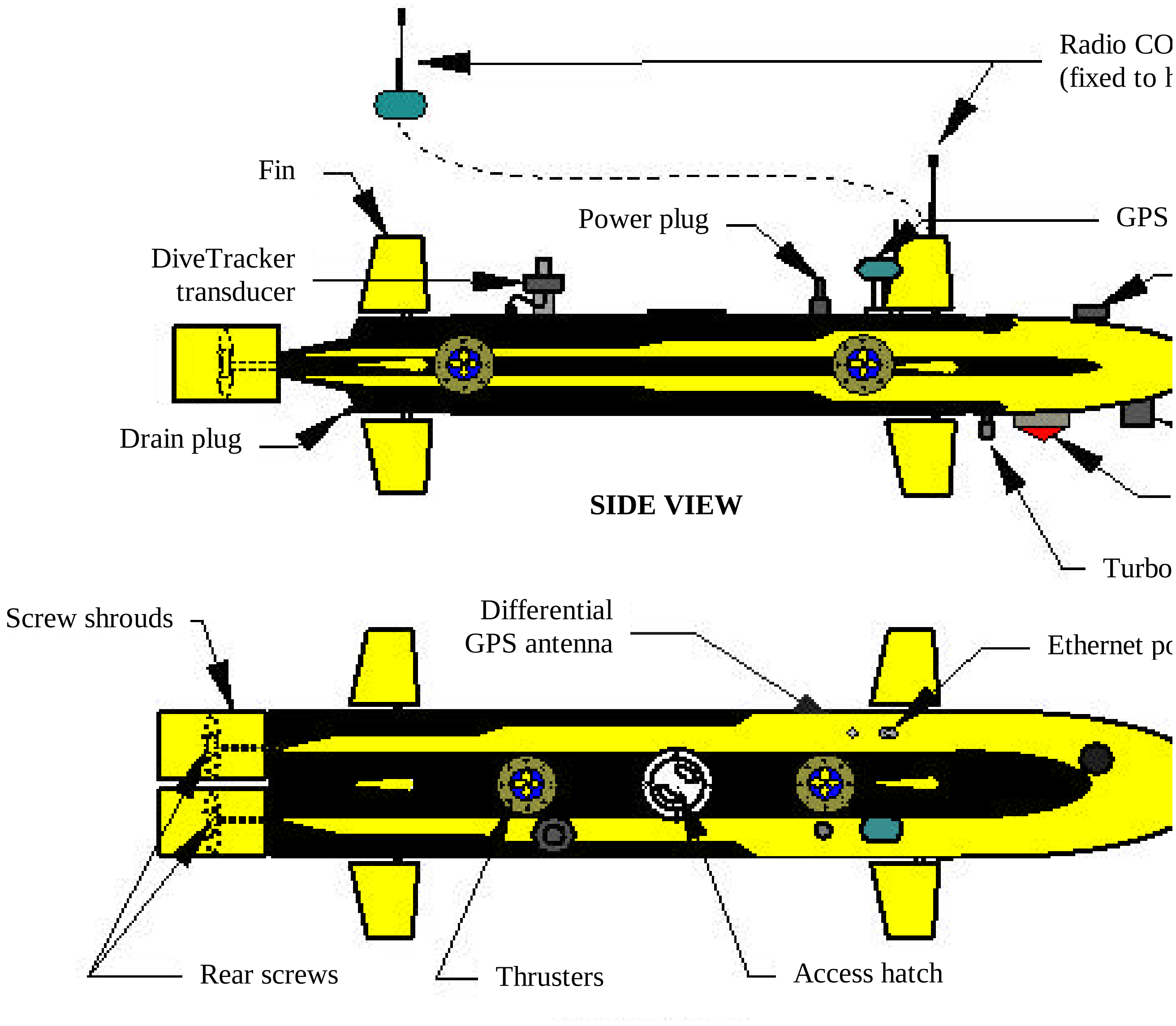
Lead-acid batteries providing endurance up to two hours electrically power the submarine.

For the survey and mine countermeasure purposes mentioned above, several devices have been installed in the AUV; some are intended for navigation and others are used for measurements.

The following list details these pieces of hardware and their purposes:

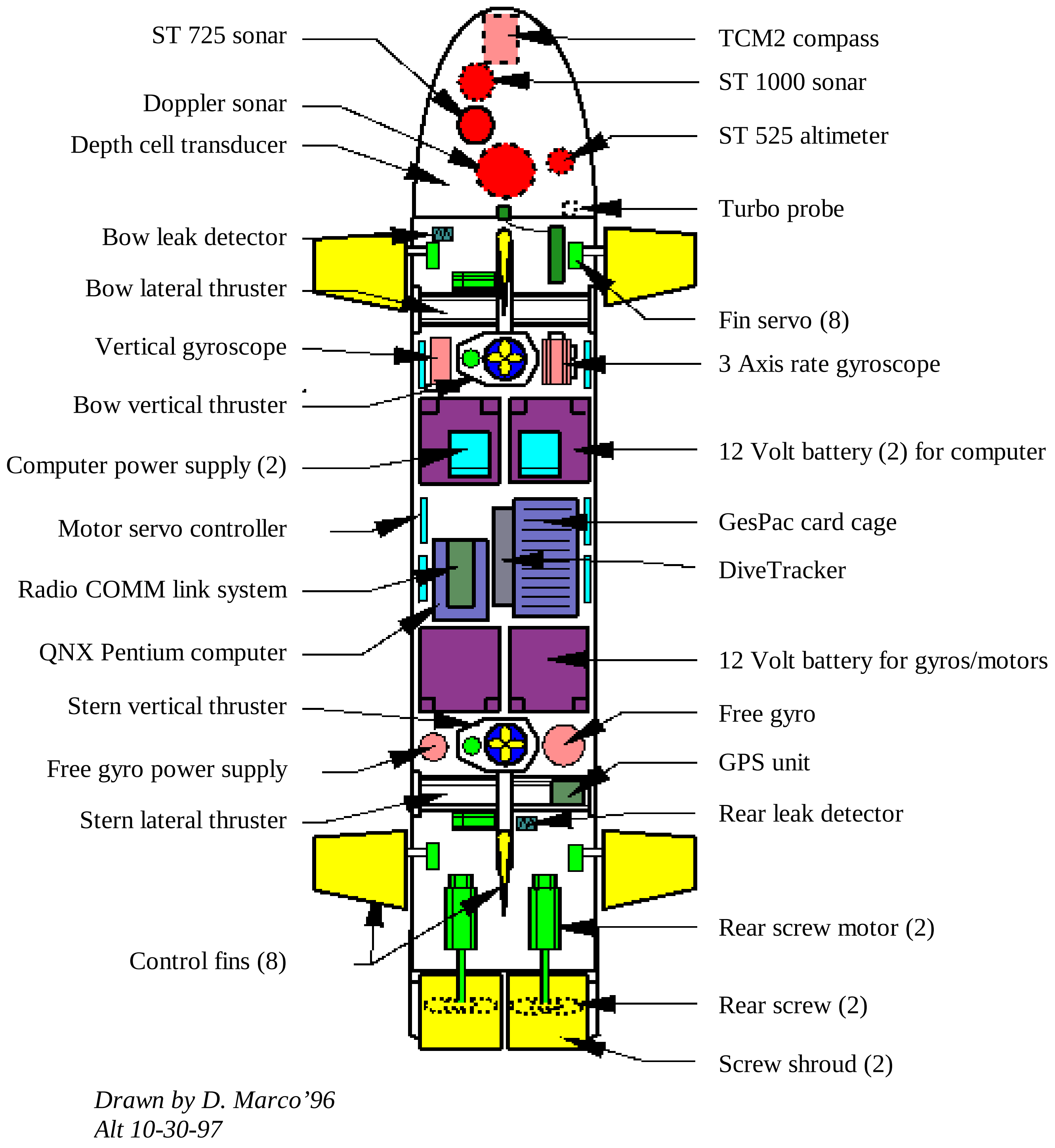
- A GesPac computer for controlling the AUV's stability, *execution* level of software,
- A Sun Sparc 5 computer for data storage and running *strategic* and *tactical* levels of software,
- Three sonars:
    - ✓ Downward looking (altimeter),
    - ✓ Overall environmental sensing (ST 725 model),
    - ✓ Obstacle classification (ST 1000 model),
- GPS and DGPS for tracking the vehicle latitude and longitude,
- DiveTracker for precision tracking,
- Gyros for sensing the vehicle's orientation by measuring angles and rates for roll, pitch and yaw respectively,
- A depth cell,
- A/D and D/A converters for computer hardware interfaces,
- Lead-acid batteries for power supply.

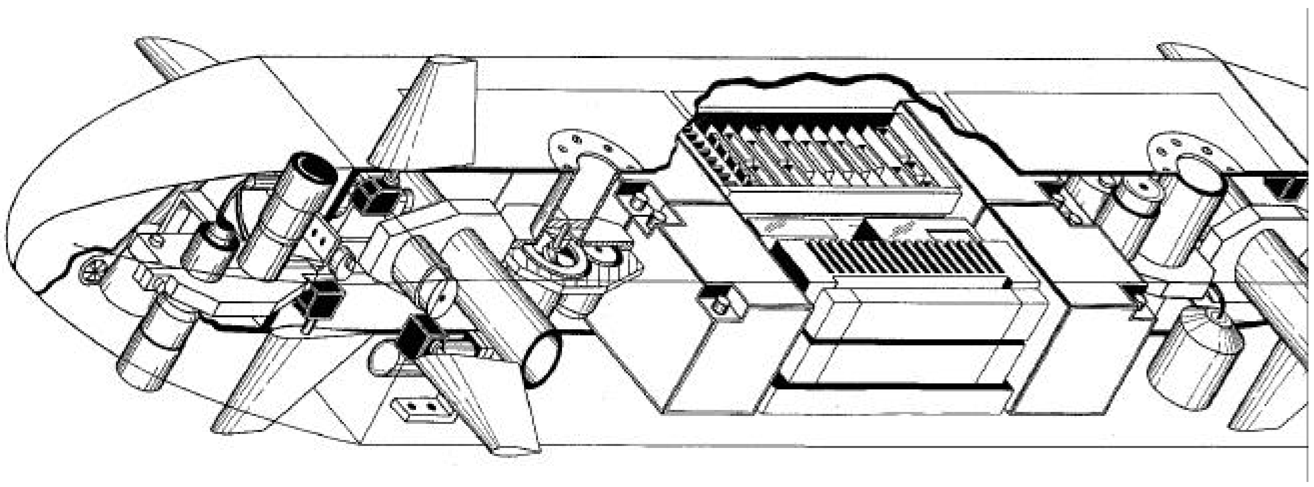
NAVAL POSTGRADUATE SCHOOL PHOENIX AUV



NAVAL POSTGRADUATE SCHOOL
CENTER FOR AUV RESEARCH

PHOENIX AUV FOR MINE RECONNAISSANCE/ NEUTRALIZATION IN VERY SHALLOW WATERS





Perspective view of the *Phoenix* AUV [Marco 96]

Note: The use of the forward rudders is optional during under-water navigation. Therefore, the upper one is
is as well superfluous during surface navigation, while the lower one remains since it is essen
conditions.

2. Software

The *Phoenix* AUV uses a tri-level software architecture called the Rational Behavior Model (RBM). The RBM architecture has been created as the model of a manned submarine operational structure.

RBM divides responsibilities into areas of open-ended strategic planning, soft real-time tactical analysis, and hard real-time execution level control.

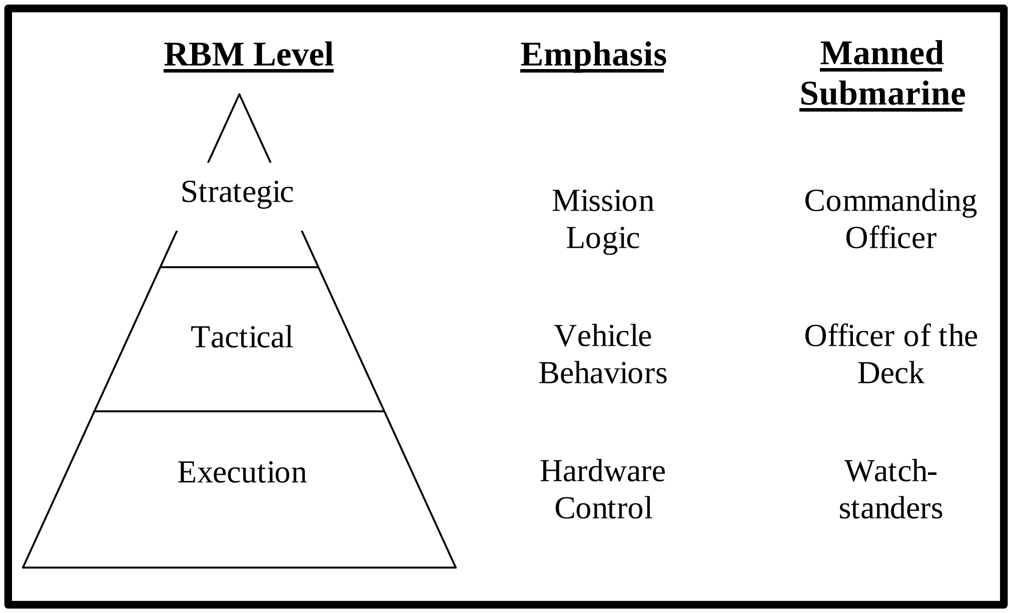


Figure 2: The Rational Behavior Model tri-level architecture hierarchy with level emphasis and submarine equivalent listed [Holden 95]. It shows the relationship between strategic, tactical and execution level in the RBM.

The execution level deals with the interface between software and hardware and has been designed to work in hard real time. The execution level is responsible for the underlying stability of the vehicle, the control of individuals devices, and providing data to the tactical level [Byrnes 93][Byrnes 96]

The level of command execution is in the tactical level. It divides the goal into individual tasks to be completed; hence it operates in terms of discrete events rather than operating on hard real-time deadlines. It provides the interface between the execution and the strategic levels, thus giving strategic level indication of vehicle state and completed tasks, and execution level commands.

The highest level in the RBM is the strategic level. It monitors the completion of mission goals and hosts the mission specification. The strategic level conducts a dialogue with the tactical level by following a set of rules.

C. Virtual World

The main asset of an AUV, that is the ability of performing tasks autonomously, also becomes its worst drawback when it comes to in-water testing of the robot, because it operates in a remote and hazardous environment and it is often impossible to observe or communicate with the vehicle. The development process becomes consequently very difficult for the designers.

To overcome this problem, a virtual world was created which models salient characteristics of the ocean environment from the robot’s perspective [Brutzman 94]. This allows developers to visualize robot behavior under diverse conditions.

Prior to the virtual world creation, the AUV had to be tested in the CAUVR laboratory test tank in order to verify any upgrade to the software. A simulation capable of modeling the AUV actions and reactions was therefore required and would make the project advance at a much faster pace.The virtual environment provides an area of underwater terrain where the AUV can maneuver.[Brutzman, 94]

The implementation of the *Phoenix* virtual world is divided into three major modules:

- vehicle hydrodynamics and sonar models,
- AUV software,
- interactive 3D graphics user window into the environment.

D. LonWorks Technology

1. Advantages of a distributed control network

The AUV has a lot of devices, all connected to a single computer frame. Therefore, the wiring is very complicated and tedious, and the number of inputs and outputs available on the computer board is limited.

It is possible to overcome these problems through the utilization of a control network. Instead of being centralized to a single device (the computer), the data flow is distributed over several intelligent devices (also called *nodes*) communicating using a common protocol.

As a consequence, a distributed control network is cheaper to install (less wiring and less labor), easier to reconfigure (logically rather than physically), improves the response time with peer-to-peer communications and makes integration of different systems from different manufacturers easier via interoperability [Echelon 96]. However the configuration equipment is quite expensive ($10-15K) and the learning curve is steep

2. LonWorks technology

When it comes to network technologies, the communication protocol is very important. LonWorks uses the International Organization for Standardization Open Systems Interconnects (ISO/OSI) standard. It is the most common data transmission model and it is composed of seven layers of communication. These layers are listed in figure 3.

The LonWorks technology allows the user to build distributed networks consisting of intelligent <u>nodes</u> communicating using a common <u>protocol</u> (LonTalk Protocol) over one or more <u>media</u> (twisted pair, radio, fiber optic, infrared…).

A network is divided into different entities: Domains, Subnets, Nodes, Groups and Channels.

- A Channel is a physical transport medium for packets of data.
- A Domain is a logical collection of nodes on one or more channels.
- A Subnet is a logical collection of up to 127 nodes within a domain.
- A Group is a logical collection of nodes within a domain. Unlike the subnet, however, nodes are grouped together without regard for their physical location in the domain.

Each node features a chip called a ***Neuron Chip***, which implements the seven layers of communication of the ISO/OSI model by the means of three processors, each managing different layers (see figure 3). Each chip has a unique identification number to avoid confusion between two chips by the software. The nodes can receive and/or send messages from any other node situated on the same network.

OSI LAYER	WHERE?	HOW?
7: Application	Application processor	User software
6: Presentation 5: Session 4: Transport 3: Network	Network Processor	Neuron Chip Firmware
2: Link	MAC Processor	
1: Physical	Media Transceiver	XCVR Hardware

Note: The three processors are usually integrated into the Neuron Chip, nevertheless the Application processor can be a non-Neuron Chip host.

Figure 3: The OSI layer model and its implementation with the LonWorks technology [Echelon seminar Day 2 pages 2 & 3, 95]

The messages are broadcast using a standard way of passing data defined in the LonTalk Protocol. The transmission is data oriented rather than command oriented. LonTalk also promotes a standard object-based view (model) of devices, which reduces the impact on the network when the device's inner working are changed and makes systems adds, moves or changes easier [Echelon seminar 95]. The devices are modeled into four types of objects (called LonMarks objects): actuators (motors…), sensors (pressure…), controllers (lighting…) and operators (displays, computers…). These objects allow the implementation of standard methods for data passing, data encoding, device documentation and device configuration.

The messages propagated throughout the network consist of an **address** and a **data string**.

Five different types of **address** formats exist, going from a short address size (message intended for all the nodes in the domain) to a long format (message intended for a single node). A network management tool called the LonManager decides itself what type of address format is used and this choice is user-transparent. Considering the facts that the *Phoenix* AUV only requires a single channel and that the number of devices on board is quite limited, there is therefore only a need for a single subnet within a single domain. Hence, the messages are sent to all the nodes in the subnet and those nodes just 'pick' the messages they are interested in from all the messages broadcast throughout the network.

Note: The possibilities of a LonWorks network are better exploited with the use of a network management tool. A network management tool is a node which has been created to perform several management tasks on the network, such as:

- Find unconfigured nodes and download network addresses,
- Stop, start and reset node application statistics,
- Download new application programs,
- Extract the topology of a running network.

[LonWorks Engineering Bulletins: LonTalk Protocol 93]

The **data string** constitutes two parts: a variable, which can belong to standard types (called SNVT) and a value or state itself.

Standard Network Variable Types (SNVTs) facilitate interoperability by providing a well-defined interface for communication between nodes made by different manufacturers. A node may be installed in a network and logically connected to other nodes via network variables as long as data types match. [Echelon SNVT Master List and Programmer's guide 97]

The network management tool is used to link those SNVTs (see Chapter IV for details).

E. LonWorks Devices

1. LonBuilder

The LonBuilder is a development workbench manufactured by Echelon. It is an integrated hardware and software development environment running on a PC host computer [Echelon LonBuilder User Guide 95]. The workbench includes the components required for the development of LonWorks applications.

The LonBuilder development station includes:

- An enclosure with power supply and backplane hardware that supports up to six operational processor boards.
- A control processor to support communication between the development station and the processor board. The control processor communicates with the PC through the LonBuilder interface adapter, and it includes a network management node and a protocol analysis node (also called ***protoanalyzer***).

2. Flex I/O

IEC's flexible I/O control module is a general purpose LonWorks based node. This module is a complete application node for implementing, monitoring and controlling functions in a LonWorks control network. It incorporates several types of inputs and outputs (analog, digital, relay and open-drain) as well as power supply and a LonWorks compatible transceiver.

The flexible I/O control module provides configurability for its analog inputs and outputs through jumper selection.

A 3150 model Neuron chip is embedded on this board, which also features a socket hosting an external memory (EPROM, EEPROM or flash). Considering the AUV control networking requirements, an ATMEL AT29C256 flash memory has been selected.

The user's Neuron C program is converted into a suitable format corresponding to the type of memory used, and then burned into the programmable memory. Afterwards, an interface file is used in order to configure the node. This interface file describes all the inputs/outputs of the card.

3. Serial Gateways

A Serial LonTalk Adapter (SLTA) is a Network Interface Card (NIC)that enables any host processor with an EIA-232 serial interface to connect to a LonWorks network. SLTAs extend the reach of LonWorks technology to a variety of hosts, including PCs, microcontrollers, etc.

For the AUV network, the SLTAs are used as interfaces for such devices as the sonars and the flow turbo probe.

F. Summary

The *Phoenix* is a high technology Autonomous Underwater Vehicle (AUV) that can operate in shallow and very shallow water. Using a tri-level software architecture RBM model, it mimics a manned submarine operational structure. In order to accelerate its development, a virtual world has been created to perform virtual tests in various and unusual conditions. With the implementation of the LonWorks technology through a distributed control network, the AUV will have better response time and control stability.

G. Network description

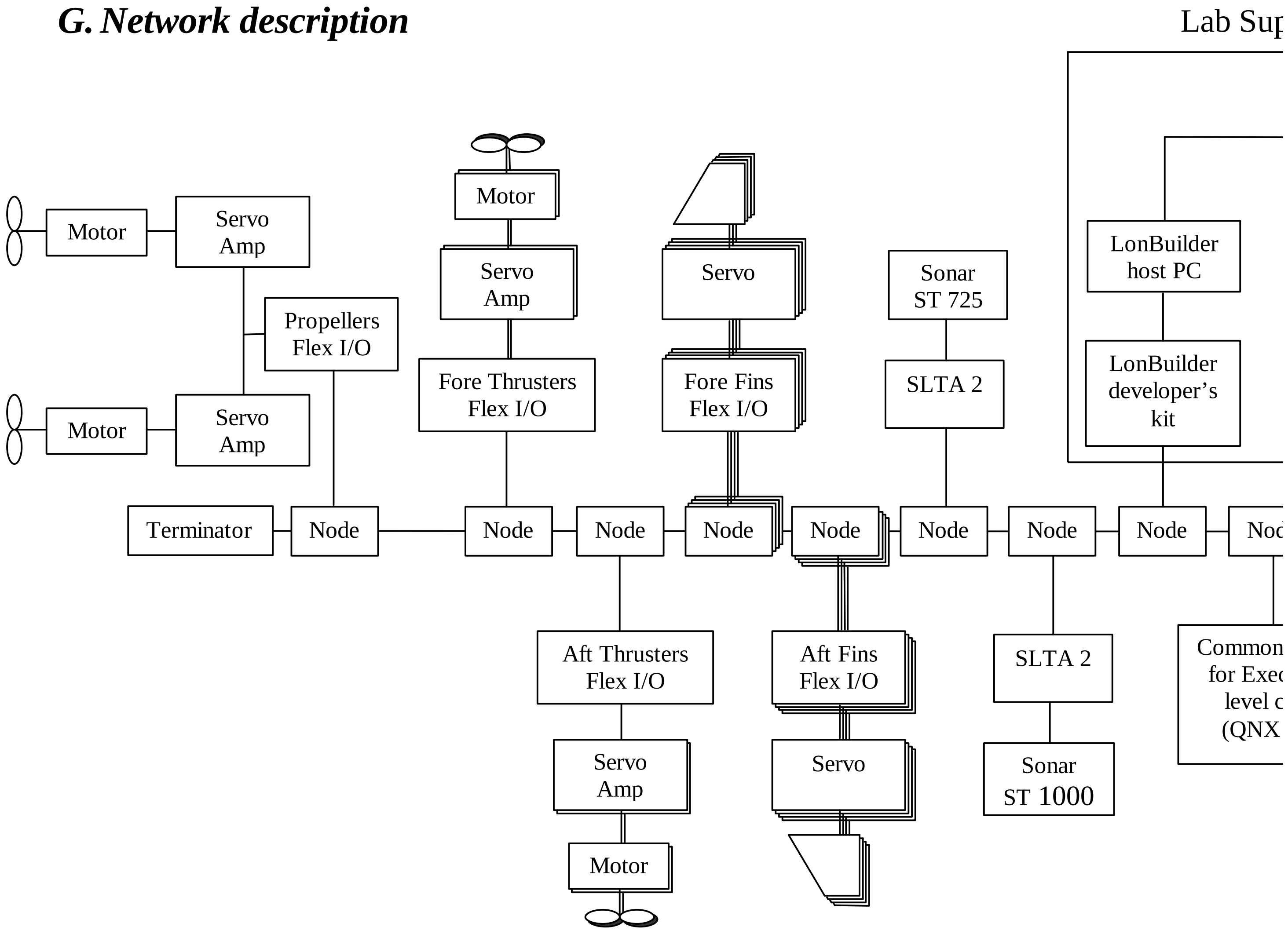


Figure 4: The LonWorks Local Control Area Network (LAN) for the *Phoenix* AUV

III. Problem Statement

A. Introduction

The *Phoenix* AUV is a prototype undergoing continuous development, and new hardware devices are regularly implemented to improve the capabilities of the vehicle. Therefore, the data flow in the current network is increasing, a communication bottleneck begins to occur and the wiring becomes more and more tedious. Individual serial port connections for numerous devices are no longer a practical solution.

B. Origin of the Problems

The problems listed above derive from several factors: the maximum data processing rate of the GesPac computer, the complexity of the central architecture network and the necessity to reconfigure the system in case of a component addition or removal.

As previously stated, the execution level of the RBM model is managed on the AUV by the means of a GesPac computer implemented in a central architecture network. The maximum data processing rate of this computer is 10 Hz, which does not provide adequate bandwidth to control all the devices onboard the *Phoenix*. It is therefore very difficult to achieve proper vehicle control stability. Furthermore, when it is necessary to modify some hardware in the network, a software reconfiguration is compulsory and some compatibility problems can occur.

It is possible to tackle all these problems by making use of a decentralized (peer to peer) network. A peer-to-peer control network based on the LonWorks technology and using 10 MHz Neuron processor-based nodes has a potential aggregate bandwidth of 1.25 Mbits per second (i.e. 800 packets per second continuously and 1000 at peak point), hence the potential response time is considerably better. Moreover, the addition or removal of devices will become much easier since the network is decentralized and features a high throughput.

C. Real Time Response Considerations

The ultimate goal of the NPS AUV Research Group is to achieve real-time response on the *Phoenix* vehicle, which is not available now. This project attempts to obtain real-time data analysis by using a Pentium computer coupled to the aforementioned decentralized control network.

A response time better than 0.1s (10 Hz) would allow the development of advanced control methodologies for using the AUV in very shallow waters (between 10 and 30 feet) where persistent wave and current action from the seaway make operations difficult.

D. Starting Point

At the beginning of this project, part of the LonWorks network described page 15 was set up. The devices in place were: the LonBuilder, the host PC, a flex I/O board connected to two D/A converters and two motors, a flex I/O board intended for the servos, and a serial gateway SLTA/2.

This small network was designed on a single channel (twisted pair technology), and was a single subnet in a single domain. No need to change this configuration has occurred yet.

Several other devices from Echelon and IEC Technologies are available for any further extension of the network and for its implementation in the AUV:

- ✓ Two Motorola Gizmo 3,
- ✓ Application interfaces,
- ✓ Four LTM-10,
- ✓ Four FTM-10 transceiver,
- ✓ One FOC FAU serial,
- ✓ Two modular jack wiring blocks,
- ✓ One LonBuilder application interface adapter,
- ✓ Two TP/XF 1250 modules,
- ✓ Five TPT/XF/1250 modules,
- ✓ One connectivity starter kit,
- ✓ Two PCNSS PC interface,
- ✓ Two serial gateway interfaces,
- ✓ One NodeBuilder,
- ✓ Two memory racks.

A very complete bibliography has been provided as well by Echelon which includes many documents covering various subjects such as user's manuals, programming manuals, introduction to the LonWorks devices, engineering bulletins (describing particular applications and devices in small leaflets) and a LonTalk seminar that we followed on our own as a training session. The latter explains the LonWorks technology and presents a few programming examples using the emulators on the LonBuilder.

E. Summary

This chapter addressed the problems inherent in the current *Phoenix* AUV configuration using a central networked control system. A distributed network would allow tackling these problems, and the AUV research group considered the LonWorks technology as a solution. By using a distributed network, most of these problems will be solved. The AUV research group decided to try the LonTalk technology.

IV. Operating the LonBuilder

A. Introduction

In this section we will describe the basic operating principles of the LonBuilder. It should not be considered as a complete explanation of the software but as an overview of the LonWorks methodology.

B. How to Run LonBuilder

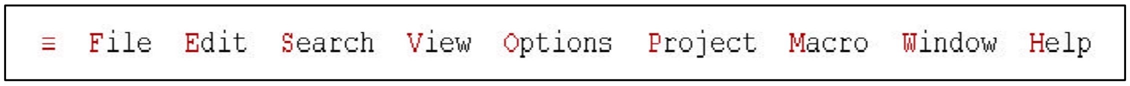
LonBuilder operates under a DOS environment. In order to run the software, it is necessary to create a dedicated directory containing another directory called ***project***.

The project directory will allow the LonBuilder to create all the files related to the project considered in the directory, such as the source code(s) or the converted code files (to be used as ROM images, interface files, EEPROM or flash memory images).

In the tutorial of the day 1 of the LonWorks training course [LonWorks tutorial, Day1, LBbegin_work, page 3], students are advised to create a new set of directories (c:\student\programX\…) every time they want to create a new program code. This method has an important drawback. In the *Phoenix* LonWorks network, several nodes are implemented, each of them requiring a different source code. If these codes are recorded in different ***project*** directories, i.e. different directories, then it will be necessary to log out from the LonBuilder software in order to switch from one program to another, and it will not be possible to run all the programs at the same time. It is therefore better at the development stage to create all the codes required for the *Phoenix* vehicle under the same directory for the user to have the possibility to debug and elaborate the network on the test bench.

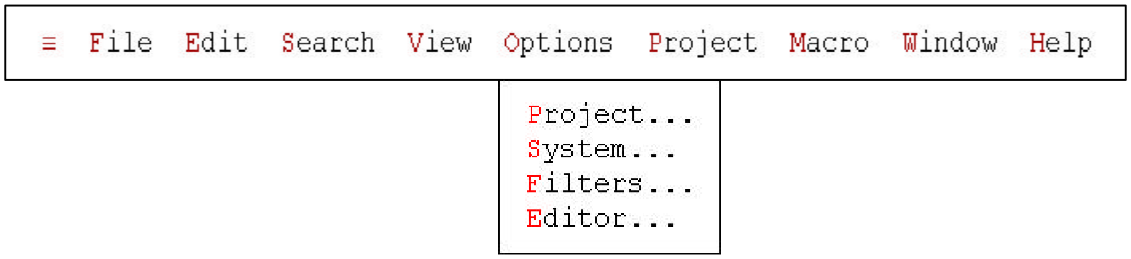
1. The Main Menu Bar

Once you have initialized your project directory, the main menu will be displayed:

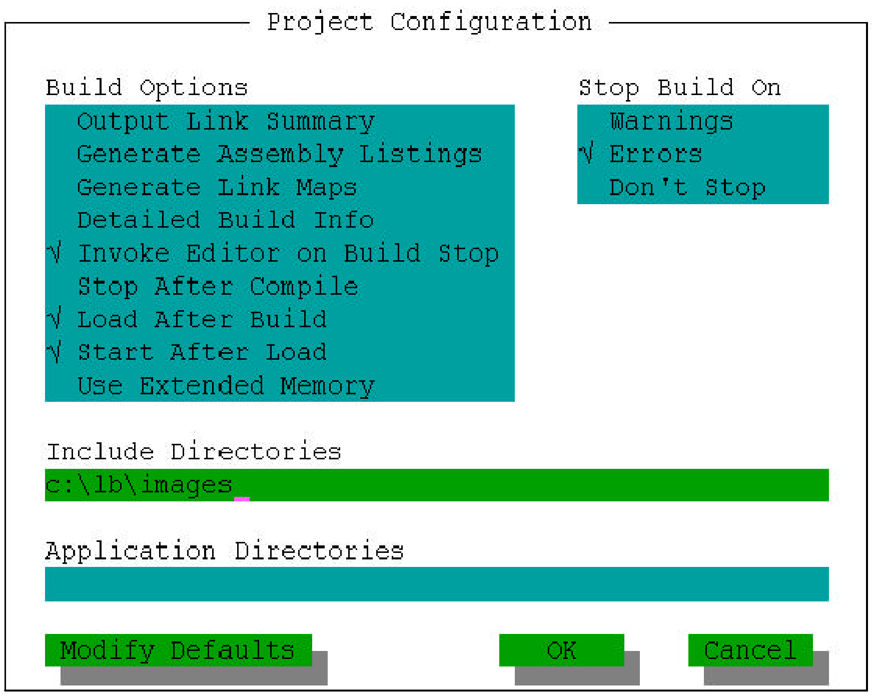


Some of those menus (***file, edit***, and ***search***) are very common and don't need any explanation. We have not used some of them. The others are more specific to the LonWorks technology and we found them very useful so we will explain how we used them here after.

For more information, please consult the LonBuilder user's guide.



Options: in this menu there is a submenu called ***Project*** which is used to customize the software to your needs. This is the configuration we are using.



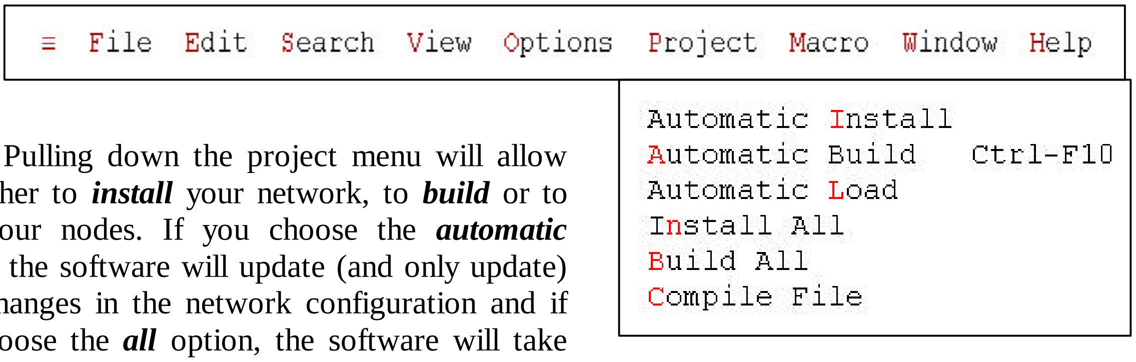
```
Project Configuration

Build Options                          Stop Build On
  Output Link Summary                    Warnings
  Generate Assembly Listings           √ Errors
  Generate Link Maps                     Don't Stop
  Detailed Build Info
√ Invoke Editor on Build Stop
  Stop After Compile
√ Load After Build
√ Start After Load
  Use Extended Memory

Include Directories
c:\lb\images

Application Directories

Modify Defaults                OK        Cancel
```

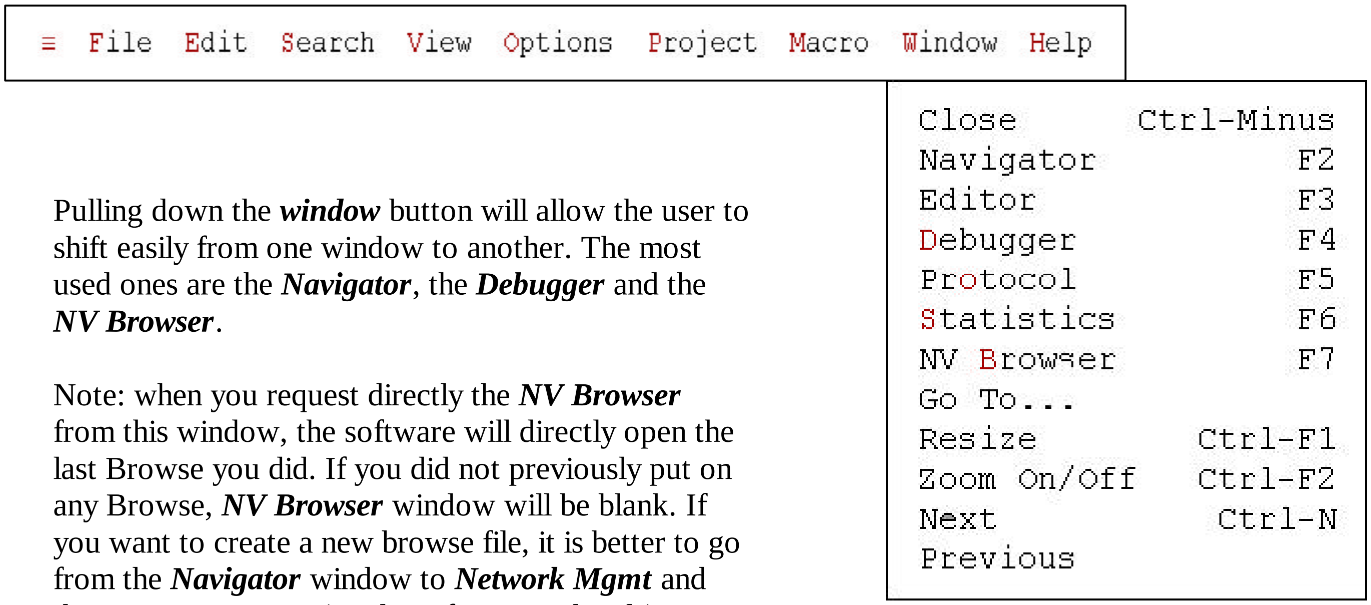


Pulling down the project menu will allow you either to ***install*** your network, to ***build*** or to ***load*** your nodes. If you choose the ***automatic*** option, the software will update (and only update) your changes in the network configuration and if you choose the ***all*** option, the software will take care of everything.

It is much quicker to do an ***automatic*** request but this can create some problems in the bindings and you can have links between SNVTs that you did not expect.

Every time the LonBuilder is switched off, an ***install*** has to be done.

Every time you want to ***load*** programs, you can either choose ***load*** (if you do not want to bind your variables) or ***build*** (if you are ready to run completely your programs).



Pulling down the ***window*** button will allow the user to shift easily from one window to another. The most used ones are the ***Navigator***, the ***Debugger*** and the ***NV Browser***.

Note: when you request directly the ***NV Browser*** from this window, the software will directly open the last Browse you did. If you did not previously put on any Browse, ***NV Browser*** window will be blank. If you want to create a new browse file, it is better to go from the ***Navigator*** window to ***Network Mgmt*** and then to ***NV Browser*** (see later for more details)

2. The Navigator Home Page

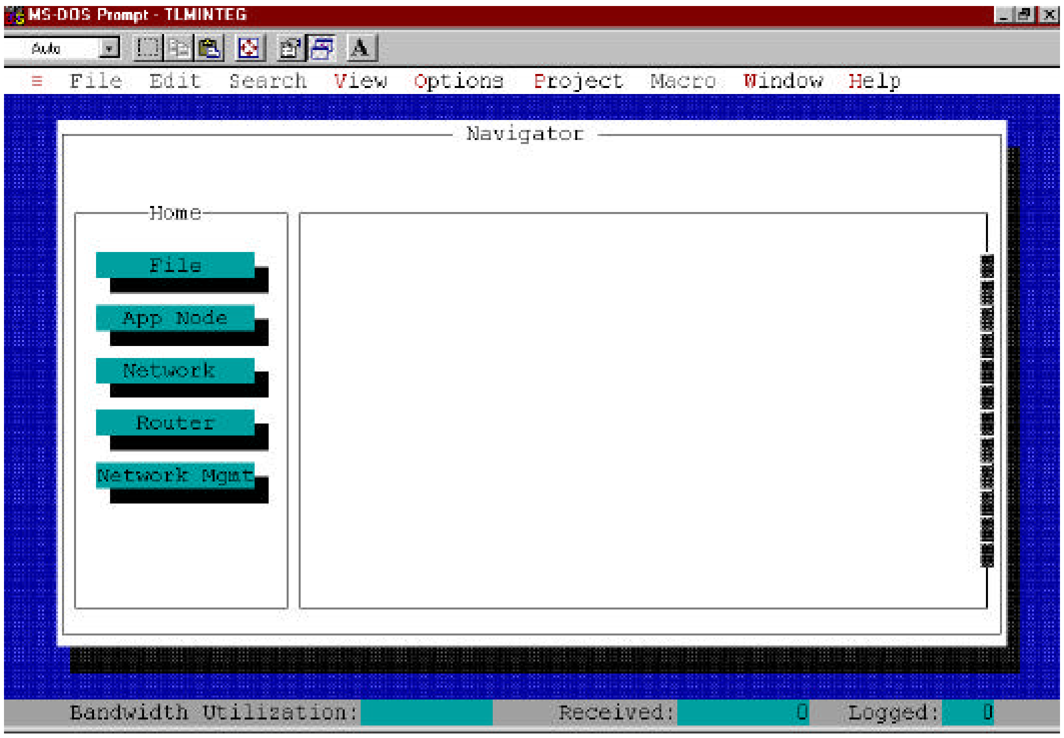


Figure 5: The Navigator Home Page

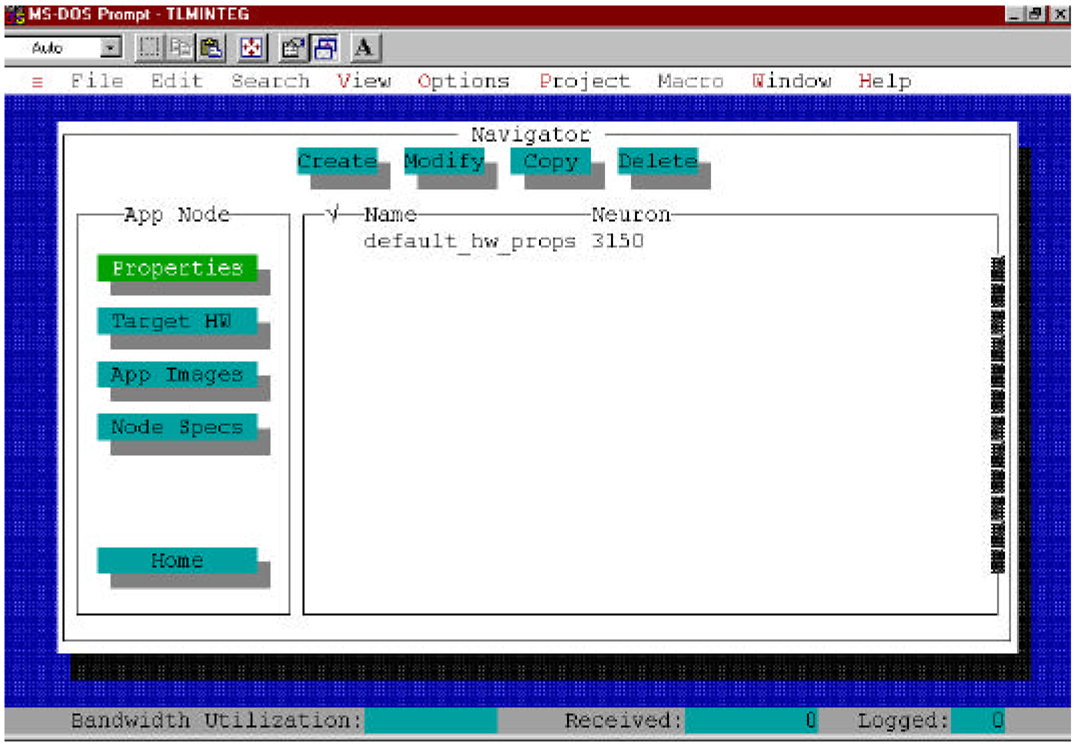
This window gives a good overview of the LonWorks methodology. First a ***file*** has to be created. It will be the program and is written in Neuron C. Then a ***node*** hardware has to be set up and an ***application*** has to be loaded in the node that will host the program (that will be explained in more details later). Once the program has been loaded in the Neuron Chip, the ***network*** connections have to be defined to identify the bindings between the different variables. There is no ***router*** for the AUV. With the ***network management*** tools it is possible to survey the network and to make changes to the variables.

By pressing the ***home*** button, you will return to the home page.
By pressing ***Neuron C*** another window will be displayed with two choices: ***File*** and ***Include.***

The user can ***create, modify, delete*** or ***compile*** Neuron C files (an object must be selected to use those functions).
The ***compiler*** button runs the compiling routine, which checks if the code is correct.
By pressing the ***Include file*** button, the user can include in his/her Neuron C files other programs. Then those programs can be created, modified or deleted.

Note: There can be a lot of programs in this section, it does not necessarily mean they will be all running when a built will be requested. It just means that the Neuron C code exists in the directory.

3. The Application Node Window



```
default_hw_props 3150
```

```
Bandwidth Utilization:            Received:          0   Logged:    0
```

Figure 6: The Application Window

From there you will perform most of the tasks.

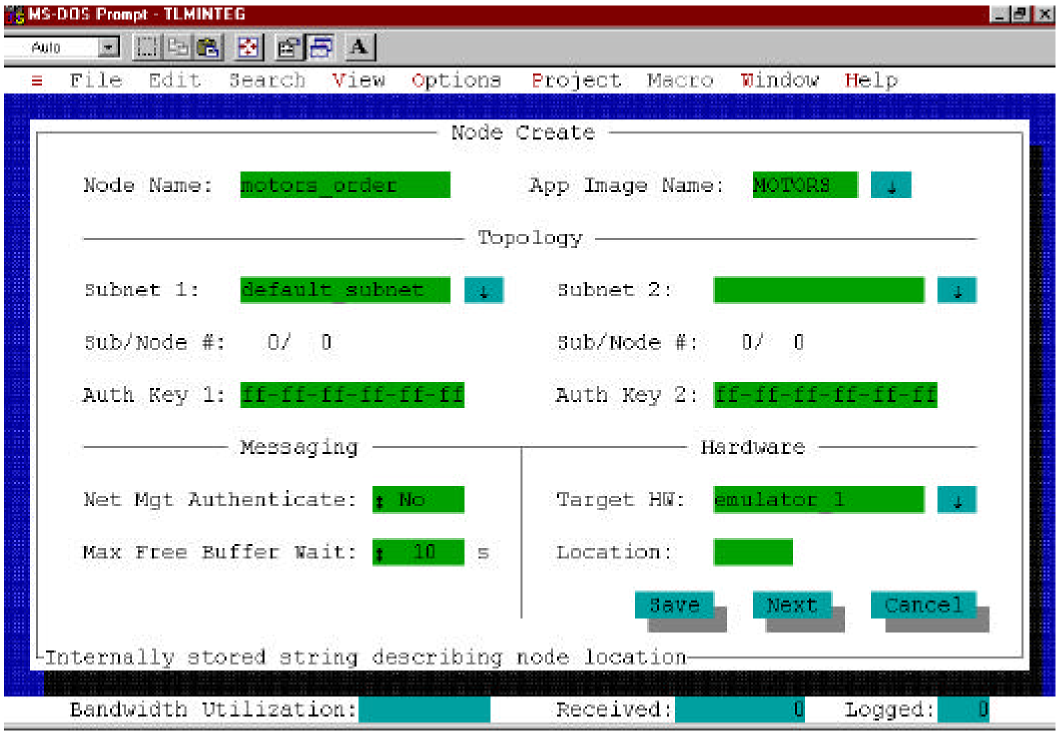
- The ***Hardware Properties*** define what kind of neuron chip you are using.
  There are two hardware properties. The first one (default_hw_props 3150) has been created for the custom nodes and include the neuron chip firmware SYS3150. The other one has been made only for emulators and, as it is a special node, it does not include the file SYS3150.

- The ***Target Hardware*** helps you to create your physical nodes. This is where the service pin of the cards has to be pushed in order for the software to record the Neuron ID number. The LonBuilder will therefore know that there is this node on the network.

- The ***Application Image*** creates a downloadable file from your program. It will take your Neuron C code (also called Source Code) and create a new file that the Neuron chip will understand.

- The ***Node Specifications*** put together what has been done on this screen.

4. The Final Creation of a Node



```
Node Create

Node Name: motors_order          App Image Name: MOTORS ↓

Topology
Subnet 1: default_subnet ↓       Subnet 2:  ↓
Sub/Node #: 0/ 0                 Sub/Node #: 0/ 0
Auth Key 1: ff-ff-ff-ff-ff-ff    Auth Key 2: ff-ff-ff-ff-ff-ff

Messaging                        Hardware
Net Mgt Authenticate: ↕ No       Target HW: emulator_1 ↓
Max Free Buffer Wait: ↕ 10 s     Location:

                                 Save   Next   Cancel
Internally stored string describing node location
Bandwidth Utilization:           Received: 0    Logged: 0
```

Figure 7: The final creation of a node

There are only three fields that needs to be updated:

- ***Node Name***
- ***Application Image Name***
- ***Target Hardware***

The Node Name is just any name you want to give to your node. Then you target an Application Image to that node and specify on which Hardware (or physical node) you want the program to be loaded

5. The Network Window

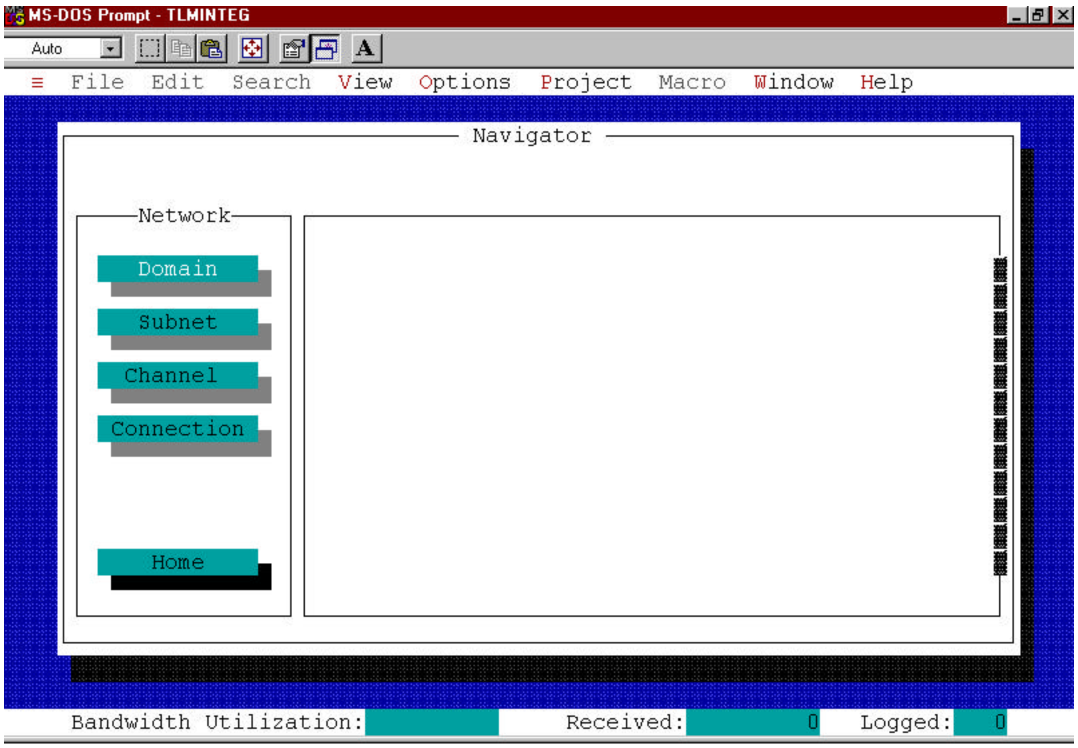


Figure 8: The Network Window

On this screen you can set up the network and the binding of the variables.

On the AUV network, there is only one ***Domain***, one ***Subnet*** and one ***Channel***. Therefore there is a default Domain, a default Subnet and a default Channel. One very important thing to do is the binding of the network variables (also called ***Connection***). Pressing each button in turn will allow us to bind the variables.

Note: The variable names are updated ONLY when the programs are loaded into the nodes. So prior requesting a Connection, make sure your programs are loaded in their respective nodes.

6. The Network Variable Connection Window

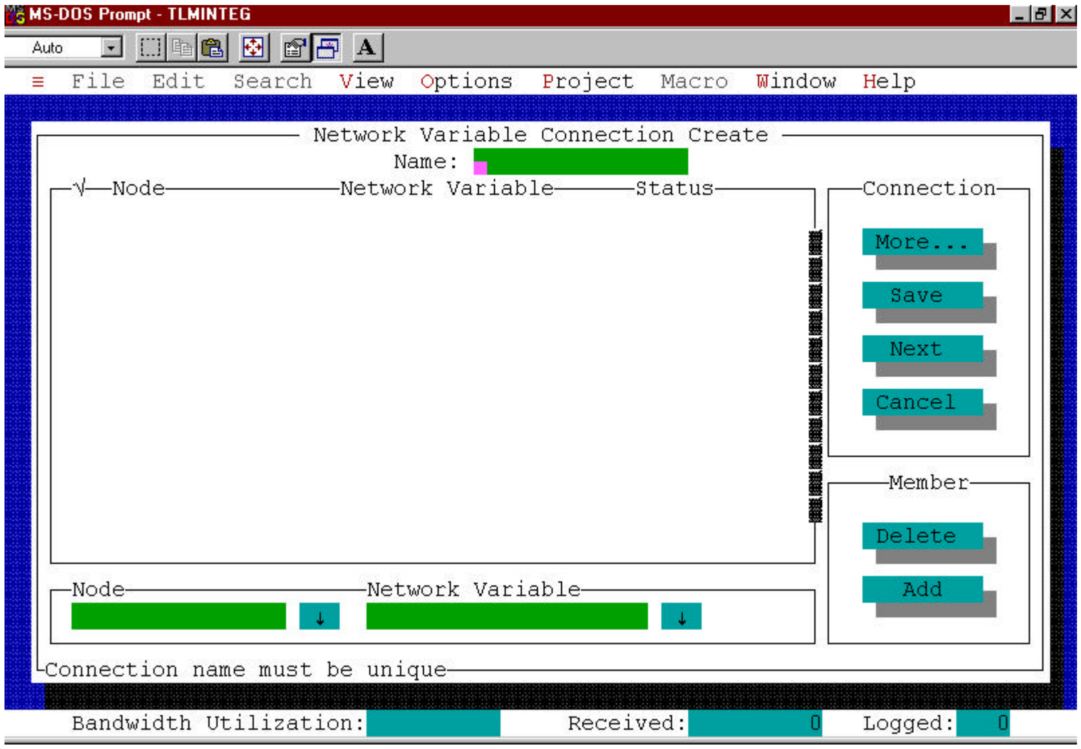


Figure 9: The Network Variable Connection Window

From this window the ***connections*** will be created. The path to arrive to this window from the Navigator home page is: Network then Connection, Net Variable and create.

By pulling down the ***node*** button, the user will see all the nodes that have been loaded. After having selected a node, the user can pull down the ***Network Variable*** menu and choose the variable he wants to be linked. When ***Add*** is pressed, the selection that the user just made is displayed into the main window. Then the user has to display by the same way all the variables he wants to be linked to the first one. When this is finished, and after having given a ***Name*** to the connection, validate this selection by pressing the ***Save*** button.

There is an ***Option*** button which allows the user to choose the type of message:

- Service type
  - Application configured
  - Acknowledged
  - Unacknowledged
  - Unacknowledged repeated
- Authenticated
  - Application configured
  - Yes
  - No
- Priority
  - Application configured
  - Yes
  - No

7. The Network Management Window

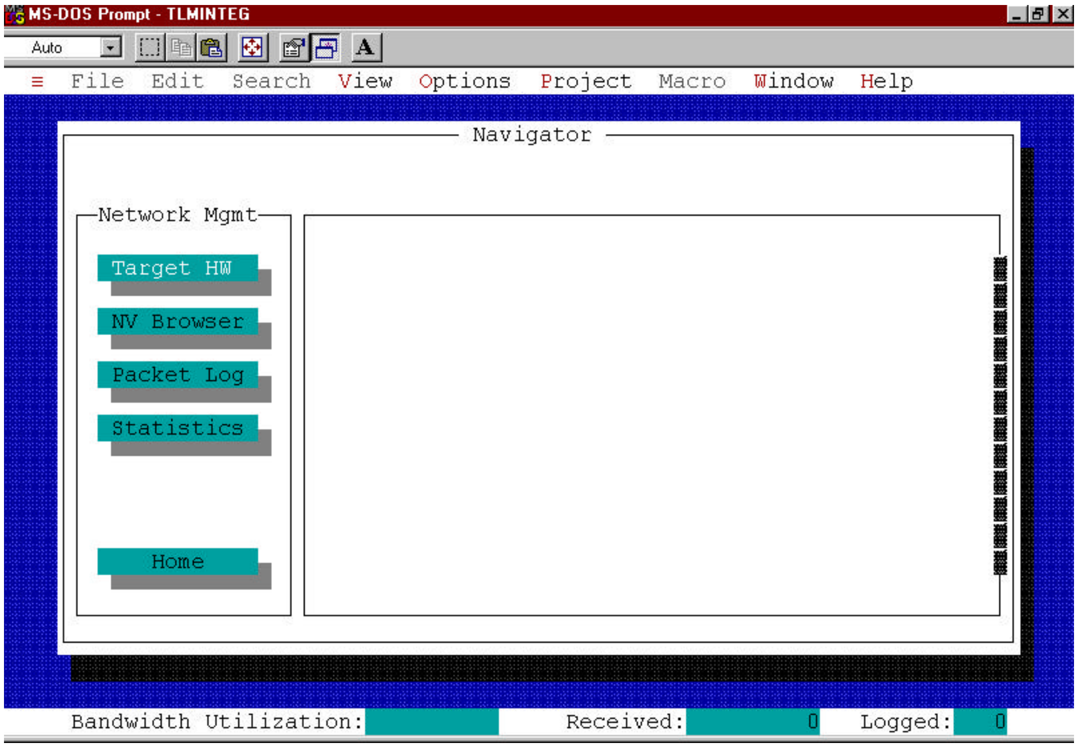


Figure 10: The Network Management Window

This window is very useful when the user wants to see how the network is performing.

On the ***Target Hardware*** there is the ***Network Manager*** and the ***Protocol Analyzer***. They allow the user to create the network.

The ***NV Browser*** window is very useful when it comes to tests (see next page for more details).

At last, the ***Statistic*** button is very useful to see the data flow through the network.

Note: the line at the bottom of the screen displays the number of packets sent and received through the network.

8. The NV Browser Window

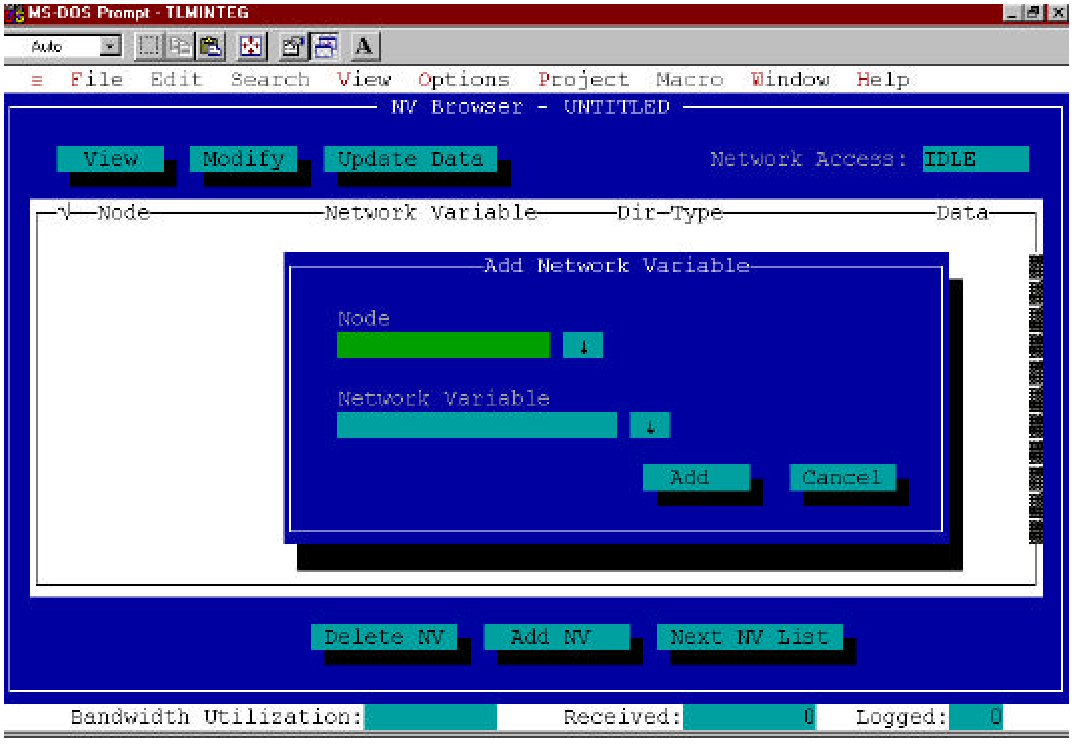


Figure 11: The NV Browser Window

From this window, the user can select some network variables from nodes and either check their value or decide which value to assign to the variable. It is very useful when the user wants to check that a program is performing properly or when a demonstration is needed.

First the user has to Add the Network Variables to the main screen and then, by ticking some of them, the user can either View the current value, Modify it and choose the new value or Update regularly the value of the data selected.

Note: When the user wants to modify the value of a network variable, it is better to stop or halt the current program. If a program is running and is updating the Network Variables, the modified value will not remain in the node. As the user does not update the data, the value he wanted to be loaded in the node will be displayed in the screen even thought it is not the one in the node.

C. Summary

Once the main concept of building/loading nodes and binding SNVTs is understood, the basis of the software is quite easy to master. However, as the user is working with other devices (Flex I/O cards, serial gateways…), the user still has to learn all the “tips” about the particular material used.

V. Various Tips to Use the Equipment

A. Introduction

This chapter addresses in turn some problems we encountered during this project:

- jamming tendency of the Flex I/O boards,
- conversion from source code to ROM or Flash Image files,
- EEPROM burning.

B. About the Flex I/O boards

The Flex I/O boards have to be handled with care. They have two serious drawbacks. The first one is that the software locks up every time there is something wrong, the second one is that, although they are short-circuit protected, the fuses blow easily.

As far as we know, the IEC boards lock up for two reasons:

- the memory allocation has not been well defined, i.e.the program is incompletely or improperly burned.
- a program has been targeted towards the node while a different programwas already in the memory. The conflict that follows completely disables the board.

In order to overcome these problems, an AT29C256 flash memory hosting a reset program called **EEBLANK** has to be inserted into the socket (board switched off). When the board is powered up again, the service led blinks during 3 seconds and then goes ON. The board is now clean.

Note: it is very important to note that there is no way to differentiate a locked-up board from a clean one, since in both cases the service led is ON. To assess the state of the board, the user can try to reinstall the node. To do so, the user has to go into the **App Images** menu, select **Install** in the **Target HW** menu, and press the service pin button. If the board is jammed, the Network management tool will display an error message.

Even though there is a fuse protecting the board, any testing on those outputs has to be carried out with extreme care, as the output voltage is usually 20V. In case of a short circuit, if the jumpers JP9 or JP10 are configured in the Pulse Width Modulation Mode, the fuse will blow. This fuse is distributed by DIGIKEY and is situated next to the POWER LED. Please refer to the Flex I/O Users Manual page 4 for the map of the board with the exact placement of the fuse. In particular, the appearance of the fuse is deceiving and it can easily be mistaken for a capacitor.

However the service LED is usually very useful to know in which state the board is. The three most common states are:

- LED on after a two seconds off period: the node is application less or the board is jammed
- LED blinking at one-second period rate: There is an unconfigured program in the flash memory
- LED off: the board has a program loaded and is ready to run or is running.

(See appendix E for more information)

C. The NRI and NEI files

These two types of file can be used for the IEC boards. NRI stands for Neuron ROM Image, while NEI means EEPROM or Flash Image. The user can use any of these two kinds of program, as long as the correct memory settings, i.e. the repartition between RAM, I/O, ROM and EEPROM/Flash memories, has been done.

The file, whether it is in NRI or in NRI format, has to be burned in the EEPROM. Once the program has been created, debugged and compiled, it needs to be targeted on a custom node. When the automatic build on this node will be started, the Network Management will have a BIG failure. This is normal, since a program can not be loaded into a custom node through the network, but it has to be burned on the memory instead. It is nevertheless necessary to do this trick, because it allows to update the old product files and to generate the latest program image file.

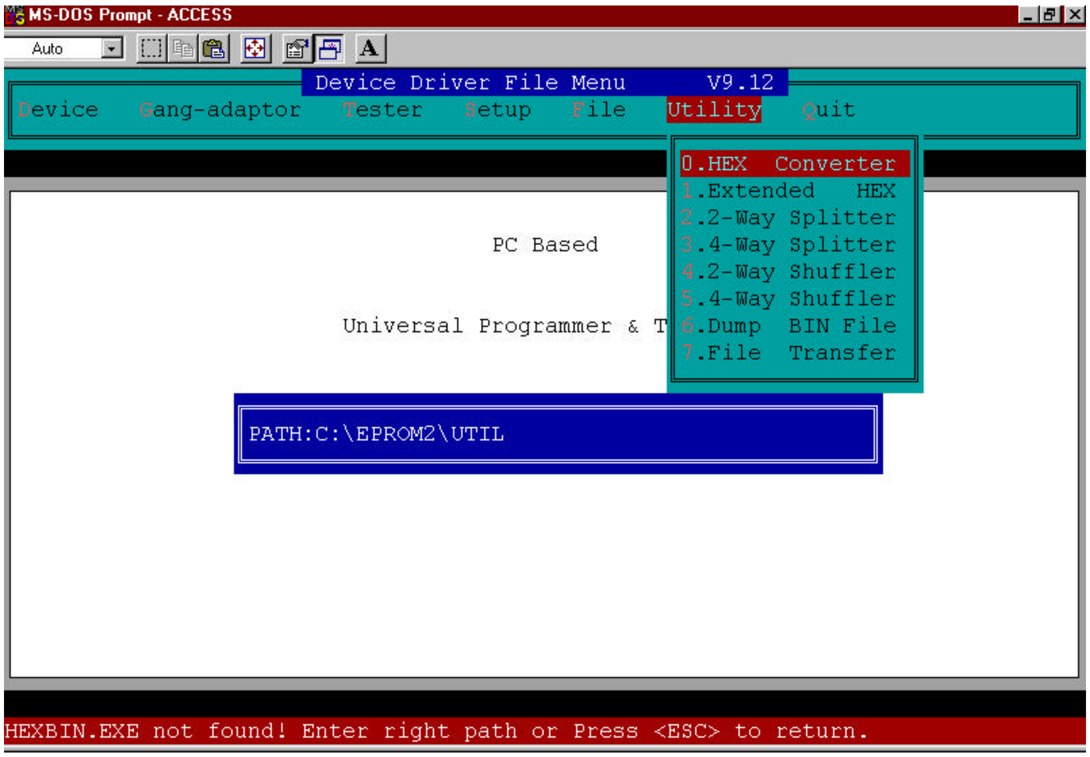
Under **Application Node** and then under **Node Spec** select the program you want to have converted in NEI format and then click on **Export**. Even though both types are acceptable, it is better to use the NEI format because this is the one the user is supposed to load in a flash memory.

D. How to burn the ATMEL AT29C256 flash memory

The equipment required to burn the flash memory is installed at the AUV lab. It is provided by Modular Circuit Technology Inc., a subsidiary company of JDR Microdevices, and runs on a PC computer. It consists of a software (the latest version available is V9.12), a Mod-Emup universal programmer and tester, and an MUP-PLCC 512 adaptor socket. The adaptor is required because the burner was originally designed to program DIP (Dual Inline Package) memories, whereas the ATMEL AT29C256 flash memory is coming in a PLCC (Plastic Lid Cover Carriage) format.

It is very important to note that the EEPROM programming software only works efficiently under a DOS environment. If it is run under Windows, “noises” will deteriorate the signal during the programming of the flash memory and there will be errors in the downloaded program.

The software is installed under the **C:\eprom2** directory. By typing **access**, the user starts the software and reaches the screen below.



```
MS-DOS Prompt - ACCESS
                       Device Driver File Menu        V9.12
 Device    Gang-adaptor   Tester    Setup    File   Utility   Quit
                                                     0.HEX   Converter
                                                     1.Extended    HEX
                                                     2.2-Way Splitter
                       PC Based                      3.4-Way Splitter
                                                     4.2-Way Shuffler
                                                     5.4-Way Shuffler
              Universal Programmer & T               6.Dump   BIN File
                                                     7.File   Transfer

                  PATH:C:\EPROM2\UTIL

HEXBIN.EXE not found! Enter right path or Press <ESC> to return.
```

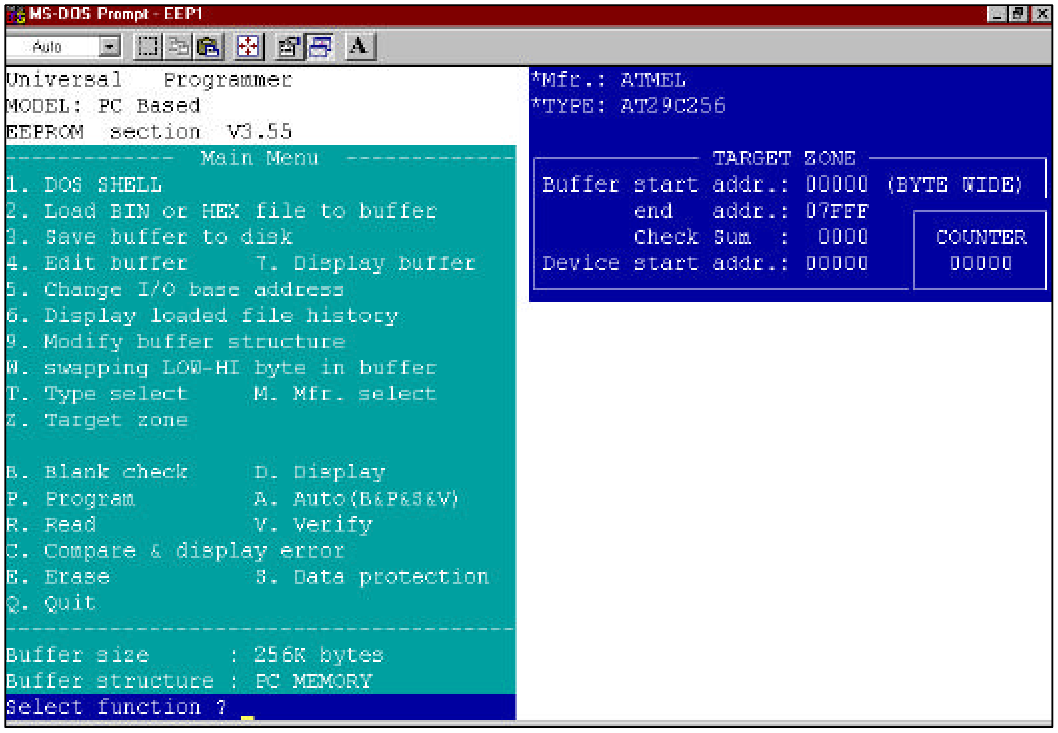
Figure 12: The Device Driver Window

First of all, the NEI or NRI file created by the user has to be transformed from an Intel hexadecimal (I Hex) to a binary (Bin) format. This can be done by selecting **Utility** and **Hex Converter** then by typing the file name and selecting the Intel Hex input format.

HEXBIN.EXE is under the directory C:\EPROM2\UTIL

Once this has been done, the user has to select the device. The AT29C256 is registered as an EEPROM manufactured by ATMEL and it is the device type number 21.

By pressing the enter key with that selection, the user will see the following screen:



```
MS-DOS Prompt - EEP1
Universal   Programmer                 *Mfr.: ATMEL
MODEL: PC Based                        *TYPE: AT29C256
EEPROM  section  V3.55
------------- Main Menu -------------     TARGET ZONE
1. DOS SHELL                           Buffer start addr.: 00000 (BYTE WIDE)
2. Load BIN or HEX file to buffer             end   addr.: 07FFF
3. Save buffer to disk                        Check Sum  :  0000    COUNTER
4. Edit buffer         7. Display buffer Device start addr.: 00000   00000
5. Change I/O base address
6. Display loaded file history
9. Modify buffer structure
W. swapping LOW-HI byte in buffer
T. Type select         M. Mfr. select
Z. Target zone

B. Blank check         D. Display
P. Program             A. Auto(B&P&S&V)
R. Read                V. Verify
C. Compare & display error
E. Erase               S. Data protection
Q. Quit
-------------------------------------
Buffer size        : 256K bytes
Buffer structure : PC MEMORY
Select function ? _
```

Figure 13: The EEPROM Burner Main Menu Window

The first commands (from 1 to Z) are acting on the computer.
The lower ones (from B through S) are acting on the device.

First of all it is always recommended to check if the memory is empty (command B) and if not, to save the program into a disk (command R and then 3) before to erase the flash memory (command E). In order to burn a program in a Flash memory, the user has to load the file into the buffer (command 2) and then to program the buffer into the chip (command P). There will be an automatic verification after the loading and if all goes well the program is loaded successfully in the chip.

E. Summary

These three troubles have been very tedious to assess and to overcome, hence, as we assumed that the students who are going to take over this project may run into the same problems, we decided to prevent this inconvenience by including this section. Technical support for diagnosing these problems was ineffective

Another thing, maybe more important, is that when we arrived on the job, we could not understand which file we had to use and when. That is why we have enclosed an explanation of our directory in appendix F.

VI.Motors

A. Introduction

The motors are used on the test bench to prepare for controlling the propellers of the AUV. They are brush motors and are monitored by D/A converters. In order to control the motors through the network, some flex I/O boards from IEC Technologies have been chosen for anolog/digital signal conversion.

B. Methodology

The output selected on the board for this purpose is the **Analog Output** (cf. Flex I/O Users Manual page 4), and the network variable considered is either "*nvi_raw_analog_X*" or "*nvi_analog_X*", where X stands for the channel selected.

For this application, two programs have been used. We created the first one and the other one, provided by Echelon, is burnt in the flash memory labeled ***Fancoil***. When a node like the IEC flex IO is used, an interface file has to be loaded into it. This interface file has been created by Echelon and describes the outputs/inputs of the printed circuit board. It will be very useful during the binding.

In our program we have used "*nvi_raw_analog_X*" instead of "*nvi_analog_X*", because the latter is less convenient than "*nvi_raw_analog_X*". It is of a structure type, which we did not need; nevertheless it can be used as long as the value entered for that field is less than 200.

C. The program **motors.nc** *(abstract)*

```
#pragma enable_sd_nv_names                /* displays network variable names on
                                          the screen while using the network
                                          management tools */

#include <snvt_rq.h>                      /* include requirements library */
#include <snvt_lev.h>                     /* include level discrete library */

IO_4 input bit ioSwitch;                  /* usage of the IO_4 switch */
IO_0 output pulsewidth clock (7) ioLed0;  /* usage of the IO_0 led */

network output SNVT_count nvo_l_prop;     /*left propeller */
network output SNVT_count nvo_r_prop;     /*right propeller */

mtimer frame1  =  10000;                  /*millisecond timer set to 10 sec */
```

```
when (reset)
{
      io_change_init (ioSwitch);                   /*value of IO_4 read */
      io_out (ioLed0,1);                           /* led set to 1 */
      nvo_l_prop     = 0;                          /*left propeller rpm = 0*/
      nvo_r_prop     = 0;                          /*right propeller rpm = 0*/
}

when (timer_expires(frame1))
{
      nvo_l_prop     = 800;
      nvo_r_prop     = 800;

      io_out (ioLed0, 20);
}

when (io_changes (ioSwitch))
{

      nvo_l_prop     = 0;
      nvo_r_prop     = 0;

      io_out (ioLed0, 0);
}
```

D. Connections

Once the Fancoil program has been loaded with the interface file on the Flex I/O board and motors on the emulator, some connections have to be made. On motors, the two SNVTs are nvo_l_prop and nvo_r_prop and on the flex I/O, the inputs are nvi_raw_analog 1 and 2.

From the Network variable Connection window, add the variables nvo_l_prop from motors and nvi_raw_analog _1 from fancoil, save this connection by giving it a name and pressing Save. Do the same thing for nvo_r_prop and nvi_raw_analog _2, save it with another name.

Once this is done, invoke the ***Build*** command and the connections will be done.

E. Summary

This program allows the user to command the motors. There is still a need to send a feedback from the motors in the network, so that the rotational speed of the propellers can be checked.

VII. Servos

A. Introduction

The servos are used on the test bench to prepare for controlling the planes, i.e. the fins and the rudders, of the AUV. They are driven by a Pulse Width Modulated (PWM) signal. As with the motors, some Flex I/O boards from IEC Technologies have been chosen in order to control the servos through the network.

B. Methodology

The output selected on the board to monitor the servos is the Open Drain Output (please refer to the Flex I/O Users Manual, page 4). Through a set of jumpers (marked JP9 and JP10 on the board), the output can be connected directly to the pins IO_0 and IO_1 of the Neuron chip.

For this application, two programs have been created. The first one (called **servo1**) is run on one emulator of the LonBuilder. Its purpose is to send a network variable taking a value ranging from 0 to 65535 by increments of 5000 every second.

The second program (called **servo2**) uses a clock to generate a pulse width modulated output. The clock chosen is a long number 6 clock, which means it has a frequency of 1.19 Hz (refer to the Neuron C Reference Guide page 8-56).

The network variable will change the width of the modulation.

C. The programs

1. fin_emu.nc

```
IO_4 input bit stop;
network output SNVT_count order;
mtimer repeating pulse =30;

when (reset)
{
        ordre = 0;
}

when (timer_expires (pulse))
{
   ordre +=20;
}
```

```
when (io_changes (stop))
{
        pulse = 0;
}
```

2. fin_flex.nc

```
network input SNVT_count pwm;
IO_0 output pulsewidth long clock(1) top;

when (nv_update_occurs(pwm))
{
        io_out(top, pwm);
}
```

D. Connections

Once **Servo1** has been loaded in the emulator and **Servo2** on the Flex I/O board, some connections have to be done. On **Servo1**, the variable name is **ordre** and on the flex I/O, the variable name is **pwm**.

The user has to bind these two variables prior to running the program.

E. Summary

These programs allow the user to command the servos. However, it is still necessary to find a servo that corresponds to the output clock rate given by the Neuron Chip. Otherwise, the servo will not be able to complete its nominal angular displacement range.

VIII. Conclusions and Recommendations

A. Introduction

Although this project demonstrated the viability of a LonWorks-based distributed control network and its ability to control some of the devices implemented in the *Phoenix* AUV, much more work is still needed. The research of this project represents just a beginning, and additional research and testing have to be completed before a working LonWorks networked control system can be implemented onboard the *Phoenix*.

B. Research conclusions

During this project, we had to implement the LonWorks technology for the AUV through a workbench model. The objectives were to achieve the 10 Hz data processing rate and to obtain a more flexible network regarding the addition, removal or upgrade of devices. As a matter of fact, the LonWorks technology is not really straightforward. Some troubles appear because of the merges of technologies (Echelon, IEC, Modular Circuit Technology, ATMEL, Advanced Motion Control…), and it is consequently very difficult to find somebody competent on all the technologies that can help satisfactorily.

As the LonBuilder Software is very specific to the LonWorks Technology, it is important to understand it thoroughly. The LonWorks tutorial helps a lot but is not accurate enough. Although it does prepare completely the user to run programs on emulators, when it comes to work with other devices (such as the Flex I/O cards) the user is left helpless. In order to prepare other students to take over our job and to work with Flex I/O boards, we included in this report our project directory, as well as some explanations on our programs and a step-by-step guide.

We ran into some troubles that took us sometimes several weeks to solve. The explanation and the solution of those special problems are included as well in this report.

Within one month, a motivated student should be able to go through the LonTalk seminar and to run motors and servos.

C. Recommendations for future work

The most important goal to achieve right now is to get the serial gateways to work, because this step represents the missing link between the current model network and a real distributed control network onboard the AUV. Once this task is done, the programming job begins. LonWorks will hopefully become user-friendlier and the project should be very interesting. From now on, the main focus of this project is programming, and since we felt sometimes that our limited experience on computer science was a curb on the progression of the project, we think that it would be most productive to find a student involved as much as possible in computer science engineering.

D. Summary

This project was a great occasion for us to do our final work as students in a complex research environment involving up-to-date technology. It has been a technical and human challenge, and we have been glad to accept it.

It allowed us to discover the United States and their way of life, and the overall experience we had here was very formative and made us encounter a science we had never worked on before.

The LonWorks is a very interesting technology once you get to know it. It allows the user to create its own network and will probably be predominant in the future.

List of References

Brutzman, Donald P., *A Virtual World for an Autonomous Undersea Vehicle*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, December 1994.
Available at: http://www.cs.nps.navy.mil/research/auv

Brutzman, D. P., *NPS Phoenix AUV Software Integration and In-Water Testing*, AUV 96, Monterey, CA, June 1996.

Byrnes, R.B., *The Rational Behavior Software Architecture for Intelligent Ships, An Approach to Mission and Motion Control,* Naval Engineers Journal, March 1996.

Byrnes, R.B., *The Rational Behavior Model: A Multi-Paradigm, Tri-Level Software Architecture for the Control of Autonomous Vehicles*, Dissertation, Naval Postgraduate School, Monterey, CA, March 1993.

Echelon Corp., *Lon Builder Hardware Guide,* Revision 6, Palo Alto, CA.

Echelon Corp., *Lon Builder User's Guide,* Revision 3, Palo Alto, CA.

Echelon Corp., *LonWorks Host Application Programmer's guide,* Revision 2, Palo Alto, CA.

Echelon Corp., *LonWorks Training Manual*, Palo Alto, CA, 1997.

Echelon Corp., *Neuron Chip Data Book,* Palo Alto, CA, February 1995.

Echelon Corp., *Neuron C Programmer's guide,* Revision 4, Palo Alto, CA.

Echelon Corp., *Neuron C Reference's guide,* Revision 2, Palo Alto, CA.

Echelon Corp., *Serial LonTalk Adapter and Serial Gateway User's Guide,* Revision 9, Palo Alto, CA.

Healey, A. J., *Mission Planning, Execution, and Data Analysis for the NPS AUV II Autonomous Underwater Vehicle*, Proceedings of the First IARP Workshop on Mobile Robots for Subsea Environments, Monterey, CA, October 1990.

Healey, A. J., *Research on an Autonomous Vehicle at the Naval Postgraduate School*, Naval Research Reviews, Office of Naval Research, Washington, D.C., Volume XLIV, Number 1, Spring 1992.

Holden, M.J., *Ada Implementation of Concurrent Execution for Multiple Tasks in the Strategic and Tactical Level of the Rational Behavior Model for the NPS AUV*, Master's Thesis, Naval Posgraduate School, Monterey, CA, March 1995.
Available at http://www.cs.nps.navy.mil/research/auv

Intelligent Technologies Corporation, *Flexible IO Users Manual*, Golden, Co.

LonMark Interoperability Association, *LonMark Layer 1-6 Interoperability Guidelines,* Version 3.0, Palo Alto, CA

Marco, D. B., *Autonomous Control of Underwater Vehicles and Local Area Maneuvering*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, September 1996.
Available at http://www.cs.nps.navy.mil/research/auv

Young, F., *Phoenix Autonomous Underwater Vehicle (AUV): Networked Control of Multiple Analog and Digital Devices Using LonTalk,* Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1997.

Initial distribution list

1. Dr. Donald P. Brutzman, Code UW/Br Undersea Warfare Academic Group Naval Postgraduate School Monterey, CA, 93943-5100	2
2. Dr. Anthony J. Healey, Code ME/Hy Mechanical Engineering Department Naval Postgraduate School Monterey, CA, 93943-5100	2
3. Didier Léandri Directeur du laboratoire CPSI ENIT 47, Avenue d’Azereix BP 1629 65016 Tarbes Cedex, France	3
4. CDR Michael J. Holden, USN, Code CS/Hm Computer Science Department Naval Postgraduate School Monterey, CA, 93943-5100	1
5. LCDR Jeffery S. Riedel, Code ME/34 Mechanical Engineering Department Naval Postgraduate School Monterey, CA, 93943-5000	1
6. David Marco, Code ME/Ma Mechanical Engineering Department Naval Postgraduate School Monterey, CA, 93943-5000	1
7. Bahadir Behazay, Turkish Navy Mechanical Engineering Department Naval Postgraduate School Monterey, CA, 93943-5000	1
8. Helen Greiner IS Robotics, Inc. 222, Mc Grath Highway Twin City Office Center, suite 6 Somerville, MA 02143	1

9. Laurent Beguery Le Moura 40260 Castets, France	2
10. Alexandre David 58, rue Georges Desclaude 17100 Saintes, France	2
11. Bibliothèque de l'ENIT 47, Avenue d'Azereix BP 1629 65016 Tarbes Cedex, France	1

Appendix A - Code Listings

Motors.nc

```
/* program to drive two motors with a Flex I/O card. Alexandre and Laurent August 1998*/
#pragma enable_sd_nv_names

#include <snvt_rq.h>
#include <snvt_lev.h>

IO_4 input bit ioSwitch;
IO_0 output pulsewidth clock (7) ioLed0;

network output SNVT_count nvo_l_prop;          /*left propeller*/
network output SNVT_count nvo_r_prop;          /*right propeller*/

mtimer frame1  =  10000;
mtimer frame2  =  20000;
mtimer frame3  =  30000;
mtimer frame4  =  40000;
mtimer frame5  =  50000;

when (reset)
{
        io_change_init (ioSwitch);
        io_out (ioLed0,1);
        nvo_l_prop      = 0;            /*left propeller*/
        nvo_r_prop      = 0;            /*right propeller*/
}

when (timer_expires(frame1))
{
        nvo_l_prop      = 800;
        nvo_r_prop      = 800;
        io_out (ioLed0, 20);
}

when (timer_expires(frame2))
{
        nvo_l_prop      = 2048;
        nvo_r_prop      = 2048;
        io_out (ioLed0, 127);
}
```

```
when (timer_expires(frame3))
{
        nvo_l_prop      = 4095;
        nvo_r_prop      = 4095;
        io_out (ioLed0, 255);
}

when (timer_expires(frame4))
{
        nvo_l_prop      = 0;
        nvo_r_prop      = 2048;
        io_out (ioLed0, 127);
}

when (timer_expires(frame5))
{
        nvo_l_prop      = 4095;
        nvo_r_prop      = 0;
        io_out (ioLed0, 255);
}

when (io_changes (ioSwitch))
{
        nvo_l_prop      = 0;
        nvo_r_prop      = 0;
        io_out (ioLed0, 0);
}
```

Fancoil

```
/******************************************************************
*       Filename:       FANCOIL.NC
*
*       Intelligent Energy Corporation
*       607 Tenth Street, Suite 203
*       GOlden, CO 80401
*       (303) 277-1503
*
*       Written By:     Stewart Goldenberg
*
*       Version:        1.21
*       Last Update:    6/25/97
*       Modified:       For Compatiblity with LonMark Fan Coil Unit
*
*       Description:    FLEX I/O Application
*
******************************************************************/

/******************************************************************
*   Software Revision
******************************************************************/

#define SOFTWARE_REVISION "FLEXIO 1.21"

/******************************************************************
    Version History
*******************************************************************

1.0:    First release.

******************************************************************/

/******************************************************************
*   Compiler Directives
******************************************************************/

#pragma enable_sd_nv_names
#pragma set_node_sd_string "Flexible IO control module configured to support 2
LonMark \
fan coil profiles.  nvi_analog_1 and nvi_analog_2 are used as inputs \
from the fan coil profile outputs nvoFanSpeed.  nvo_temp_1 and \
nvo_temp_2 are used in conjunction with an ATP E1004 series thermistor \
to get the nviSpaceTemp for the fan coil profile. \
 \
The other FLEXIO inputs and outputs are still available to get and set \
generic analog and digital values."

/****************************************************************
*   Include Directives
****************************************************************/

   // Include Neuron C headers
#include <stdlib.h>
#include <snvt_lev.h>
#include <addrdefs.h>
#include <access.h>

   // Include type and constant definitions, I/O assignments,
   // and low-level functions for the FLEX I/O
#include "FLEXIO.H"
#include "iectypes.h"

/****************************************************************
*   Macros
****************************************************************/

   /* Convert degrees C to Degrees K (SNVT_temp_p) */
#define ConvertCtoK( c ) ( ((c) * 10UL) + 2740UL)

/****************************************************************
*   Definitions
****************************************************************/

#define ONE_MINUTE 60058

/****************************************************************
*   Network Variables
****************************************************************/

network input  sd_string("ST_ON except for ST_OFF")  SNVT_lev_disc  nvi_relay_1
= ST_OFF;
network input  sd_string("ST_ON except for ST_OFF")  SNVT_lev_disc  nvi_relay_2
= ST_OFF;
network input  sd_string("ST_ON except for ST_OFF")  SNVT_lev_disc  nvi_relay_3
= ST_OFF;
network input  sd_string("ST_ON except for ST_OFF")  SNVT_lev_disc  nvi_relay_4
= ST_OFF;

network input  sd_string("ST_ON except for ST_OFF")  SNVT_lev_disc
```

```
nvi_open_drain_1 = ST_OFF;
network input  sd_string("ST_ON except for ST_OFF")  SNVT_lev_disc
nvi_open_drain_2 = ST_OFF;

network input  sd_string("0-4095 = full range, also controls PWM outputs if
jumpered for PWM mode") SNVT_count  nvi_raw_analog_1 = 0;
network input  sd_string("0-4095 = full range, also controls PWM outputs if
jumpered for PWM mode") SNVT_count  nvi_raw_analog_2 = 0;

network input  sd_string("0-100 = full range, also controls relay, also controls
PWM outputs if jumpered for PWM mode")  SNVT_switch  nvi_analog_1 = {0,0};
network input  sd_string("0-100 = full range, also controls relay, also controls
PWM outputs if jumpered for PWM mode")  SNVT_switch  nvi_analog_2 = {0,0};

network output sd_string("two state output")  SNVT_lev_disc  nvo_contact_1 =
ST_OFF;
network output sd_string("two state output")  SNVT_lev_disc  nvo_contact_2 =
ST_OFF;
network output sd_string("two state output")  SNVT_lev_disc  nvo_contact_3 =
ST_OFF;
network output sd_string("two state output")  SNVT_lev_disc  nvo_contact_4 =
ST_OFF;

network output sd_string("meaningful only if jumpers are set for thermistor")
SNVT_temp_p nvo_temp_1 = ConvertCtoK(0);
network output sd_string("meaningful only if jumpers are set for thermistor")
SNVT_temp_p nvo_temp_2 = ConvertCtoK(0);
network output sd_string("meaningful only if jumpers are set for thermistor")
SNVT_temp_p nvo_temp_3 = ConvertCtoK(0);
network output sd_string("meaningful only if jumpers are set for thermistor")
SNVT_temp_p nvo_temp_4 = ConvertCtoK(0);

network output sd_string("0-4095 = full range")  SNVT_count  nvo_raw_analog_1 =
0;
network output sd_string("0-4095 = full range")  SNVT_count  nvo_raw_analog_2 =
0;
network output sd_string("0-4095 = full range")  SNVT_count  nvo_raw_analog_3 =
0;
network output sd_string("0-4095 = full range")  SNVT_count  nvo_raw_analog_4 =
0;

network output sd_string("0-100 = full range")  SNVT_lev_cont  nvo_analog_1 = 0;
network output sd_string("0-100 = full range")  SNVT_lev_cont  nvo_analog_2 = 0;
network output sd_string("0-100 = full range")  SNVT_lev_cont  nvo_analog_3 = 0;
network output sd_string("0-100 = full range")  SNVT_lev_cont  nvo_analog_4 = 0;
network input  sd_string("A/D scan time in seconds, default = 5 sec") config
SNVT_count  nci_scan_time = 5;

network input  sd_string("temperature calibration, default = none")  config
SNVT_zerospan nci_temp_1_cal = { 0, 2000 };
network input  sd_string("temperature calibration, default = none")  config
SNVT_zerospan nci_temp_2_cal = { 0, 2000 };
network input  sd_string("temperature calibration, default = none")  config
SNVT_zerospan nci_temp_3_cal = { 0, 2000 };
network input  sd_string("temperature calibration, default = none")  config
SNVT_zerospan nci_temp_4_cal = { 0, 2000 };

network input sd_string("Hearbeat (Sec)") eeprom SNVT_count    nciHeartbeat = 4
* 60;
network output sync  sd_string("Node Heartbeat") SNVT_lev_disc nvoHeartbeat;

/****************************************************************
*   Timers
****************************************************************/

stimer repeating  A2D_timer;              // A/D loop timer
mtimer          tmrStagger;            // For staggering heartbeats.
stimer repeating  tmrHeartbeat;           // Maintenance operations.

/****************************************************************
*   Object Code Software Revision
****************************************************************/

const char software_revision[]=SOFTWARE_REVISION;

/****************************************************************
* Global Variables
****************************************************************/

word8 FLEX_IO_Node;
word8 FLEX_IO_Subnet;

/****************************************************************
*   When Clauses
****************************************************************/

when(reset) {
        /* Enable conversion on all four analog input channels */
    const static ANALOG_CHANNEL_SELECT channels = { 0, 0, 0, 1, 1, 1, 1, 0 };
    domain_struct domain_info;
    /* NOTE:  The "reset" when clause should be kept first. */


       /* Initialize Digital Outputs -- Initial states are all off */
    InitializeDigitalOut(0);

       /* Initialize Analog-to-Digital Converter */
    InitializeAnalogIn(&channels);


       /* Initialize all digital input states to inactive */
    io_change_init(IO_input1, INPUT_INACTIVE);
    io_change_init(IO_input2, INPUT_INACTIVE);
    io_change_init(IO_input3, INPUT_INACTIVE);
    io_change_init(IO_input4, INPUT_INACTIVE);

       /* Start A/D Acquisition Timer */
    A2D_timer = nci_scan_time;

   // Retrieve Domain/Subnet/Node of our node.
   domain_info = *(access_domain(0));
   FLEX_IO_Subnet = domain_info.subnet;
   FLEX_IO_Node   = domain_info.node;

   // Stagger starts the the maintenance timer at an offset to the rest of the
   // network.
   tmrStagger = ( (((word16) FLEX_IO_Node   * 117) % ONE_MINUTE) * 4 + //
Network offset.
                ((word16)  FLEX_IO_Subnet * 57)) % ONE_MINUTE;

    }

when(nv_update_occurs(nci_scan_time)) {
    A2D_timer = nci_scan_time;
    }

when(timer_expires(A2D_timer)) {
    RAW_ANALOG_INPUT *raw_analog_input;

 /* Linearization table for an ATP E1004 series thermistor */
    const static SNVT_temp_p linearization_table[] = {
       17320, 10060, 7630, 6230, 5220, 4410, 3710, 3070, 2500, 1920, 1360,
       770, 140, -590, -1480, -2790, -4910 };

       //Get raw values
    raw_analog_input = ReadAnalogInputs(ANALOG_INPUT_CHANNEL1, 4);
    nvo_raw_analog_1 = raw_analog_input[0] << 2u;
    nvo_raw_analog_2 = raw_analog_input[1] << 2u;
    nvo_raw_analog_3 = raw_analog_input[2] << 2u;
    nvo_raw_analog_4 = raw_analog_input[3] << 2u;

        //Convert raw values to SNVT_lev_cont
    nvo_analog_1 = (uint)muldiv(raw_analog_input[0], 200u, 1023u);
    nvo_analog_2 = (uint)muldiv(raw_analog_input[1], 200u, 1023u);
    nvo_analog_3 = (uint)muldiv(raw_analog_input[2], 200u, 1023u);
    nvo_analog_4 = (uint)muldiv(raw_analog_input[3], 200u, 1023u);

        //Convert raw values to SNVT_temp_p
    nvo_temp_1 =
        ConvertTemperature(raw_analog_input[0], &nci_temp_1_cal,
            linearization_table);
    nvo_temp_2 =
        ConvertTemperature(raw_analog_input[1], &nci_temp_2_cal,
            linearization_table);
    nvo_temp_3 =
        ConvertTemperature(raw_analog_input[2], &nci_temp_3_cal,
            linearization_table);
    nvo_temp_4 =
        ConvertTemperature(raw_analog_input[3], &nci_temp_4_cal,
            linearization_table);
    }

when(nv_update_occurs(nvi_analog_1)) {
    io_out(IO_analog_out1, (uint)(muldiv(nvi_analog_1.value, 255u, 200u)));
    DigitalBitOut(RELAY1_MASK, nvi_analog_1.state);
    }

when(nv_update_occurs(nvi_analog_2)) {
    io_out(IO_analog_out2, (uint)(muldiv(nvi_analog_2.value, 255u, 200u)));
    DigitalBitOut(RELAY2_MASK, nvi_analog_2.state);
    }

when(nv_update_occurs(nvi_raw_analog_1)) {
    io_out(IO_analog_out1, (uint)(nvi_raw_analog_1 >> 4u));
    }

when(nv_update_occurs(nvi_raw_analog_2)) {
    io_out(IO_analog_out2, (uint)(nvi_raw_analog_2 >> 4u));
    }

when(io_changes(IO_input1)) {
    nvo_contact_1 = ((input_value == INPUT_ACTIVE) ? ST_ON:ST_OFF);
```

```
   }

when(io_changes(IO_input2)) {
   nvo_contact_2 = ((input_value == INPUT_ACTIVE) ? ST_ON:ST_OFF);
   }

when(io_changes(IO_input3)) {
   nvo_contact_3 = ((input_value == INPUT_ACTIVE) ? ST_ON:ST_OFF);
   }

when(io_changes(IO_input4)) {
   nvo_contact_4 = ((input_value == INPUT_ACTIVE) ? ST_ON:ST_OFF);
   }

when(nv_update_occurs(nvi_open_drain_1)) {
   DigitalBitOut(OPEN_DRAIN1_MASK, nvi_open_drain_1);
   }

when(nv_update_occurs(nvi_open_drain_2)) {
   DigitalBitOut(OPEN_DRAIN2_MASK, nvi_open_drain_2);
   }

when(nv_update_occurs(nvi_relay_1)) {
   DigitalBitOut(RELAY1_MASK, nvi_relay_1);
   }

when(nv_update_occurs(nvi_relay_2)) {
   DigitalBitOut(RELAY2_MASK, nvi_relay_2);
   }

when(nv_update_occurs(nvi_relay_3)) {
   DigitalBitOut(RELAY3_MASK, nvi_relay_3);
   }

when(nv_update_occurs(nvi_relay_4)) {
   DigitalBitOut(RELAY4_MASK, nvi_relay_4);
   }

when (timer_expires(tmrStagger))
{
  tmrHeartbeat = nciHeartbeat;
}

when (timer_expires(tmrHeartbeat))
{
  nvoHeartbeat = ST_ON; // Force an NV update.
```

```
}

when (nv_update_occurs(nciHeartbeat))
{
  // Restart the heartbeat clock:
  // Stagger starts the the maintenance timer at an offset to the rest of the
  // network.
  tmrStagger = ( (((word16) FLEX_IO_Node   * 117) % ONE_MINUTE) * 4 + //
Network offset.
              ((word16)  FLEX_IO_Subnet * 57)) % ONE_MINUTE;
}

/*****************************************************************
*   END of File
*****************************************************************/
```

PWM.nc

```
/* program to create a PWM output on an emulator. Alexandre and Laurent August 1998*/

IO_0 output pulsewidth long clock(6) io_rudder;

unsigned long brightness;
mtimer repeating steps;
when (reset)
{
   brightness = 0;
   steps = 1000;
}
when (timer_expires(steps))
{
   brightness +=5000 ;
   io_out(io_rudder, brightness);
}
when (brightness == 65000)
{
      steps = 0;
      brightness = 0;
      io_out(io_rudder,  brightness);
}
```

Fin_emu.nc

```
/* program to drive one servo with a emulator and a Flex I/O card.
Alexandre and Laurent August 1998 (emulator part)*/
network input SNVT_count pwm;
IO_0 output pulsewidth long clock(1) top;

when (nv_update_occurs(pwm))
{
   io_out(top, pwm);
}
```

Fin_flex.nc

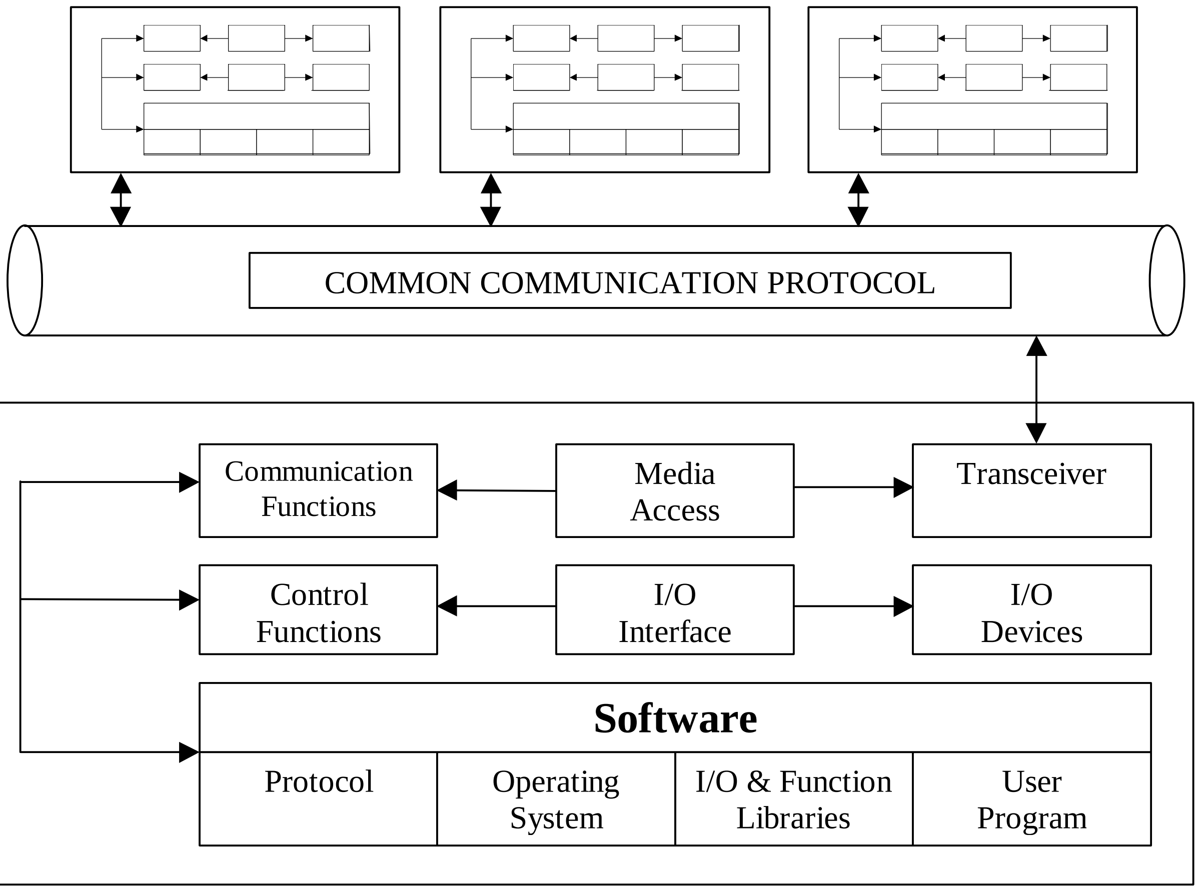
```
/* program to drive one servo with a emulator and a Flex I/O card.
Alexandre and Laurent August 1998 (Flex I/O part)*/

network input SNVT_count toto;

IO_0 output pulsewidth short clock(4) top;
when (nv_update_occurs(toto))
{
   io_out(top,  (short) toto);
}
```

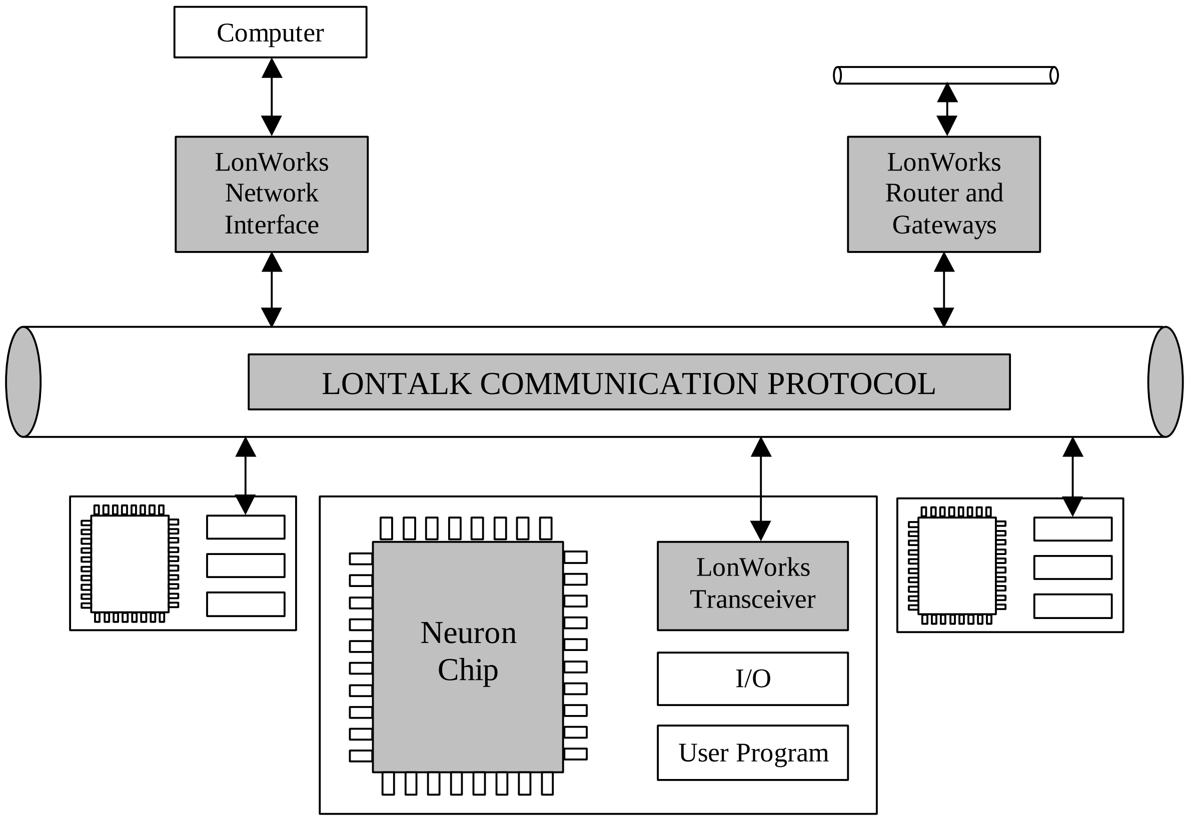
Appendix B – Control Networks

Part I: Requirements for a control network



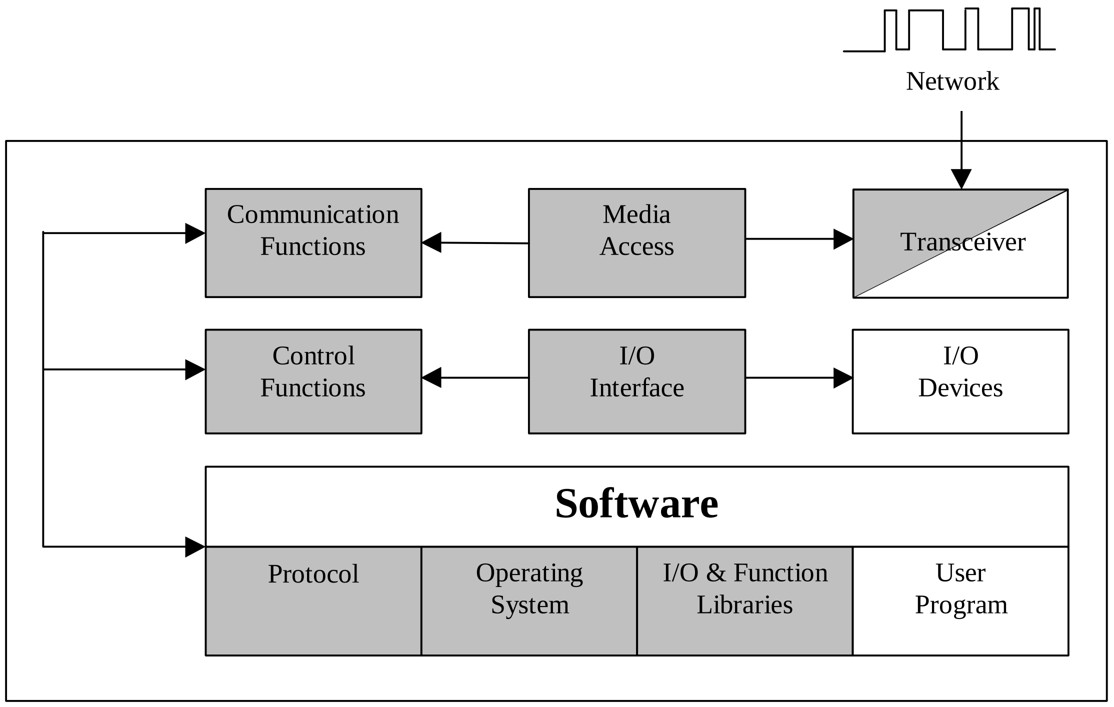
The figure above summarizes the parts of equipment needed to communicate via a network using the ISO/OSI common protocol.

Part II: One solution: LonWorks Technology


Computer
LonWorks Network Interface
LonWorks Router and Gateways
LONTALK COMMUNICATION PROTOCOL
Neuron Chip
LonWorks Transceiver
I/O
User Program

The figure above explains the pieces of Software and Hardware that Echelon proposes in its LonWorks technology as a solution to the problem stated in Part I.

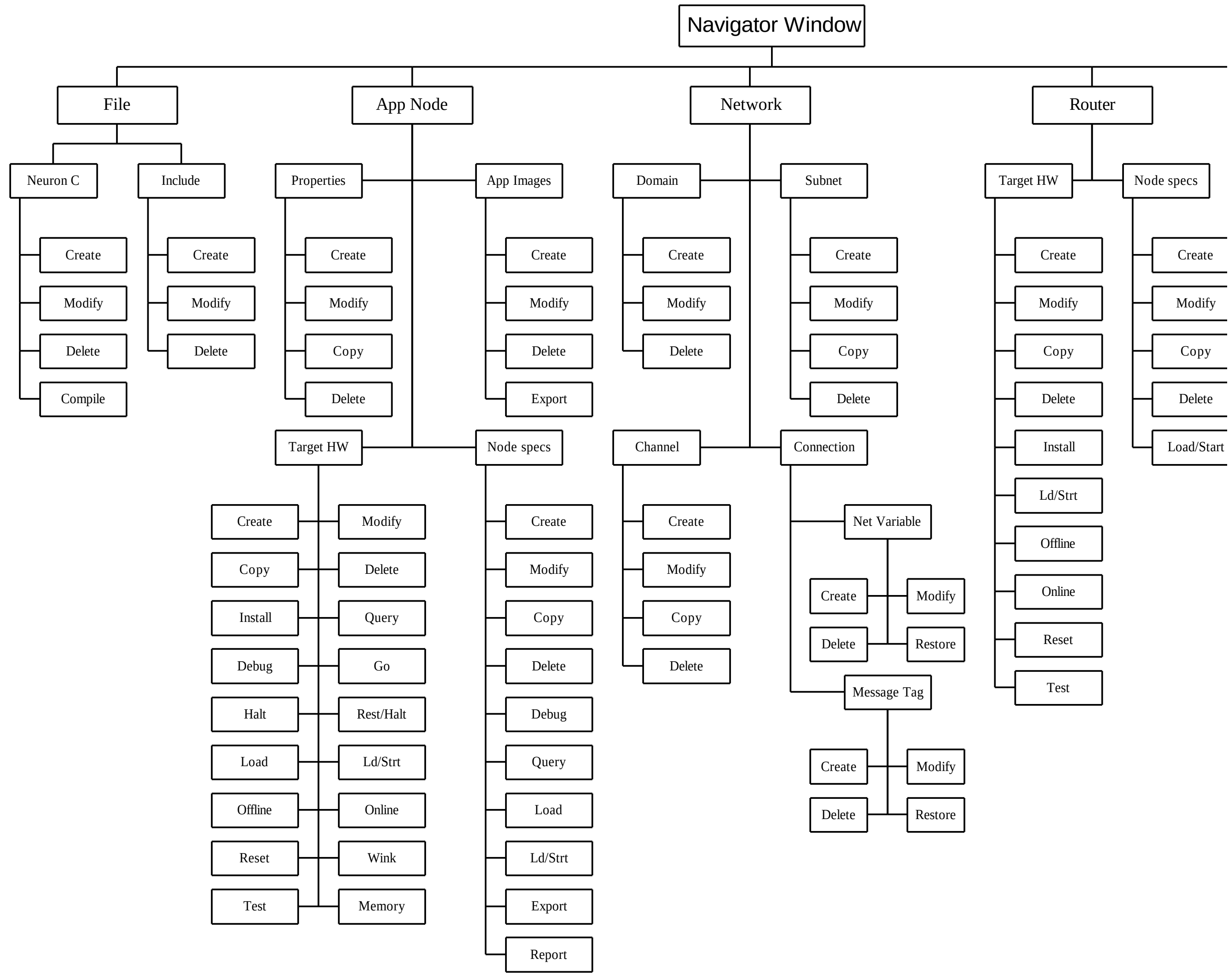
Appendix C - Basis of LonWorks Devices



▨ The Neuron Chip contains:

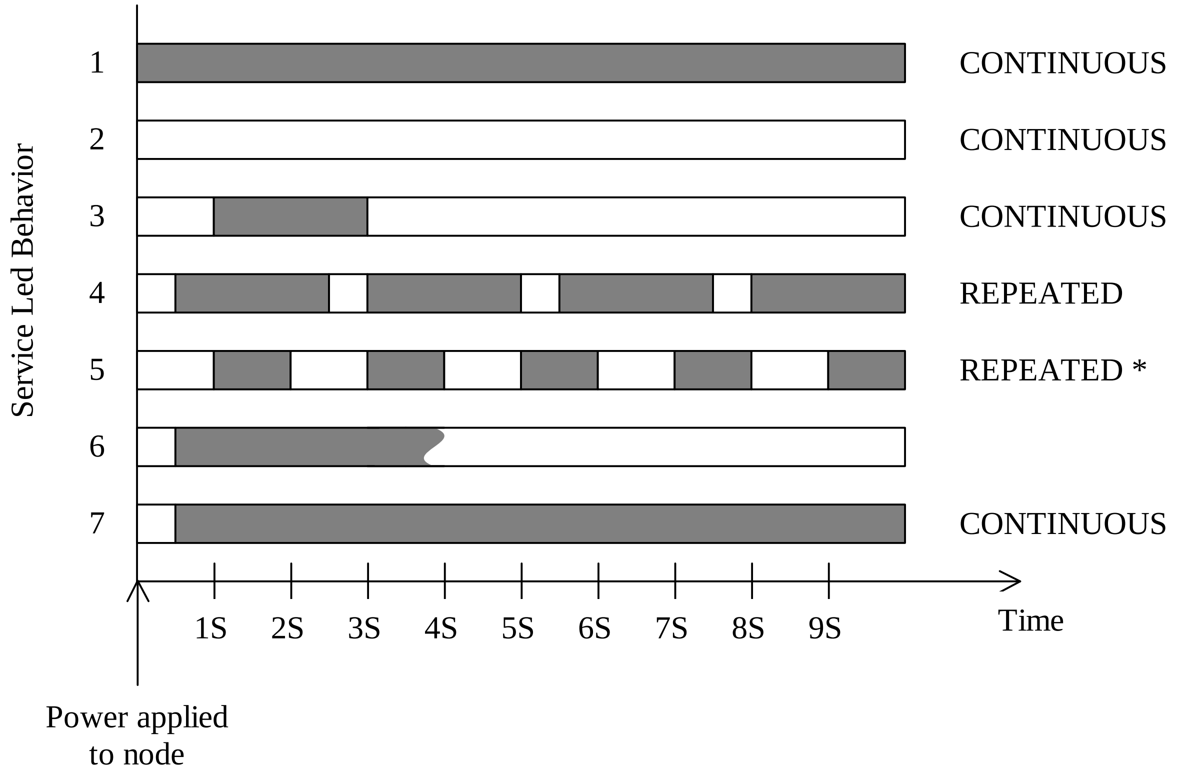
- Hardware
    - 3 stack-based pipelined CPUs
    - Unique 48-bit Neuron ID
    - Programmable Transceiver Interface
    - EEPROM, ROM, and RAM memories
    - 11 I/O pin interface, Two 16 bit timer/counters
    - 5 pin communications interface
- Software
    - LonTalk Protocol
    - Event driven Operating System
    - Application run-time libraries
    - 34 I/O devices controllers

Appendix D – The LonBuilder Navigator Command Chart



Appendix E – LED Behaviors of the Flex I/O

Part I: Possible Service Led Behaviors



☐ =ON ▨ =OFF

* Does not scale with the Neuron Chip clock

Part II: Explanation of the Service LED Behavior

Behavior	Context	Likely Explanation
1	Power-up of a Neuron 3120xx Chip-based node with any Prom	Bad node hardware
2	Power-up of a Neuron 3120xx Chip-based node with any Prom	Bad node hardware
3	Power-up/Reset	Node is application less or the board is jammed
4	Anytime	Watch dog timer resets occuring. Possible corrupt EEPROM.
5	Anytime	Node is unconfigured but has an application. Proceed with loading the node.
6	Using EEBLANK	Cleaning process in progress
7	Anytime	Node configured and running normally

Note: more information is available in the LonWorks Engineering Bulletin: *LonWorks custom Node Development*, January 1995.

Appendix F – Program Directory

The directory we created is called **c:\lb\auv\network**.
All the files we ever used or created are saved under the directory **c:\lb\auv\old_net**
Hereafter is listed the content of **c:\lb\auv\network**:

fancoil.xif is used as an interface file to run on the Flex I/O board called ***iec_fx1*** when *motors.nc* is loaded on one emulator.

pwm.nc creates a PWM output on the LED when it is loaded in an emulator. This program allows the user to physically see the pwm by the means of a clock set with a 1 second period.

fin_emu.nc and ***fin_flex.nc*** have to be used together in order to generate a PWM output on the Flex I/O board. The jumper (JP10) has to be set to that mode. The PWM output will be located on the open_drain_1 of the board.

All the other files and directories (except ***project***) have been created by the LonBuilder itself during normal utilization of the software.