

THERMOCLINE TRACKING USING AN  
UPGRADED OCEAN EXPLORER  
AUTONOMOUS UNDERWATER VEHICLE

MATHIEU CLABON



**THERMOCLINE TRACKING  
USING AN UPGRADED OCEAN EXPLORER  
AUTONOMOUS UNDERWATER VEHICLE**

by

Mathieu Clabon

A Thesis Submitted to the Faculty of  
The College of Engineering  
in Partial Fulfillment of the Requirements for the Degree of  
Master of Science

Florida Atlantic University

Boca Raton, Florida

August 2003

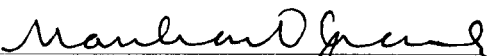
# THERMOCLINE TRACKING USING AN UPGRADED OCEAN EXPLORER AUTONOMOUS UNDERWATER VEHICLE

by

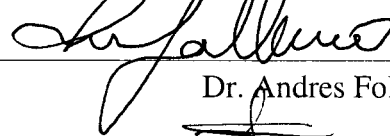
Mathieu Clabon

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Manhar R. Dhanak, Department of Ocean Engineering, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the College of Engineering and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

## SUPERVISORY COMMITTEE



Thesis Advisor, Dr. Manhar Dhanak



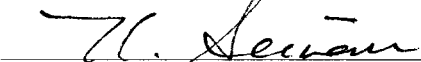
Dr. Andres Folleco



Dr. Edgar An



Chairperson, Department of Ocean Engineering



Dean, College of Engineering



Division of Research and Graduate Studies

7/18/03

Date

## **ACKNOWLEDGMENTS**

I would like to take this opportunity to express my profound acknowledgments to my thesis committee, Dr. Manhar Dhanak, Dr. Edgar An and Dr. Andres Folleco.

I also would like to extend my most sincere gratitude to all the people who helped me conduct the research that led to this thesis. Among all these people, I would like to particularly thank Abby Chronister, Gabriel Grenon and Joe Lambiotte, from the Advance Marine System Laboratory, as well as John Kielbasa, from the Electronics Laboratory, to name a few. It has always been a pleasure to work with them, and without their valuable assistance and advices, I wouldn't have been able to achieve such goals.

The work was supported by the Office of Naval Research under grant number N000149615023 [program managers Drs Thomas Curtin and Thomas Swean].



## **ABSTRACT**

Author: Mathieu Clabon

Title: Thermocline Tracking using an Upgraded Ocean Explorer Autonomous Underwater Vehicle

Institution: Florida Atlantic University

Thesis Advisor: Dr. Manhar R. Dhanak

Degree: Master of Science

Year: 2003

This thesis addresses the problem of tracking a thermocline – a layer of water showing an intense vertical temperature gradient – with an Autonomous Underwater Vehicle (AUV). One of Florida Atlantic University's Ocean Explorer (OEX) AUV has been upgraded, as part of the work described here, by integration of a standard and convenient software interface, and used in several thermocline survey experiments aimed at gathering oceanographic data relevant to thermoclines. A tool that simulates the longitudinal motion of the OEX through a water slice, whose temperature map is read using a simulated temperature and depth sensor, has been developed. Using this tool and information from at-sea experiments, several control methods for the OEX to track a thermocline were analyzed. In particular, two different algorithms were implemented and tested by simulation. Overall, two control algorithms have been validated, and it will soon be possible to provide the AUV with a thermocline tracking capability.

# Table of Contents

<b>LIST OF ILLUSTRATIONS.....</b>	<b>XII</b>
<b>LIST OF TABLES.....</b>	<b>XVI</b>
<b>I. INTRODUCTION.....</b>	<b>1</b>
<b>I.1. Motivation for the Thesis Work.....</b>	<b>1</b>
<b>I.2. Statement of the Problem and Summary of the Thesis Work.....</b>	<b>4</b>
<b>I.3. Structure of the Document.....</b>	<b>5</b>
<b>II. BACKGROUND.....</b>	<b>6</b>
<b>II.1. Thermocline Mapping.....</b>	<b>6</b>
<b>II.1.1. Thermocline Definition.....</b>	<b>6</b>
II.1.1.1. Permanent Thermocline.....	8
II.1.1.2. Seasonal Thermocline.....	8
II.1.1.3. Diurnal Thermocline.....	9
II.1.1.4. Realistic Temperature Profile.....	10
<b>II.1.2. Thermocline Characterization.....</b>	<b>11</b>
<b>II.1.3. Temperature Profiles and Thermocline Mapping.....</b>	<b>13</b>
<b>II.1.4. Thermocline Tracking.....</b>	<b>15</b>
<b>II.2. Feature-Based Navigation and Feature Tracking.....</b>	<b>18</b>
<b>II.2.1. The Concept of Feature-Relative Navigation.....</b>	<b>18</b>
<b>II.2.2. Advantages.....</b>	<b>19</b>
<b>II.2.3. Difficulties.....</b>	<b>20</b>



<b>II.2.4. Techniques Involved.....</b>	<b>21</b>
II.2.4.1. Feature Identification.....	22
II.2.4.2. Feature Finding.....	23
II.2.4.3. Intelligent Control for Feature Tracking.....	24
<b>II.3. The Ocean Explorer.....</b>	<b>27</b>
<b>II.3.1. Existing Advanced Marine Systems Laboratory Vehicles.....</b>	<b>27</b>
II.3.1.1. The OEX-B.....	27
II.3.1.1.1. Features.....	28
II.3.1.1.2. Hardware.....	30
II.3.1.1.3. Software.....	31
II.3.1.2. The OEX-C.....	31
II.3.1.2.1. Improvements.....	31
II.3.1.2.2. Hardware.....	33
II.3.1.2.3. Software.....	33
II.3.1.3. The Morpheus.....	36
<b>II.3.2. The New OEX-D Project.....</b>	<b>37</b>
II.3.2.1. The Common Intelligent Distributed Control System.....	37
II.3.2.1.1. General Idea.....	37
II.3.2.1.2. Components.....	39
II.3.2.1.3. LonWorks Application.....	40
II.3.2.1.4. Application to the OEX.....	41
II.3.2.1.5. Parts.....	43
II.3.2.2. The OEX-B Sensors and Actuators.....	45
II.3.2.2.1. Main Health System.....	45
II.3.2.2.2. Global Positioning System (GPS) and Differential GPS.....	45
II.3.2.2.3. GPS/DGPS Antenna.....	45
II.3.2.2.4. Conductivity Temperature Depth (CTD) Sensor.....	46
II.3.2.2.5. Doppler Velocity Log (DVL).....	46
II.3.2.2.6. Attitude and Heading Reference Sensor (AHRS).....	47
II.3.2.2.7. TopSide Acoustic Modem (TSAM).....	47
II.3.2.2.8. Thruster.....	47
II.3.2.2.9. Fins.....	48
II.3.2.2.10. Dropweight.....	48
II.3.2.2.11. Batteries.....	48
II.3.2.2.12. Control Box.....	49
II.3.2.2.13. Compass.....	49
II.3.2.3. The Morpheus/OEX-C Host Software.....	50
II.3.2.3.1. System Overview.....	50
II.3.2.3.2. Shared Memory.....	52
II.3.2.3.3. LonDaemon Module.....	53
II.3.2.3.4. ProcessData Module.....	54

II.3.2.3.5. <i>Logger</i> .....	55
II.3.2.4. <i>Tools</i> .....	55
II.3.2.4.1. <i>Development Tools</i> .....	56
II.3.2.4.2. <i>Monitoring Tools</i> .....	58
II.3.2.4.3. <i>Processing Tools</i> .....	60
<b>III. UPGRADING THE OCEAN EXPLORER.....</b>	<b>61</b>
<b>III.1. Implementation of a New Computer.....</b>	<b>61</b>
<b>III.2. Operating System and Software Modifications.....</b>	<b>64</b>
<b>III.2.1. Operating System and High-Level Software Installation.....</b>	<b>64</b>
<b>III.2.2. Software Integration.....</b>	<b>65</b>
III.2.2.1. General Method.....	66
III.2.2.2. An Example of Software Integration Tasks: Integration of the GPS.....	67
III.2.2.3. A More Thorough Example.....	76
III.2.2.4. Other Software Modifications.....	81
<b>III.3. Tests.....</b>	<b>83</b>
<b>III.3.1. Pool Test.....</b>	<b>83</b>
III.3.1.1. Mission Planning.....	83
III.3.1.2. Mission Execution.....	84
III.3.1.3. Mission Analysis.....	84
III.3.1.3.1. <i>Mission 1</i> .....	84
III.3.1.3.2. <i>Mission 2</i> .....	86
III.3.1.3.3. <i>Mission 3</i> .....	87
III.3.1.4. Conclusion.....	88
<b>III.3.2. At-Sea Test.....</b>	<b>89</b>
III.3.2.1. Mission Planning.....	89
III.3.2.2. Mission Execution.....	91
III.3.2.3. Mission Analysis.....	91
III.3.2.3.1. <i>Mission 1</i> .....	92
III.3.2.3.2. <i>Mission 2</i> .....	92
III.3.2.3.3. <i>Mission 3</i> .....	95
III.3.2.3.4. <i>Mission 4</i> .....	96
III.3.2.3.5. <i>Mission 5</i> .....	97
III.3.2.4. Conclusions.....	99
<b>III.4. Various Fixes and Conclusion.....</b>	<b>100</b>



<b>IV. MAPPING THE THERMAL STRUCTURE OF A WATER COLUMN.....</b>	<b>101</b>
<b>IV.1. Requirements and Assumptions.....</b>	<b>102</b>
<b>IV.2. Method of Survey.....</b>	<b>103</b>
<b>IV.3. December 2002 Missions.....</b>	<b>105</b>
<b>IV.3.1. Preparing the Missions.....</b>	<b>106</b>
IV.3.1.1. Payload Integration.....	106
<i>IV.3.1.1.1. Turbulence Package Integration.....</i>	<i>106</i>
<i>IV.3.1.1.2. Enabling the ADCP Mode of the DVL.....</i>	<i>107</i>
IV.3.1.2. Sensor Calibration.....	107
IV.3.1.3. Pool Test and Vehicle Trimming.....	108
IV.3.1.4. Writing the Mission.....	110
<b>IV.3.2. Mission Execution.....</b>	<b>112</b>
<b>IV.3.3. Data Analysis and Results.....</b>	<b>117</b>
IV.3.3.1. December 16th Data.....	117
<i>IV.3.3.1.1. Shipboard CTD Data.....</i>	<i>117</i>
<i>IV.3.3.1.2. AUV Mission Data.....</i>	<i>120</i>
<i>IV.3.3.1.3. AUV CTD Data.....</i>	<i>121</i>
<i>IV.3.3.1.4. AUV DVL/ADCP Data.....</i>	<i>124</i>
IV.3.3.2. December 18th Data.....	125
<i>IV.3.3.2.1. Shipboard CTD Data.....</i>	<i>125</i>
<i>IV.3.3.2.2. AUV Mission Data.....</i>	<i>127</i>
<i>IV.3.3.2.3. AUV CTD and DVL/ADCP Data.....</i>	<i>128</i>
<b>IV.3.4. Conclusions.....</b>	<b>128</b>
<b>IV.4. March 2003 Missions.....</b>	<b>129</b>
<b>IV.4.1. Preparing the Missions.....</b>	<b>130</b>
IV.4.1.1. Fixing the Problems Revealed by the Previous Missions.....	130
IV.4.1.2. Sensor Payload Integration.....	131
<i>IV.4.1.2.1. Physical Connection of the Sensors to the Payload Bus....</i>	<i>131</i>
<i>IV.4.1.2.2. Logical Integration of the Sensor Nodes in the Software...</i>	<i>133</i>
IV.4.1.3. Sensor Calibration.....	136
IV.4.1.4. Pool Test and Vehicle Trimming.....	136
IV.4.1.5. Writing the Mission.....	137
<b>IV.4.2. Mission Execution.....</b>	<b>138</b>

<b>IV.4.3. Data Analysis and Results</b> .....	<b>139</b>
IV.4.3.1. Shipboard CTD Data.....	139
IV.4.3.2. AUV Mission Data.....	142
IV.4.3.3. AUV CTD Data.....	145
IV.4.3.4. AUV DVL/ADCP Data.....	148
<b>IV.4.4. Multiple Datasets Comparison</b> .....	<b>151</b>
<b>IV.4.5. Conclusions</b> .....	<b>154</b>
<b>IV.5. Conclusions</b> .....	<b>155</b>
<b>V. THERMOCLINE TRACKING SIMULATION</b> .....	<b>157</b>
<b>V.1. Description of the Problem</b> .....	<b>157</b>
<b>V.1.1. Motivations</b> .....	<b>157</b>
<b>V.1.2. Typical Mission to Simulate</b> .....	<b>158</b>
<b>V.1.3. Simulation Needs</b> .....	<b>160</b>
<b>V.2. Simulator Design and Implementation</b> .....	<b>161</b>
<b>V.2.1. Method</b> .....	<b>162</b>
V.2.1.1. AUV Motion Simulation.....	162
V.2.1.1.1. <i>Frames of Reference</i> .....	162
V.2.1.1.2. <i>Equations of Motion</i> .....	164
V.2.1.1.3. <i>Solving the Equations of Motion</i> .....	168
V.2.1.2. Water Slice Simulation.....	170
V.2.1.2.1. <i>Generation of a Temperature Map</i> .....	171
V.2.1.2.2. <i>Use of the Map by the CTD Simulator</i> .....	172
V.2.1.3. Other Main Requirements.....	174
V.2.1.3.1. <i>Logger</i> .....	174
V.2.1.3.2. <i>Fins Controller</i> .....	175
V.2.1.3.3. <i>Scheduler</i> .....	176
<b>V.2.2. Implementation</b> .....	<b>177</b>
V.2.2.1. Required Modules.....	177
V.2.2.2. Module Interactions.....	179
V.2.2.3. Simulation Interface.....	179
V.2.2.3.1. <i>Simulation Parameters</i> .....	180
V.2.2.3.2. <i>Simulation Output</i> .....	181
V.2.2.4. Thermocline Tracking Interface.....	181



V.2.2.5. Implementation of Each Module.....	181
V.2.2.5.1. <i>Simulation Initialization</i> .....	181
V.2.2.5.2. <i>Simulation Termination</i> .....	182
V.2.2.5.3. <i>Scheduler</i> .....	182
V.2.2.5.4. <i>Vehicle Motion Simulation</i> .....	183
V.2.2.5.5. <i>CTD Simulation</i> .....	185
V.2.2.5.6. <i>Fins Controller Simulation</i> .....	186
V.2.2.5.7. <i>Logger Simulation</i> .....	187
V.2.2.5.8. <i>Summary Generator</i> .....	188
<b>V.2.3. Graphical User Interface.....</b>	<b>189</b>
V.2.3.1. Overview.....	189
V.2.3.2. Implementation.....	190
V.2.3.2.1. <i>Main Window</i> .....	190
V.2.3.2.2. <i>Timing Parameters Edition</i> .....	190
V.2.3.2.3. <i>Input and Output Files Specification</i> .....	191
V.2.3.2.4. <i>Tracking Controller Specification</i> .....	192
V.2.3.2.5. <i>CTD Simulation Configuration</i> .....	193
V.2.3.2.6. <i>Mission Parameters</i> .....	193
V.2.3.2.7. <i>Running the Simulation</i> .....	194
V.2.3.2.8. <i>Displaying Results</i> .....	194
V.2.3.2.9. <i>Terminating the Simulation</i> .....	194
V.2.3.3. Graphical Layout.....	195
<b>V.2.4. Test of the Simulation Platform.....</b>	<b>195</b>
V.2.4.1. Description of the Test Simulation.....	196
V.2.4.2. Simulation Configuration.....	196
V.2.4.3. Results.....	198
V.2.4.4. Conclusion.....	201
<b>V.3. Thermocline Tracking Controller Simulation.....</b>	<b>201</b>
<b>V.3.1. Method.....</b>	<b>201</b>
<b>V.3.2. Thermocline Tracking Depth Controller.....</b>	<b>204</b>
V.3.2.1. Approach.....	204
V.3.2.2. Design.....	205
V.3.2.3. Implementation.....	208
V.3.2.4. Testing.....	210
V.3.2.5. Conclusions.....	221
<b>V.3.3. Thermocline Tracking Sternplane Controller.....</b>	<b>221</b>
V.3.3.1. Approach.....	221
V.3.3.2. Design.....	222

V.3.3.3. Implementations.....	226
V.3.3.4. Testing.....	229
V.3.3.5. Conclusions.....	234
<b>V.4. Comparison of both Methods and Conclusions.....</b>	<b>235</b>
<b>VI. CONCLUSIONS AND FUTURE WORK.....</b>	<b>237</b>
<b>VI.1. Summary of the Thesis Work.....</b>	<b>237</b>
<b>VI.2. Main Conclusions and Future Work.....</b>	<b>239</b>
<b>APPENDIX.....</b>	<b>242</b>
<b>Summary File Generated by the Simulation Tool.....</b>	<b>242</b>
<b>BIBLIOGRAPHY.....</b>	<b>244</b>

## List of Illustrations

Figure 1: A Three-Layer Ocean.....	7
Figure 2: Variations of the Temperature Profile with Latitude.....	8
Figure 3: Two Representations of the Seasonal Thermocline.....	9
Figure 4: Example of Diurnal Temperature Profile Variation.....	10
Figure 5: Combination of Permanent and Seasonal Thermoclines.....	10
Figure 6: Example of Idealized Temperature Profile and Temperature Gradient Profile..	12
Figure 7: Data from March 14th, 2002 Cross Shelf Experiment.....	16
Figure 8: The Ocean Explorer (OEX).....	28
Figure 9: Ocean Explorer System Overview.....	29
Figure 10: Overview of the Control Software Architecture.....	35
Figure 11: The Morpheus Ultramodular AUV.....	36
Figure 12: Centralized vs. Distributed Control Architecture.....	38
Figure 13: A Node Attached to the Distributed Control Network.....	40
Figure 14: The OEX-B Intelligent Distributed Control Network.....	42
Figure 15: Software Overview: Main Functional Groups.....	50
Figure 16: Shmem.in Content Example.....	52
Figure 17: LonDaemonNv Table Content Example.....	54
Figure 18: Lgr.in Content Example.....	55
Figure 19: Integration of the New Main Computer Stack.....	63
Figure 20: Simplified GPS Data Flow Chart.....	67
Figure 21: GPS Variables and Logical Connections Synoptic Diagram.....	69
Figure 22: GPS Data Processing and Conversion Algorithm Summary.....	74
Figure 23: Multiple Batteries with one Single Application.....	78
Figure 24: Batteries Energy Gauge Reset from MissionCheck.....	80
Figure 25: OEX-D Distributed Control Network.....	82
Figure 26: Thruster Test Results.....	87

Figure 27: Dead-Reckoning Navigation Compared to GPS Position.....	93
Figure 28: AUV State Summary.....	94
Figure 29: AUV Depth during Depth-Controlled Mission.....	95
Figure 30: AUV Altitude, Depth and Bathymetry during Altitude-Controlled Mission. .	96
Figure 31: Speed Test Results (Rpm and Speed Data).....	97
Figure 32: Vertical Lawn-Mower Pattern.....	105
Figure 33: CTD Noise Characterization from Pool Test Data.....	109
Figure 34: ADCP Data from Pool Test.....	110
Figure 35: Macro Instruction "Vertical Leg" .....	111
Figure 36: Whole Mission Plan Summary.....	112
Figure 37: CTD Casts at Edges of Mission Location before Launching the Vehicle.....	113
Figure 38: Mission Summary (Navigation).....	115
Figure 39: CTD Casts at Edges of Mission Location before Launching the Vehicle.....	116
Figure 40: Mission Plan for December 18th Mission.....	116
Figure 41: Temperature and Conductivity Profiles (December 16th).....	118
Figure 42: Temperature and Conductivity Vertical Gradient Profiles (December 16th).	118
Figure 43: AUV Vertical Path from Dead-Reckoning Navigation and CTD Depth.....	120
Figure 44: AUV Horizontal Path from Dead-Reckoning Navigation .....	121
Figure 45: Raw Depth and Temperature Data from AUV CTD Sensor.....	122
Figure 46: CTD Data Noise and Vertical Temperature Profile.....	123
Figure 47: Raw Water Current Velocity in the Vehicle Body-Fixed Frame.....	124
Figure 48: Temperature and Conductivity Profiles (December 18th).....	125
Figure 49: Temperature and Conductivity Vertical Gradient Profiles (December 18th).	126
Figure 50: AUV Trajectory in both the Vertical and Horizontal Planes.....	127
Figure 51: Upward Looking ADCP and SBE CTD Payload Interface Module.....	133
Figure 52: CTD Casts at Edges of Mission Location before Launching the Vehicle.....	138
Figure 53: Temperature and Conductivity Profiles (March 19th).....	140
Figure 54: Temperature and Conductivity Vertical Gradient Profiles (March 19th).....	140
Figure 55: AUV Vertical Path from Dead-Reckoning Navigation and CTD Depth.....	142
Figure 56: AUV Horizontal Path from Dead-Reckoning Navigation.....	143
Figure 57: AUV Horizontal Path, Corrected for Current-Induced Drift.....	144
Figure 58: Raw Depth and Temperature Data from AUV CTD Sensor.....	145
Figure 59: Vertical Temperature Profile of the Water Column from AUV CTD Data...	146

Figure 60: Temperature Map from AUV CTD Data.....	147
Figure 61: Raw ADCP Current Magnitude Profile as seen from the Vehicle.....	148
Figure 62: ADCP Velocity Error and Corresponding Validation Mask.....	149
Figure 63: Absolute Current Velocity Magnitude.....	150
Figure 64: Temperature Profiles Comparison.....	151
Figure 65: Temperature Maps Comparison.....	153
Figure 66: Sketch of a Typical Thermocline Tracking Mission.....	160
Figure 67: Simulator Structure Overview.....	161
Figure 68: 3-D Geographical Frame.....	163
Figure 69: Body-Fixed Frame.....	163
Figure 70: Local-Level ( $x_L$ , $z_L$ ) and Body-Fixed ( $x_B$ , $z_B$ ) Frames for Simulation.....	164
Figure 71: Sparse Temperature Map.....	172
Figure 72: Simulation Modules Interactions.....	179
Figure 73: Timing Parameters Section of the GUI.....	191
Figure 74: I/O Parameters Section of the GUI.....	192
Figure 75: Tracking Controller Section of the GUI.....	192
Figure 76: CTD Configuration Section of the GUI.....	193
Figure 77: Mission Configuration Section of the GUI.....	193
Figure 78: Graphical Layout of the whole GUI.....	195
Figure 79: Test Simulation Inputs: Temperature Profile and Depth Command Pattern..	198
Figure 80: Trajectory of the Vehicle.....	199
Figure 81: Depth Comparison (Desired, Measured, Actual and Maximum).....	199
Figure 82: Simulated Noise on Depth Measurements.....	200
Figure 83: Simulated Noise on Temperature Measurements.....	200
Figure 84: Measured Depth Error and Sternplane Angle.....	201
Figure 85: Local Temperature Gradient Measurement.....	203
Figure 86: Simple Temperature Map.....	211
Figure 87: Complex Temperature Map showing a Diving Thermocline.....	212
Figure 88: Simulation Results: Vehicle Trajectory over the Temperature Map.....	214
Figure 89: Depth Variables (Desired, Measured, Real and Mean).....	215
Figure 90: Temperature Profile Measured by the Vehicle.....	216
Figure 91: Simulations Results for Noisy CTD Measurements (Low and High Noise)..	217
Figure 92: Temperature Profile Measured in Case of High Noise.....	218



Figure 93: Real Temperature and Temperature Gradient Profiles used for Simulation. .	219
Figure 94: Simulation Results: Vehicle Trajectory over the Temperature Map.....	220
Figure 95: Simplified Reasoning: Four Cases.....	223
Figure 96: Simple Membership Functions.....	227
Figure 97: Control Surface.....	229
Figure 98: Simulation Results: Vehicle Trajectory over the Temperature Map.....	231
Figure 99: Depth, Pitch and Sternplane Angle vs. Time.....	232
Figure 100: Temperature Profile Measured by the Vehicle.....	233

## List of Tables

Table 1: IDCS Channel Characteristics Summary.....	41
Table 2: Main Computer Components.....	62
Table 3: Network Variables Interface for the GPS Node.....	70
Table 4: Binding Information Table for GPS Network Variables.....	71
Table 5: LonDaemonNv Table Section for GPS Network Variables.....	71
Table 6: Shared Memory Variables Declaration for raw GPS Variables.....	72
Table 7: Shared Memory Variables Declaration for Converted GPS Variables.....	75
Table 8: Logger Input File Section for GPS Variables.....	75
Table 9: Vehicle Health Data Summary for Mission 1 on 10/14/02.....	85
Table 10: Vehicle State Data Summary for Mission 1 on 10/14/02.....	86
Table 11: CTD Noise Characterization from Pool Test Data.....	109
Table 12: Payload CTD Network Interface.....	135
Table 13: Simulation Modules Description.....	178
Table 14: Simulation Parameters.....	180
Table 15: Simulation Initialization Implementation Summary.....	182
Table 16: Simulation Termination Implementation Summary.....	182
Table 17: Scheduler Implementation Summary.....	183
Table 18: Motion Simulation Implementation Summary.....	184
Table 19: CTD Simulation Implementation Summary.....	186
Table 20: Fins Controller Implementation Summary.....	187
Table 21: Logger Implementation Summary.....	187
Table 22: Message Writing Implementation Summary.....	189
Table 23: Input Parameters for Test Simulation.....	197
Table 24: Tracking Controller Main Module Implementation Summary.....	209
Table 25: Measurements Processing Implementation Summary.....	209
Table 26: Oscillations Generation Implementation Summary.....	210

Table 27: Controller Parameters for Test Simulation.....	212
Table 28: Simulation Parameters for Controller Test.....	213
Table 29: Simulation Parameters for Controller Test in Low Noise Environment.....	216
Table 30: Simulation Parameters for Controller Test in High Noise Environment.....	216
Table 31: Simulation Parameters for Controller Test with Real Temperature Data.....	219
Table 32: Controller Parameters for Controller Test with Real Temperature Data.....	220
Table 33: Characteristics of the Fuzzy Engine.....	225
Table 34: Inference Rules.....	228
Table 35: Controller Parameter for Test Simulation.....	230
Table 36: Simulation Parameters for Controller Test.....	230

# I. Introduction

## *1.1. Motivation for the Thesis Work*

Understanding the world we live in – especially in our case, the ocean – requires some extended studies, including detailed experiments and measurements. Although some of these can be carried out using remote sensing techniques, they need to be complemented with in-situ measurements. Instrumented unmanned mobile platforms are very useful for undersea measurements since they allow for fast 3-Dimensional sampling. Especially, Autonomous Underwater Vehicles (AUVs) provide low-cost platforms for these types of missions.

Because of both the spatial and temporal variability of oceans, in order to obtain significant results, the optimal sampling mission would need to be performed over a large area, instantaneously, with infinite resolution. Obviously, this is not feasible. Therefore, a compromise has to be decided between accuracy, resolution, extent of the area, duration of the mission, energy consumption and cost of the mission.

A possible choice of mission path is in the form of a "lawn mower pattern" that consists of a grid of parallel straight lines. The characteristics of this pattern, such as total area, leg separation, cruise speed, are determined based on prior-knowledge - or assumptions - about the feature to sample. These types of patterns have proved to be very well suited to certain kinds of problems. Indeed, they have the advantages of being exhaustive and of

allowing the vehicle to travel in an energetically efficient way.

However this pattern is efficient – in terms of sampling performances – only when enough prior knowledge is available about the feature to be mapped. Moreover, this kind of pattern is not always suited to the feature of interest, and may provide limited information. Finally, when choosing such a pattern, one is already committed to traveling a long distance. In such cases, the efficiency of the survey may be dramatically reduced. The worst case is that of a dynamic feature, about which only little knowledge is available, and whose very essence makes it difficult – or impossible – to be surveyed along a regular pattern. Unfortunately, many features one would benefit to analyze often fall in that category, as illustrated by the numerous attempts to track jellyfishes or map chemical plumes, to name a few.

A possible way to improve the efficiency of these surveys would require the AUV to determine by itself the area of interest, by reacting interactively to sensor measurements. These techniques, referred to as "feature-relative navigation", would allow the AUV to navigate no longer along a predefined pattern, but along a pattern that is constantly updated depending on what the AUV senses about the feature to sample. It is commonly admitted that the implementation of such techniques would offer a possibility to dramatically increase the efficiency of a survey mission, and use AUVs at the best of their capabilities.

Until recently, most of the work in the area of feature-relative navigation has been confined to tracking of static man-made features, such as the inspection of underwater cables or pipelines. But now, the application of these techniques to scientific sampling is being successfully addressed, suggesting the possibility to extend this to various



applications.

In the case we are considering here, the feature to track is a thermocline, that is a layer of water with a more intensive vertical gradient in temperature than that found in the layers above or below. Characteristics of thermoclines, such as location, thickness, depth, or intensity of the temperature gradient, vary with numerous parameters, and in turn, affect several phenomena, such as sound propagation or biological activity to name a few. For this reason, it is of some interest to measure the spatial variability of the thermocline in a given area.

The localization of a thermocline is usually addressed with multiple grid-arrayed Conductivity-Temperature-Depth (CTD) sensor casts, or with remote sensing techniques such as satellite or radar measurements. However, in-situ measurements are expected to give more detailed and accurate results. CTD casts, conducted at different points, require both significant prior knowledge about the thermocline variability - in order to design the survey grid - and significant post-processing to interpolate the point measurements in order to reconstruct the whole field.

Therefore, it is highly desirable to have an autonomous platform follow a thermocline layer. This would allow for more efficient high sampling rate measurements, which, incorporated with the trajectory of the platform, makes possible to develop a precise map of the thermocline. Moreover, this would also allow the measurement of various parameters along the thermocline, which is not possible otherwise.

These goals can be achieved with an AUV capable of performing survey missions safely and efficiently. The AUV has to be upgraded so that it has the ability to follow a thermocline. To do so, a thermocline controller needs to be designed. To facilitate an easy

implementation of such a controller, reduce testing and validation time, while maintaining a satisfactory level of efficiency and safety, the best solution involves spending much time in developing and performing various simulations.

The department of Ocean Engineering at Florida Atlantic University (FAU) currently has two series of survey-class AUVs, the Morpheus and the Ocean Explorer. The AUV platform considered here is one of FAU's Ocean Explorers.

## ***1.2. Statement of the Problem and Summary of the Thesis Work***

The aim of the work described in this thesis is to address the problem of tracking a thermocline with an Ocean Explorer autonomous underwater vehicle. The first task is to upgrade one of the OEX vehicles high-level software so as to offer a more convenient programming interface. Specifically, a new vehicle, the OEX-D, is built from parts taken from the existing OEX-B, OEX-C and Morpheus. Once the vehicle is fully tested and able to perform survey missions efficiently, it is used to gather oceanographic data pertinent to the thermocline. Particularly, temperature data is collected to construct a temperature map of a given water slice. Based on this information about the variability of the temperature profile, as well as that obtained from other experiments, a thermocline tracking controller is simulated for that vehicle. In order to efficiently test the controller simulator, a convenient and robust, yet simple, simulation tool is designed. This includes a basic vehicle dynamics simulation, a realistic CTD simulator that provides meaningful temperature and depth inputs to the tracking controller, as well as some other functionalities such as the possibility to record the time history of selected variables. This

constitutes a systematic simulation platform on which any designed tracking controller can be exhaustively tested.

This thesis describes the work that has been conducted in order to build the new OEX-D AUV from existing parts taken from other vehicles, to use this vehicle to perform temperature mapping missions, and to design a simulation tool on which a few thermocline tracking controllers are tested.

### ***1.3. Structure of the Document***

The structure of this document is as follows: first, we summarize the theoretical background required for a complete understanding of the motivations of the thesis and the work that has been conducted. This includes a brief description of thermoclines, a literature review of the topic of feature tracking, or more generally feature-relative navigation, and finally a description of the most important parts of the existing AUVs that were used in building the new OEX-D. We then discuss the work we carried out, in three main topics: the building and testing of the OEX-D, its use in missions intended to gather data about the temperature structure of the water column, and the design and implementation of a simulation tool and several thermocline tracking controllers. Finally, we draw some conclusions and guidelines relative to possible improvements of the OEX-D, and the effective implementation of a thermocline tracking capability in that vehicle.

## **II. Background**

This chapter summarizes the necessary theoretical background, along with some motivations, for the thesis work. Three topics are studied: first, existing knowledge related to thermocline and its mapping, then the concept of feature relative navigation, and finally, the essential components of the Ocean Explorer AUV.

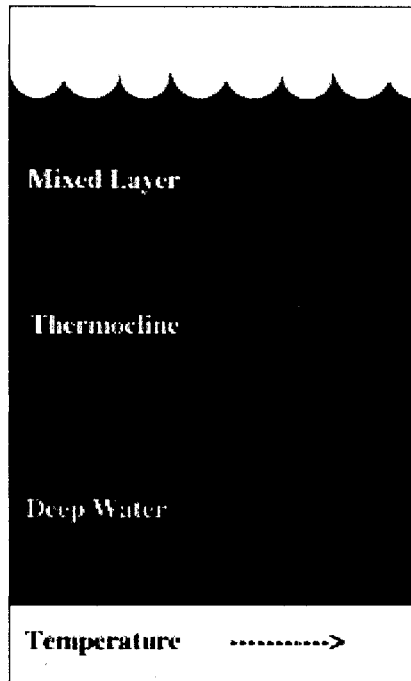
### ***II.1. Thermocline Mapping***

Seawater temperature has long been the most commonly studied oceanographic variable because it is easily measured and because of its direct influence on the chemical, biological and physical properties of seawater [1]. Numerous applications of this property have made the temperature structure of the ocean a major oceanographic study.

#### **II.1.1. Thermocline Definition**

According to the layered ocean model, below the free surface, the water can usually be divided into three zones (Figure 1) in terms of its temperature structure [2]. There is an upper zone with temperature similar to that at the surface, a zone below this in which the temperature decreases rapidly with depth, and a deep layer in which the temperature changes slowly. The upper layer is either called the mixed layer, because of its homogeneity, Ekman layer or epilimnion. The last term is mostly used in describing lake waters. The thermocline is the layer of water with a more intensive vertical gradient in

temperature than that found in the layers above or below it [1]. The deepest layer is commonly referred to as deep (isothermal) water, or, especially for lake water, hypolimnion. The term “thermocline” was originally proposed for a lacustrine environment, by Birge in 1897.



*Figure 1: A Three-Layer Ocean*

These layers are of varying depth, thickness, areal extent and permanence [3]. Thermoclines can be affected by practically any physical process occurring in the oceans and by meteorological processes above the surface. Thermoclines, in turn, affect various properties such as sound propagation, water transparency and biological population distribution.

Three kinds of thermoclines are commonly defined [1], which are described hereafter.



### II.1.1.1. Permanent Thermocline

This is generally the deepest thermocline, with its upper boundary residing between 100-700m below the surface. It is said to be permanent since changes in its structure do not seem to be related to seasonal or shorter periods. It tends to be symmetrical about the equator, being shallow, moderately thick and quite strong near the equator, becoming thicker, deeper and less intense in the mid-latitudes, and ending at the surface in an intense gradient in the vicinity of  $\pm 50-60^\circ$  latitude (Figure 2) [2].

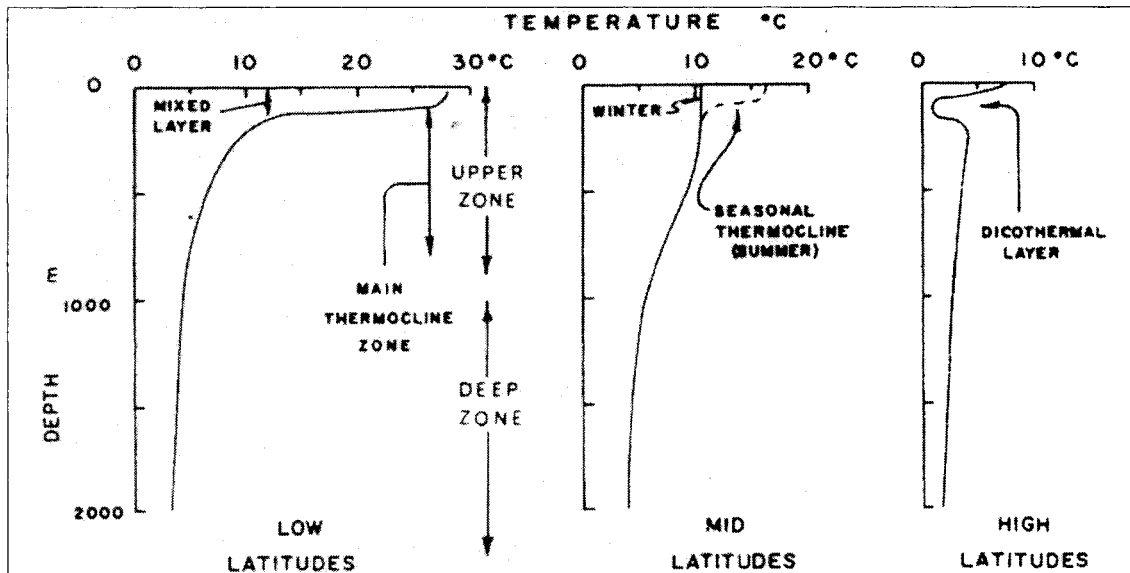


Figure 2: Variations of the Temperature Profile with Latitude

### II.1.1.2. Seasonal Thermocline

It is so named because of its variation with the seasons. It develops in spring, becomes stronger with the summer and disappears during fall and winter. It is found nearer the surface than the permanent thermocline, and in those areas where the seasons show noticeable differences. Mid-latitude oceans in particular, demonstrate a well-developed seasonal thermocline.

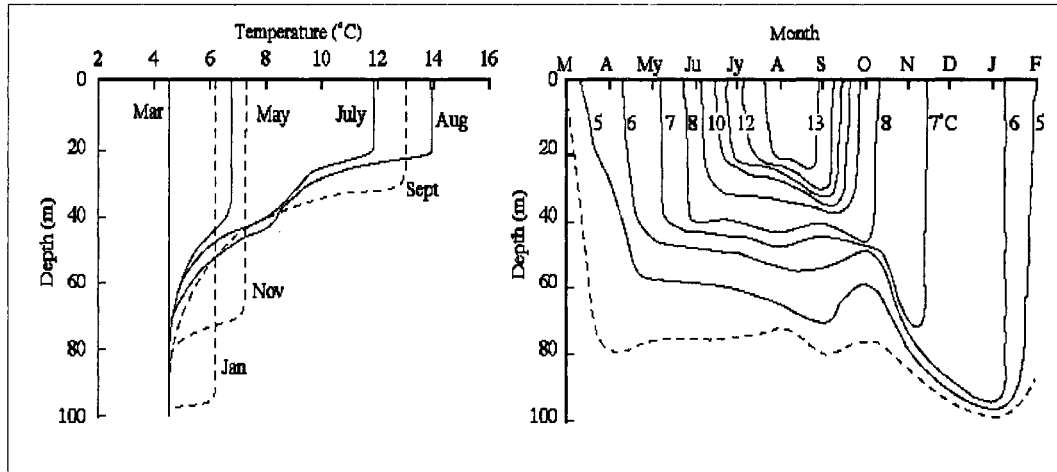


Figure 3: Two Representations of the Seasonal Thermocline

In Figure 3, two different representations of the thermocline are shown. On the left, the temperature profile is shown, with the thermocline corresponding to non-constant temperature ranges of the profile. On the right, the thermocline is located where the isothermal curves get closer to each other. In both figures, it appears clearly that the location, thickness and strength of the thermocline vary with the seasons [4].

In spring, the water is nearly isothermal. As the weather warms, the water surface receives more heat than it loses to the air and becomes warmer. The wind then acts to cause mixing of this surface water to a slightly greater depth. As more heat is received and the wind continues to mix it deeper, the thermocline layer will be found farther from the surface. The maximum depth and intensity of the thermocline depend on the strength and the duration of the wind and the amount of radiation received.

### II.1.1.3. Diurnal Thermocline

Thermocline can develop on smaller scales. The diurnal thermocline (Figure 4) is perhaps the one that occurs on the smallest detectable time scale. A thermocline can form very

near the surface during the day and disappears at night. The intensity can be affected by such things as the degree of cloudiness and the temperature difference between sea and air.

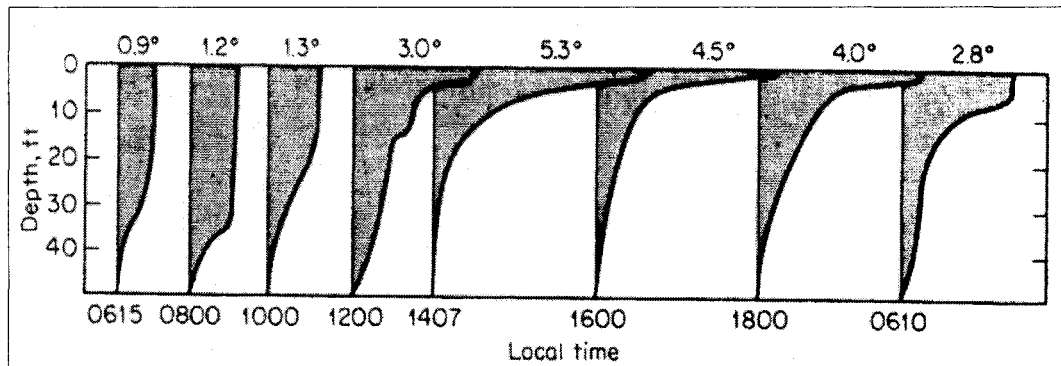


Figure 4: Example of Diurnal Temperature Profile Variation

#### II.1.1.4. Realistic Temperature Profile

These different thermoclines combine together to form the whole temperature profile that is obviously more complex and smoother than what is described in the previous definitions. An idealized example is given in Figure 5.

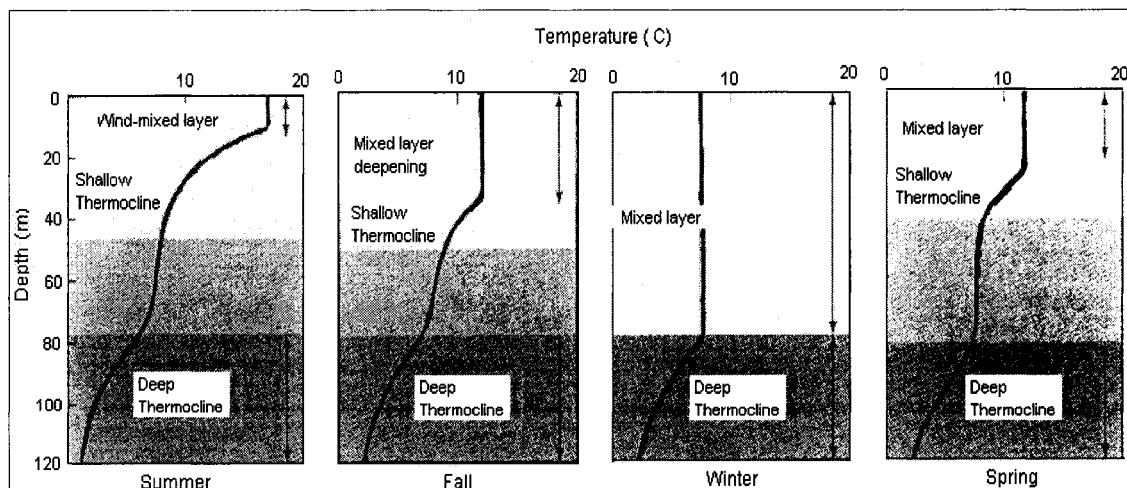


Figure 5: Combination of Permanent and Seasonal Thermoclines

## **II.1.2. Thermocline Characterization**

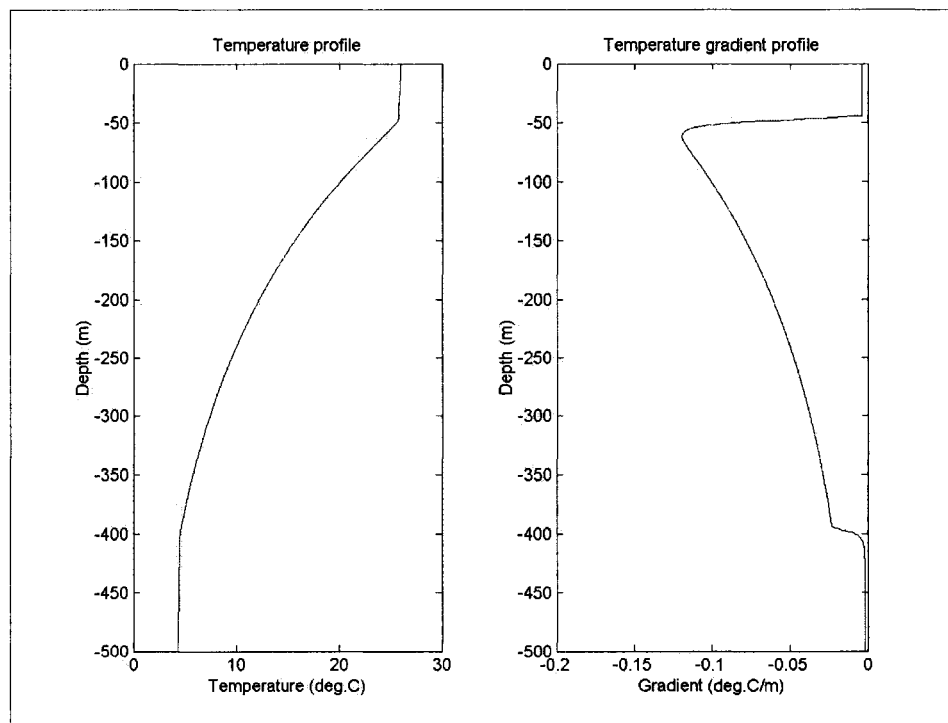
Then the question that arises is how to locate a thermocline in the ocean. To begin to answer this, we can look at some numbers about the real temperature distribution. The order of magnitude of the variation in temperature with depth in the deep thermocline (low troposphere) is around  $0.05^{\circ}\text{C}/\text{m}$  [1]. For comparison, in the cold deep layer (stratosphere), the vertical gradient drops to  $0.004^{\circ}\text{C}/\text{m}$  at 1000m,  $0.001^{\circ}\text{C}/\text{m}$  at 2000m and  $0.0005^{\circ}\text{C}/\text{m}$  below 3000m. An example of the order of magnitude of the temperature gradient in the mixed layer can be obtained from the data of our October 2001 Ocean Current Profiling Experiment off Dania Beach. Based on 25 meters of the mid-section of the mixed layer, a mean temperature gradient of around  $0.006^{\circ}\text{C}/\text{m}$  could be measured. Moreover, the temperature profile variation over the day appeared clearly. This  $0.05^{\circ}\text{C}/\text{m}$  gradient in the thermocline is quite weak, but still higher than in the other layers. Moreover, stronger temperature gradients have been found in shallower thermoclines. A good example of a strong thermocline is reported in the experiment of the Littoral Ocean Observing and Predictive System (LOOPS) conducted in Massachusetts Bay in September 1998 [5]. The experiment consisted of shipboard profiling systems and AUVs deployed along sawtooth patterns. During this experiment, it was found that the water column, with depths of 20-55 meters, could approximately be divided into three layers:

- 1) A 10-20 m thick surface layer with warm and fresh water,
- 2) A 10-30 m thick bottom layer with cold and salty water,
- 3) In between, a thin thermocline layer with sharp vertical gradients.

In that thermocline layer, a vertical gradient of  $8^{\circ}\text{C}$  over less than 5m ( $1.6^{\circ}\text{C}/\text{m}$ ) was

typically observed. A thin chlorophyll maximum layer was often present in this strong thermocline layer as evidenced in the fluorometer data sets. Several horizontal patches of phytoplankton were evident as well.

Obviously, the gradient variation over the different layers is quite smooth. It is particularly interesting to look at the variation of the temperature in the thermocline. Based on Stommel's conceptual model [6], which uses the planetary-geostrophic equations (PGE) [5], assuming that the vertical advection is balanced mainly by vertical diffusion, it can be shown that the temperature in the thermocline layer can be modeled as an exponential of the depth  $z$ ,  $z$  being zero at the top of the layer, negative downward. The temperature gradient is thus larger near the top of the layer, which is therefore the easiest boundary to detect, as appears in Figure 6.



*Figure 6: Example of Idealized Temperature Profile and Temperature Gradient Profile*



Numerous references [1], [2], [5], [6], [7], [8] also show that the location of the thermocline is highly related to the location of the pycnocline (a layer of rapid variation in water density with depth), and possibly a halocline (a layer of rapid variation of water salinity with depth). Specifically, in particular conditions, the top of the thermocline may correspond to a local maximum in the salinity profile. Moreover, some parameters like the oxygen concentration, the chlorophyll density or the water currents vary with the thermocline. These relations among various parameters can give additional ways to locate the thermocline upper boundary.

### **II.1.3. Temperature Profiles and Thermocline Mapping**

Two approaches exist to determine the position of a thermocline: simulation of mathematical models, and direct measurements.

The modeling of the thermocline or more generally of the temperature distribution in the ocean is an important area of research. Scientists are testing the likelihood of different hypothesis and assumptions, and are evaluating the goodness-of-fit of various models. An example of these researches is the World Ocean Circulation Experiment (WOCE) program, related, among others, to what they call “The Thermocline Problem”, which tries to make a connection between dynamical theories and observational programs [9].

Direct measurement approach uses various sensors and techniques. The first and most widely used method to measure the temperature is the use of thermometers. The simple thermometer presents the disadvantage that the readings are corrupted as soon as the thermometer traverses layers of different temperature before reading. For this reason these thermometers were enclosed in thermally isolated bottles. The bottle is filled at the

desired depth, then closed and taken on board. Since the bottle is isolated, the temperature reading is that of the depth where the bottle was opened. An improvement of this technique was introduced with the Deep Sea Reversing Thermometers (DSRTs) [8]. This thermometer is specially constructed so that the act of turning it over at the desired depth locks in the temperature reading. The main drawback of these thermometers is that they are point sensors that give the temperature at a particular depth of a particular location. The reconstruction of a 3-Dimensional temperature map requires a lot of measurements, which is time consuming.

To solve this problem, new sensors were introduced in two main forms: BathyThermograph (BT) and Conductivity-Temperature-Depth sensors (CTD). Mechanical BT was the first instrument developed to rapidly measure temperature changes with depth [8]. This torpedo-shaped device traces a temperature-versus-depth profile on a small slide that can be read directly once the BT is retrieved. The BT has then been replaced by expandable BT (XBT) [3]. In this version, the device sinks at a known constant velocity, and the temperature data is electrically transmitted to the shipboard recorder, while the depth is obtained from the elapsed time. The XBT, as indicated by its name, is not recovered. An Aircraft deployed XBT (AXBT) also exists and uses the same principles [3].

The CTD is an electronic sensor that measures simultaneously the conductivity, temperature and depth. The main advantage of the CTD over thermometers is that, as BT, they measure profiles and not only the temperature at one point. Moreover, where the BT is launched once, along a vertical path, CTDs can be used in different ways. They can be embedded on almost any platform. For this reason, it is the preferred sensor used in

underwater vehicles. The use of such a sensor on AUVs allows to determine temperature map along almost any desired path, which is a great advantage compared to point-measurement sensor since it allows to take into account the spatial variation of the measured parameter without the need for numerous sensors.

Recently, new measurement methods have been introduced such as acoustic tomography, [8] which is based on the observation of acoustic propagation in seawater to determine the medium properties. The main disadvantages of this method are that it uses a complex array of sensors, which is quite expensive and hard to deploy, and it requires a lot of computations.

Finally, other methods try to obtain the depth of the thermocline from the measurements of other quantities such as surface temperature anomaly, that can be obtained by satellite, eddies and internal waves analysis [3].

#### **II.1.4. Thermocline Tracking**

As explained before, according to its definition, the thermocline is related to the vertical variation of the temperature. For that reason, the question may arise as to the possible advantages of using an AUV to track a thermocline. Indeed, this is interesting not necessarily to analyze the thermocline itself, but rather its variation over horizontal distance.

An interesting way to illustrate that idea is to look at some data from the Cross Shelf CTD Experiment. This experiment was intended to estimate the variation of the CTD profile over some horizontal distance across the Florida Current. It consisted of a number of CTD casts at various locations along a line oriented towards the East-West direction.

The experiment was repeated several times with one month interval so as to also estimate the variability with time. The following data (Figure 7), that shows the variability of the thermocline over distance, is taken from the experiment conducted on March 14<sup>th</sup>, 2002.

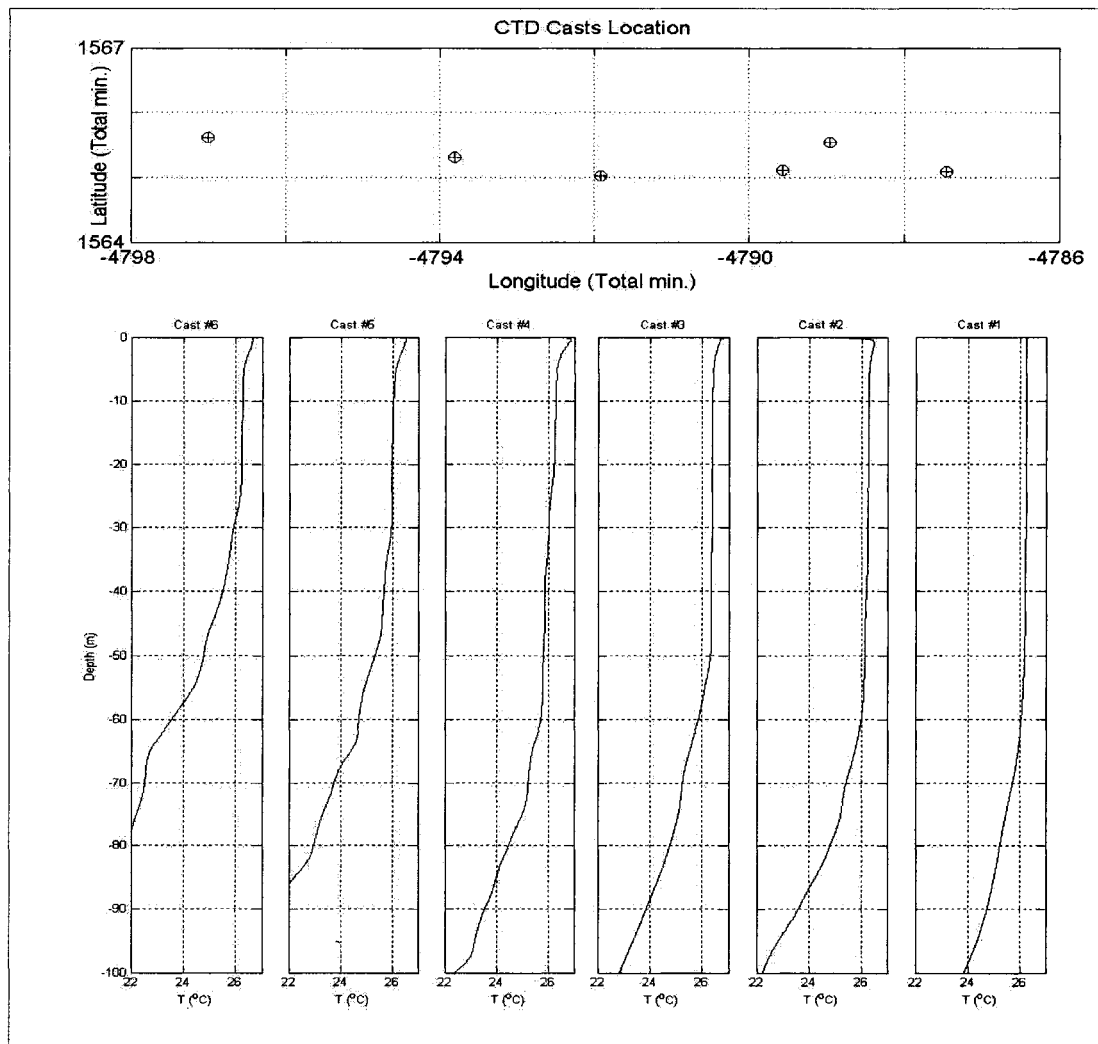


Figure 7: Data from March 14th, 2002 Cross Shelf Experiment

The interesting thing to notice is that each of these six CTD casts took between 30 minutes and 1 hour. It took 5 hours to get the temperature profile at these 6 points. Some other experiments, with closer cast locations took approximately the same time, since

what is time consuming is not to travel between points, but rather to perform the CTD casts. Moreover, once the data are collected, it is necessary to perform a lot of data processing to eliminate the noise in the measurements, especially that due to the motion of the ship and the current that prevents the CTD from being deployed vertically with constant depth rate. Once the measurement are cleaned of noise, it is still necessary to interpolate the data so as to reconstruct the information between the sampling points. The conclusion is that it takes a lot of time to obtain results with a poor resolution.

Now, if we assume an AUV capable of tracking a thermocline, a large distance can be covered in a small amount of time. Moreover, the resolution is dramatically increased. This illustrates the first advantage of using an AUV to track the top of a thermocline: It would allow for a fast and high-resolution snapshot. This would make it possible to obtain a precise map of the top of the thermocline, which can be useful for various applications, as well as for the verification or refinement of some theoretical models. The applications include understanding and forecast of weather systems, sound propagation for oceanographic and military purposes, biological studies and so on.

The second advantage of the tracking is that once we are able to track a thermocline, we can measure various parameters along its extent. This cannot be done otherwise, except maybe using some kind of profiler to locate the thermocline while traveling and towing a sensor platform, which is obviously more expensive. Being able to measure various parameters along a thermocline boundary would improve our understanding of the phenomena that take place at the interfaces between layers.

## ***II.2. Feature-Based Navigation and Feature Tracking***

Feature tracking is one particular example of a larger class of problem known as feature-relative navigation (or feature-based navigation) [10], which requires the vehicle to determine its trajectory online, in response to various sensor data.

### **II.2.1. The Concept of Feature-Relative Navigation**

The problem consists of locating a feature of interest without exhaustive search, and tracking it in an environment where a-priori information is unavailable [10], [11]. The goal is to maximize the amount of data one can obtain from these features.

Bennett and Leonard [10] suggested a distinction of three scenarios in which the capability to navigate relative to contour features is required: adaptive mapping of a region, adaptive mapping of a dynamic structure and geophysical navigation using natural terrain features. They gave examples for each case. Automated survey of ice keels in the Arctic presents one instance of the problem of adaptive region mapping, in which an AUV should be capable of locating and mapping the extent and distribution of ice keels. Examples of adaptive mapping of a dynamic feature are mapping of mixing-zones, or tracking of dynamic features such as eddies. The work conducted at FAU in order to have an AUV follow the wake of a surface vessel [12] is also an example of these problems. The third application, geophysical navigation using natural terrain features is a technique whereby a vehicle matches sensor measurements to an a-priori map to determine its position.

Current research areas encompass military applications such as mine-hunting [13], [14], commercial applications, like the survey of man-made underwater structures such as

pipelines, telecommunication cables, and so on [15], and scientific applications such as efficient oceanographic sampling [10].

### **II.2.2. Advantages**

It is commonly admitted that “the full potential of AUVs to revolutionize ocean sampling cannot be realized until methods are developed to allow AUVs to react to sensor measurements in real-time as they are being acquired” [10], [16].

The main advantages in the use of AUVs for ocean sampling purpose are autonomy, relative low cost, expandability, and the possibility for discrete or clandestine operations. The autonomy allows the platform to be highly independent of the surface support and human supervision. This has a great impact on operations costs. Moreover, AUVs are decoupled from surface, which allows them to operate in conditions where towed or remotely operated platforms are unusable, such as in rough seas [17]. Their autonomy and possible expandability allow AUVs to perform various missions in possibly hazardous environments [18]. Finally, the discretion and possibility for clandestine operation are some important advantages mainly for military applications, where it is not desirable that the enemy be aware of the operations.

Implementing tracking capabilities, or more generally feature-relative navigation capabilities, can improve the use of AUVs, given that they may dramatically enhance their efficiency in terms of survey duration, resolution, power consumption and data quality. A typical AUV mission may require the vehicle to locate and sample a feature of interest while traversing a minimal distance [11]. Although the commonly used “lawn-mower patterns” ensures a complete coverage of the given area, by choosing this method,

one is already committed to traversing large regions that may provide little or no information.

Oceanic processes are characterized by both temporal evolution and spatial variability [19]. Therefore, for efficient ocean sampling, it is necessary to acquire synoptic data at a fast rate, in order to avoid temporal smearing, and with a high sampling density, to avoid spatial aliasing in the reconstructed field. But usually, to obtain significant information, it is also necessary to gather data over a wide area, which is contradictory with the previous requirements. For surveys carried out with AUVs, compromise between resolution, total survey time and area on one hand, and vehicle speed and energy on the other, must be made. One proposed way to improve the efficiency is to adapt that compromise to the environment and the information that is being acquired. These strategies, known as intelligent or adaptive sampling, attempt to increase the survey efficiency by concentrating measurements in region of interest [20].

### **II.2.3. Difficulties**

Feature relative navigation tasks are difficult because they require the integration of advanced techniques from intelligent control, navigation, sensor fusion, perception, modeling, mapping, and others, in real-time, to enable the vehicle to perform its mission safely and efficiently [10].

The vehicle has to constantly plan its local path based on measurements that are being acquired. The difficulties in local path planning are due primarily to the complexity and uncertainty of environment model and sensory information, and the contradiction between inaccurate vehicle model and vehicle control in real-time [21]. Moreover, the



difficulty increases as the variations of the measured parameters decrease and approach the noise level or the resolution of the sensors. A lot of processing is required to obtain a good perception of the environment on which a trajectory planning decision can be based, but these computations have to be done in a very limited time, so as to enable the vehicle to react quickly enough to the changes in its environment.

Bennett and Leonard stated in April 2000 [10] that “In these scenarios..., it is desirable for the AUV to detect and track features based on the data measured by the vehicle sensors. These tasks are beyond the state-of-the-art of what has been demonstrated to date in high level control of AUVs.”

#### **II.2.4. Techniques Involved**

The techniques used for efficient adaptive oceanographic sampling range from fairly unsophisticated strategies, like appropriately-sized grid surveys to quite complex algorithms such as coupled observation-modeling systems [20]. Between these two extremes lie various other methods, by which measurements are concentrated in the region of greatest interest, usually regions with high spatial gradients. For example, to map an ocean front, an AUV might first run a very coarse survey to localize the front, and then concentrate the operations in its vicinity. The implementation of feature-relative navigation possibilities requires some detailed study in three areas, namely feature identification, search methods for feature finding and intelligent control for feature tracking.

### **II.2.4.1. Feature Identification**

Feature identification relies on the use of multiple sensors whose information has to be combined using sensor fusion techniques. Sensors outputs have to be processed efficiently, in order to enhance the qualities of the data. Mobile robots most likely have multiple sensors. Traditionally, sensors have been used purely complementary. Nevertheless, there are now some methods of close integration of multiple sensor information that can be applied before modeling, matching or identification of targets [15], [17], [22]. The complexity of this sensor fusion spans all the way from the purely elementary to the very complicated.

The sensors in use today all have their special characteristics and are therefore generally aimed towards specific areas where their properties are best utilized. This usually gives reasonably good results but it may not be good enough for some tasks, especially those that demand a high reliability, or involve operations in difficult environments [17]. In order to ensure high quality data in all circumstances, it is necessary to combine in a convenient way data from multiple sensors, taking into account their properties such as update rate, range, confidence and accuracy. The main reason for integrating these sensor data is to achieve a common representation that not only is as good as but better than what would be achieved by processing the data independently and then comparing results. Using multi-sensor fusion techniques on a system may improve reliability, robustness, confidence, efficiency and resolution. Four level of sensor fusion are commonly recognized: signal level, pixel level, feature level and symbol level (or intelligent level). A complete intelligent system may involve multi-level sensor fusion [22]. It is also important to merge the data from the sensors at an early stage, not only to reduce the

amount of redundant information, but also to make it possible for higher level algorithm to take advantage of the process as early as possible. These techniques of sensor fusion are usually recognized as particularly important in target tracking applications. Examples of this sensor fusion technique are the use of redundant sensors to obtain a single information with high reliability and high signal to noise ratio, and the use of radically different sensors that co-operates to provide complex information.

Another concept, which has emerged from the Massachusetts Institute of Technology (MIT) Artificial Intelligence Lab, is the concept of sensor fission [23]. Based on the observation that sensor fusion techniques may be computationally expensive, they decided to ignore it. Instead, they use a form of sensor fission where different sensors trigger different behaviors, and arbitration is done at the actuator stage. By organizing the intelligence system in this way, such that various sensors introduce themselves at various levels of control to initiate distinct behaviors, there is no need for maintaining a model for the world or having to make judgments about which sensor to believe should there be conflicts. Nevertheless, it appears that this kind of judgment is still done at the arbitration level.

#### **II.2.4.2. Feature Finding**

One of the key issues in feature tracking is feature finding, which involves a searching method. The searching for any target is nothing more than running along a selected path until the target is found. But the choice of that path has important effects on the probability to find the target in a reduced amount of time.

The simplest method that has been used is the straight-line pattern. The AUV runs

straight ahead until either the feature is found or a timeout or out-of-range occurs. Obviously, such a method is not very efficient unless it is a priori known that this trajectory will cross the feature. That may be the case for the tracking of a feature with a long extent, such as cables.

A more exhaustive method is based on regular patterns such as “lawn-mower” pattern, which involves multiple parallel straight line, radial pattern, which consists of following different radii of an imaginary circle, or, more suited to the vehicle dynamics, circular patterns. These patterns allow an exhaustive coverage of a small area. Nevertheless, these patterns may take an important amount of time. Moreover, if once the feature is found, the searching is aborted, it can prevent from locating other features that may be more interesting (depending on what features we are looking for). Therefore, these methods may be subjected to what is usually called deadlocks [11].

More recently the idea of intelligent search method has been introduced, such as simplex search, annealing, or taboo search (or tabu). All of these offer viable solutions to the problem of goal optimization in a relative unknown environment [11].

### **II.2.4.3. Intelligent Control for Feature Tracking**

Once the target has been found, the AUV must track it. The simplest controller that can be used is based on a closed loop control of the distance between the target and the tracker, as measured by various sensors. Nevertheless, in some cases, depending on the feature to track, this kind of control may be totally inefficient. New “intelligent” control techniques are being applied to target tracking. In general, the intelligence of a control system can be characterized by its ability to manage and process uncertain and imprecise

information with attributes such as adaptability, memory and learning. Intelligent control as practiced today encompasses many fields from conventional control to the more recent fuzzy, genetic, and neuro-control technologies.

Numerous studies have shown that the solution for efficient tracking is based on the implementation of a behavioral architecture, with emphasis on the perception-action behavior [10], [14], [22]. Indeed, such an architecture has the advantage to guarantee a faster response than a hierarchical architecture. In hierarchical architectures, the high-level control relies on a central world model. What is being sensed has to be compared with the available model, then incorporated with all the other pieces of information, on which a decision can be made. The main drawback of this architecture is its inability to handle rapidly changing environment in a timely fashion [10]. A direct perception-action method decomposes the procedure of complex information processing, simplifies the uncertain problem in high-level, reduces the computational complexity of planning, and enhances the real-time reactivity to the environment [21]. In these behavioral architectures, the control strategy is embedded in a set of preprogrammed condition-action pairs or “reflexes” [10]. Moreover, the decision-making among all the information and multiple goals is inherently done by the arbiters among the behaviors, which simplifies the high-level planning tasks. It has also been shown [21] that local path-planning and navigation control based on fuzzy logic are also key techniques for intelligent reactive planning. Fuzzy control does not rely on an entire environment model, reduces the quantity of complex computation and reasoning, and is able to handle both sensor inaccuracy and system model uncertainty [24]. Also, the use of fuzzy logic allows to take into account some elastic constraints that allow partial satisfaction [25].

The first technique historically used has been predictive tracking based on “information augmentation” [26]. It aims at maintaining robust tracking performances, augmenting the nominal tracking function by knowledge-based data such as motion capabilities of the target. Then, using sensor measurements, projected on the target model, the robot is able to predict the region in which the target is likely to be at the next instant [15]. The main problem is that it requires a model accurate enough to be useful. In many cases, the exact models are either difficult to obtain or need complex mathematical descriptions, thus cannot be used in real-time. An answer to this problem can be brought by learning techniques that the robot can use to improve, or build online, the target model [27].

Another way to use learning methods, commonly supposed to be more efficient though more complex to design, is to get rid of the target model, and let the robot improve its tracking capabilities by learning [24], [28]. Neural networks techniques are generally used in mobile robots for environment recognition or to achieve reactive navigation. It has been shown that complex models and control schemes could be replaced by a connectionist learning approach, based on neural networks. Using these techniques, the robot learns to change its behavior directly from experience of its actions in the world, based on a connectionist implementation of model-free reinforcement learning, through the use of a reward signal [24].

Several other aspects or techniques of intelligent control are still being developed or are to be developed. One of the main goals of implementing intelligent control for tracking purpose is to emphasize a strong perception-reaction behavior. Such a behavior tends to overrule the complex phases of reasoning and modeling in order to make the AUV more reactive while its structure becomes less complex [14], [10].

## ***II.3. The Ocean Explorer***

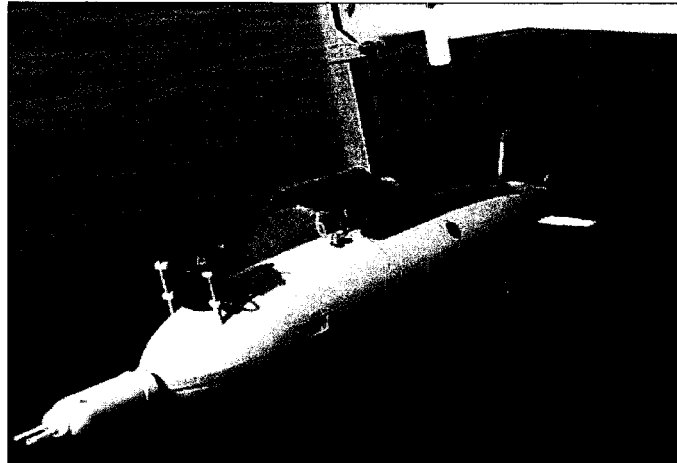
This part describes the FAU AUV used for this thesis work. A brief overview of the AUVs existing at the Advanced Marine Systems (AMS) Laboratory at FAU is given, followed by a detailed description of the knowledge required to understand the development of the OEX-D from parts taken from existing vehicles.

### **II.3.1. Existing Advanced Marine Systems Laboratory Vehicles**

Since the late eighties, the Department of Ocean Engineering (OE) at FAU has been strongly involved in the development of AUVs [29]. Particularly, the AMS Laboratory is pursuing a wide range of research projects related to AUVs, Intelligent Control Systems and Naval Shipboard Automation [30]. Small, long range, low cost AUVs have been developed as sensor platforms for educational, scientific and military applications [31]. Currently, two generations of AUVs are being developed or refined: The Ocean Explorer (OEX) B and C series, and the Morpheus.

#### **II.3.1.1. The OEX-B**

The Ocean Explorer is a family of small, low-cost AUVs designed and built at FAU. They are primarily designed for oceanographic survey missions and coastal warfare [32]. The OEX (Figure 8) is a platform for multiple sensor payloads used for performing search and mapping operations in shallow water [33].



*Figure 8: The Ocean Explorer (OEX)*

#### **II.3.1.1.1. Features**

The vehicles most significant feature is their highly reconfigurable architecture. The idea that allows achievement of this goal is to gather the mission dependent systems and sensors in a payload that is self-contained and exchangeable [32], [33]. The OEX then consists of:

- A 4ft long tail section, housing of all the navigation systems and sensors, propeller, batteries, computer and controllers,
- A collection of modular payload noses that are mission dedicated. The nose section is totally available for any desired type of sensor, except for an emergency dropweight system that allows the vehicle to surface should a problem occur.

When necessary, a mid-body payload can be added. The sections are assembled by bayonet-mount interfaces, that allow for a fast and easy switching between payloads.

The vehicle is flooded, and the equipments are enclosed in a few waterproof containers, named pressure vessels, connected together using wet cabling [34].



The OEX is a Gertler body, whose dimensions are limited to 21 inches diameter, and 7 to 12 feet length, depending on the payload (Figure 9). The vehicle is rated for a 300m depth, and a speed range of 2 to 5 knots. Its batteries allow an endurance of 4-12 hours, depending on the configuration, the housed payload and the speed.

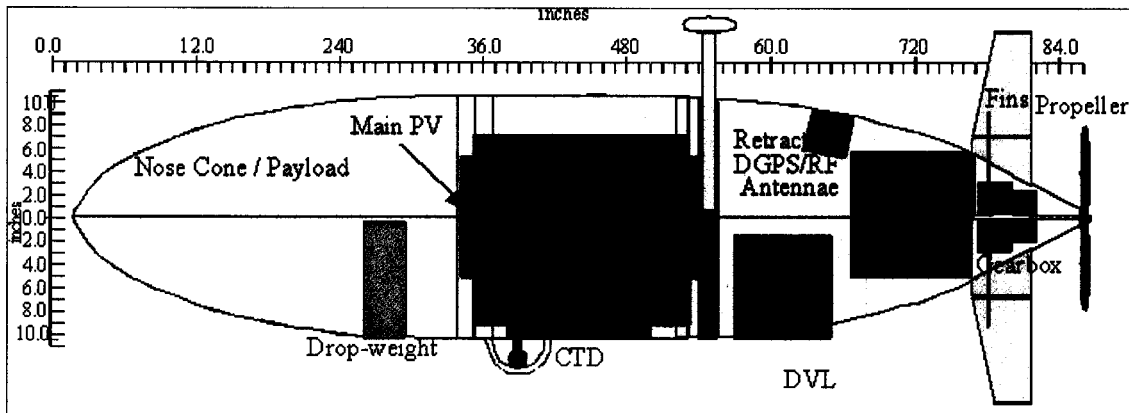


Figure 9: Ocean Explorer System Overview

An autopilot running on a computer controls the vehicle [32]. The preprogrammed missions [33] consist of a set of waypoints and setpoints, along with some background requirements such as bottom avoidance, health monitoring, and other safety features.

OEX AUVs house various sensors. Some of them are dedicated to navigation while others are only used for data acquisition that are mission dependent. At this time, many payloads have already been designed. Other designs are made easier by the use of a simple mechanical, electrical and logical interface that allows non-AUV engineers to design and integrate new sensor payloads [31],[35].

So far the OEX-B and C are similar. They differ mainly by their architecture which will now be discussed more specifically.

### **II.3.1.1.2. Hardware**

The hardware architecture is essentially made of a common network, based on LonWorks and Neuron technologies, on which sensors, actuators and main computer form some “nodes”, communicating together. One main computer is in charge of the high-level control and the computationally intensive tasks, and various nodes take care of the local low-level control and data acquisition, which is commonly referred to as local logic. This creates an Intelligent Distributed Control System (IDCS), which plays an important role in the reconfigurability of the OEX [32].

On the OEX-B, the main pressure vessel, housing the main computer, is based on a VersaModule Eurocard bus (VMEbus). The VME architecture, defined by the IEEE 1014-1987 standard, is based on a backplane mounted as a cage (or “crate”) on which standard VME cards are connected on single slots [36]. On that bus is mounted the main computer, based on a Motorola 68030 processor. The crate also houses other cards such as LonWorks gateway, Ethernet controllers, and serial adapters [32].

The LonWorks gateway is used to communicate with the IDCS. Most of the sensors and actuators communicate with the main computer through LonWorks, although a few of them use specific interface such as direct connections to the VMEbus, or serial RS232 ports.

Throughout the vehicle runs a “main bus” carrying power (48 VDC, and sometimes 12 and 24 VDC), and communications (LonTalk, Ethernet and sometimes serial) [37].

### **II.3.1.1.3. Software**

The OEX-B software is based on two parts: The main computer software and the Neuron nodes local logic applications.

The main computer runs the VxWorks operating system (OS), available from Wind River. This is a Real-Time OS (RTOS) particularly designed for the embedded system industry, and available on most popular computer platforms [38]. That computer mainly runs a high-level mission scheduler, the Navigation and Autopilot managers, as well as some processes that handle communications and data logging to name a few.

Then each node on the network participates in the software, running its own local logic application which is mostly responsible for local low level control, data acquisition, and health monitoring. These applications, designed using LonWorks tools, share information with one another through the network.

The OEX-B software is not described further here since the main computer software will be replaced, and the node applications are described later.

### **II.3.1.2. The OEX-C**

The OEX-C is very similar to the OEX-B, since it was designed so as to implement a few improvements. Only the differences with the OEX-B are discussed hereafter.

#### **II.3.1.2.1. Improvements**

Among others, the following significant improvements were implemented on the OEX-C:

- Standardization: Standardization of the IDCS, the main bus, the software, the health monitoring and so on,

- Addition on a Inertial Navigation System (INS),
- Modification of the batteries so as to increase their lifetime, improve their charging methods, in a word have much efficient batteries,
- Addition of a much convenient and powerful Human Machine Interface (HMI) for a diver/operator,
- Addition of a much convenient development and programming interface.

As far as the standardization is concerned, where the OEX-B sensors sometimes communicate over dedicated connection to the main computer (VME or serial), every sensor and actuator of the OEX-C communicates over LonWorks. The health monitoring is now implemented the same way in every pressure vessel. Where the main buses on the OEX-B sometimes carried 12 and 24VDC and serial lines, the standard buses on the OEX-C only carries 48VDC, LonTalk and Ethernet. The Ethernet connection is not intended for subsystems to communicate together, but with an external computer.

So as to improve the navigation efficiency, a new INS was designed by Grenon [29]. This dramatically improved the accuracy of the position estimator. The implementation of this INS required the installation of a second computer.

Concerning the diver/operator interface, the OEX-B only offered a control box made of a few LEDs and switches that allowed very limited operations. A console was designed for the OEX-C, to offer more control through the use of menus and dot matrix text display.

A much convenient development and programming interface was implemented through the design of a new software for the main computer.

### **II.3.1.2.2. Hardware**

The OEX-C hardware is based on the same kind of network as the OEX-B. On that network are connected the sensors and actuators, and two computers.

The two computers, are based on a PC104 architecture. This architecture, born from the standards established in 1992, has been created with the goal to offer the power and flexibility of a PC in a size and power requirements suited for embedding. PC104 cards stack together to eliminate the need for backplane or cage [39]. These computers are Pentium-based PC104 computer cards . Additional cards provide in the same stacks Hard Drives and LonWorks interfaces. Both computers interface with LonWorks through a Microprocessor Interface Program (MIP) board, and are moreover connected together in a local Ethernet network accessible from the outside.

### **II.3.1.2.3. Software**

While the computer-hosted software is responsible for the high level scheduling and computationally intensive tasks, the low level software is distributed over the LonTalk network. Each node is programmed with its own local logic, as on the OEX-B. The nodes applications are described in details later.

The OEX-C achieves high-level command and control through use of QNX microkernel-based OS, available from QNX Softwares Systems Ltd., which provides a true real-time, multi-tasking, multi-threaded OS. QNX also provides priority-driven preemptive scheduling and fast context switching [40] which are essential for an effective and robust real-time system.

The main control software is based on a hybrid architecture that combines hierarchical and behavioral control architectures.

The hierarchical approach is a planning-based multilayer architecture. Usually, sensors information arrives at the lowest level and is propagated to the highest level. Each layer modifies the data. At the highest level, a decision is made, which in turn is propagated to the low-level actuators layer.

On the other hand, the behavioral architecture approach does not recognize such high/low level distinction. The raw sensor information is made available to any process that requires it. Then every task is decomposed on basic behaviors that executes independently, cooperatively or competitively. At the other end, the outputs generated by the behaviors are combined by some arbiters that in turn control the actuators.

The hybrid architecture is a mix of the previous ones. A high-level planner mainly loads, configures and activates as necessary some behaviors from a library, and is responsible for the whole supervisory operations. Each behavior is formed of a reactive combination of primitives [41]. It performs an independent task that cooperates with other behaviors to perform a given mission. The output commands – or desired setpoints – generated by the behaviors are then combined by some arbiters that decide what the final output combination is. Finally, at the lowest level, the sensors deliver their information and the actuators execute the commands.

The software architecture of the OEX is based on multiple processes, managers and arbiters, shared memory, logger and monitor [25], [41]. An overview is provided in Figure 10.

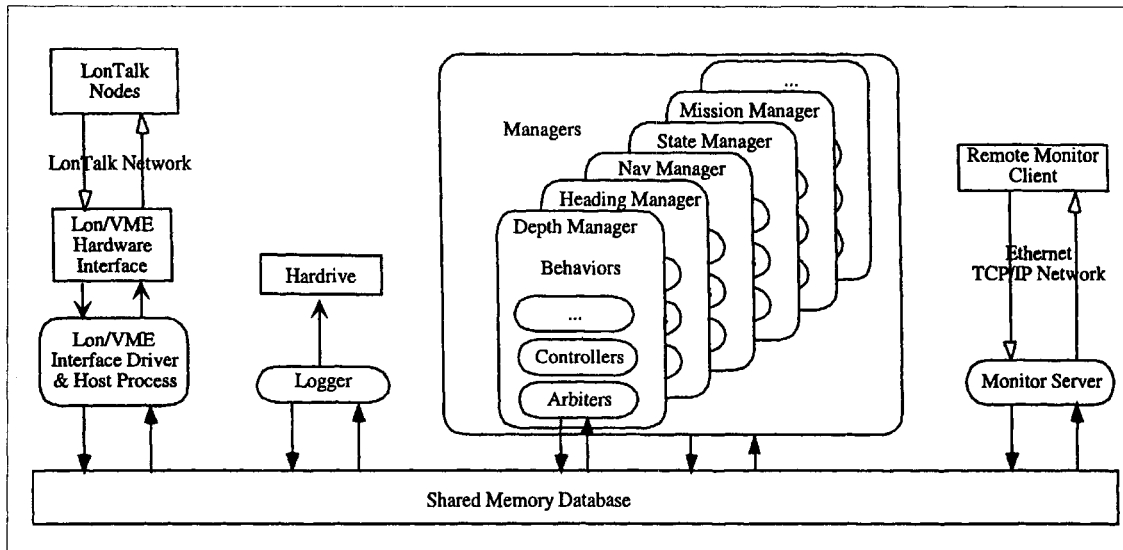


Figure 10: Overview of the Control Software Architecture

The multiple processes strategy allows to deal with numerous tasks whose computation load vary, and allows for a more efficient scheduling. This also makes it easier to integrate new modules in the system.

The managers are used to implement a particular behavior, and schedule its execution.

The arbiters are special behaviors used to resolve among other behaviors. They are responsible for taking the outputs from other behaviors and deciding which combination should be the active output. The decision is made based on the confidence level, importance or priority of each behavior.

The shared memory allows the processes to communicate with each other. It has been chosen because it is considered the most flexible yet simple kind of InterProcess Communication (IPC) [25]. The common data are stored in a memory segment that may be accessible by any process that asks for it. Data integrity is ensured through the use of MUTual EXclusion (MUTEX) scheduling techniques, based on critical sections of code, that allow only one process to write data at a time.

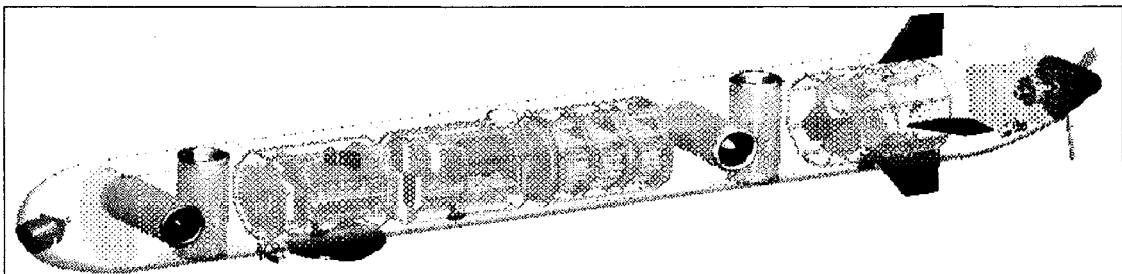
The logger is responsible for automatically logging some selected variables from the shared memory into a file that can be used for post-processing.

The monitor allows a distant host system to remotely monitor or edit selected shared-memory variables.

### **II.3.1.3. The Morpheus**

The design of the Morpheus results from experiences learned from the previous generations of AUVs built at FAU, while adding new original features. Two of the main lessons learned from the Ocean Explorers are the importance of modularity, and the relation between operation cost and weight and size [34].

The result is an ultramodular plastic mini AUV, called the Morpheus, as a reference to the Greek god who had the ability to change his shape. The Morpheus (Figure 11) is composed of several standard pressure vessels that form the hull of that dry vehicle. Modules are selected and assembled based on specific needs of each mission.



*Figure 11: The Morpheus Ultramodular AUV*

The Morpheus's hardware is based on the ideas that have proved their efficiency with the OEXs: distributed control network and PC104 computer stacks. The software is very similar to that of the OEX-C, especially the high-level software, and therefore is not described specifically here.



## **II.3.2. The New OEX-D Project**

The general idea that drives the OEX-D project is to upgrade the OEX-B using some features of the OEX-C and Morpheus.

The vehicle, as an OEX-B mechanical system, will be kept as is, while the main computer and its software will be replaced by that of the OEX-C/Morpheus. The goal to achieve is to offer an operation and development interface similar to that of the OEX-C/Morpheus. Whenever possible, the local logic embedded in the OEX-B nodes will be kept since it is designed to specifically interface the OEX-B sensors and actuators.

Therefore, the OEX-D is merely made of components coming from different vehicles. These components will now be described with more details than the overview given above, with emphasis on the parts we have to deal with for the thesis.

### **II.3.2.1. The Common Intelligent Distributed Control System**

The Intelligent Distributed Control System, common to the OEX and Morpheus AUVs, is based on the LonWorks technology of which we will now give an overview. A discussion with a much specific emphasis on the application to the case of the OEX follows.

#### **II.3.2.1.1. General Idea**

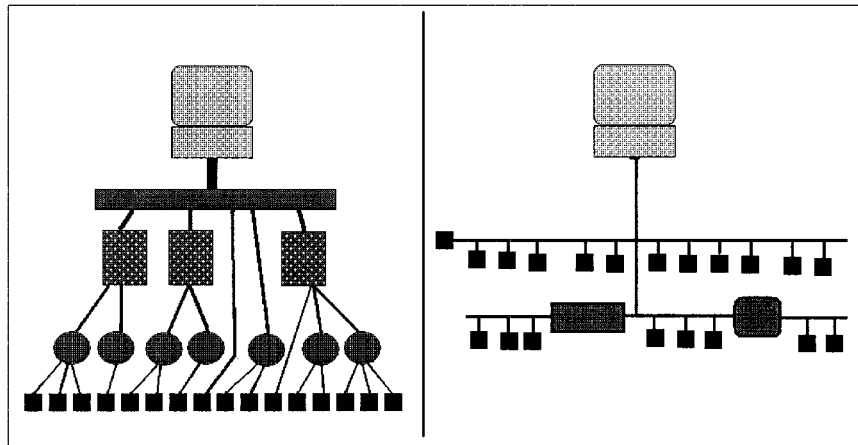
A control system is usually made of sensors, actuators, and controllers running the control application programs [42].

Until a few years ago, most of the control system were typically based on a centralized architecture in which every sensor and actuators were controlled by a single computer [43]. The application running on the computer was then complex, had to deal with many

tasks simultaneously so as to perform the control, and was not easily modifiable.

The idea behind distributed control is to divide a control system into several subsystems, each made of sensors, actuators and controllers [42], [44].

Figure 12 compares centralized and distributed control architecture.



*Figure 12: Centralized vs. Distributed Control Architecture*

The key components in a distributed control system are the communication protocol and the hardware support. Until recently, each distributed control problem was considered unique because of a lack of standards in that domain. LonWorks technology is one of the standard solutions that can be used to create such distributed, interoperable systems.

The LonWorks technology, developed by Echelon and available as an open standard, is a platform that allows to create control system architectures that are open, distributed and interoperable [42]. This technology is based on a highly distributed, peer-to-peer architecture that connects together several device so that the local logic embedded in each device participates in the whole control application, hence the distributed control. LonWorks technology consists of tools, modules and circuits required to build intelligent devices and install them in distributed control networks [43].

### **II.3.2.1.2. Components**

The IDCS is based on the following components [42]:

- Nodes, that are basic elements of control. A node is made of a controller and the sensors and actuators it supervises,
- One or several communications media to connect the nodes. If several different media are used, they are connected together with either bridges or routers,
- A logical design of the network, that describes how a node is logically connected to another,
- A standard communication protocol: the LonWorks protocol also known as LonTalk protocol, described by the ANSI/EIA 709.1 Control Networking Standard.

Each node is in turn composed of [45]:

- A programmable device, similar to a microcontroller or microprocessor, that implements the LonTalk communication protocol, and performs the application it is programmed for. This device is commonly referred to as Neuron Chip.
- A Transceiver that provides the interface between the Neuron Chip and the communication medium
- A circuitry to connect the Neuron Chips to the I/O devices they control and supervise,
- An application that defines the behavior of the node.

Figure 13 details a node attached to the channels linking several nodes.

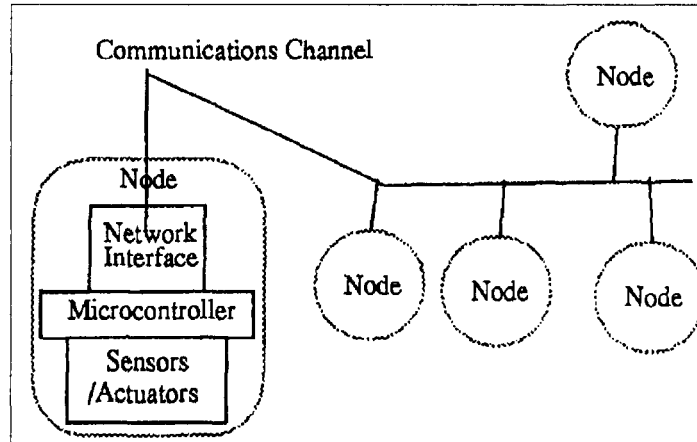


Figure 13: A Node Attached to the Distributed Control Network

### **II.3.2.1.3. LonWorks Application**

LonWorks applications consist of intelligent devices or nodes that communicate over a control network [43], [45]. The major particularity of LonWorks is its communication interface based on Network Variables (NVs). The idea is to make information, stored in variables, available throughout the whole network. This allows to create a control network that is information-based instead of command-based [42]. Where the network messages on a command-based system are explicit commands from a node to another, the information-based system only communicates information about the state of a node, so that each other node can make a decision depending on the whole state of the system, and perform the adequate command. Doing so, each node only runs its own local logic based on state information coming from the outside, and does not have to model the whole system. This allows for a better interoperability, scalability and reconfigurability.

The information transiting on the network are either based on [43]:

- Network Variables (NVs), variables “stored” on the network and therefore accessible to any node. In order for a node to access a variable , either as a writer or a reader, the

NV has to be bound to that node, that is logically connected,

- Messages, that can be unicast (towards a single node), multicast (towards several nodes) or broadcasts (toward every node).

The application is programmed using network management tools to create the logical network, and a language to program the local application of each node. This language is the Neuron C, supporting most of the feature of ANSI C, and including extensions to use the specificities of Neuron Chip [46]. The main features of that language are: A new Network Variable (NV) object, a new “when ( )” statement, introducing event-driven control, and I/O objects and operations specific to Neuron Chips.

#### **II.3.2.1.4. Application to the OEX**

On the OEX, and later the Morpheus, the IDCS based on LonWorks relies on usually one node per actuator or sensor to control, and one node for the main computer(s), connected together by a 1,25Mbps Twisted Pair (TP) serial bus. The characteristics of that bus are summarized in Table 1 [43]:

<b><i>Channel Type</i></b>	<b><i>Medium</i></b>	<b><i>Bit Rate</i></b>	<b><i>Compatible Transceivers</i></b>	<b><i>Maximum Devices</i></b>	<b><i>Maximum Distance</i></b>
TP/XF-1250	Twisted Pair, bus topology	1.25Mbps	TPT/XF-1250	64	125 m

*Table 1: IDCS Channel Characteristics Summary*

An overview of the OEX-B IDCS is given in Figure 14.

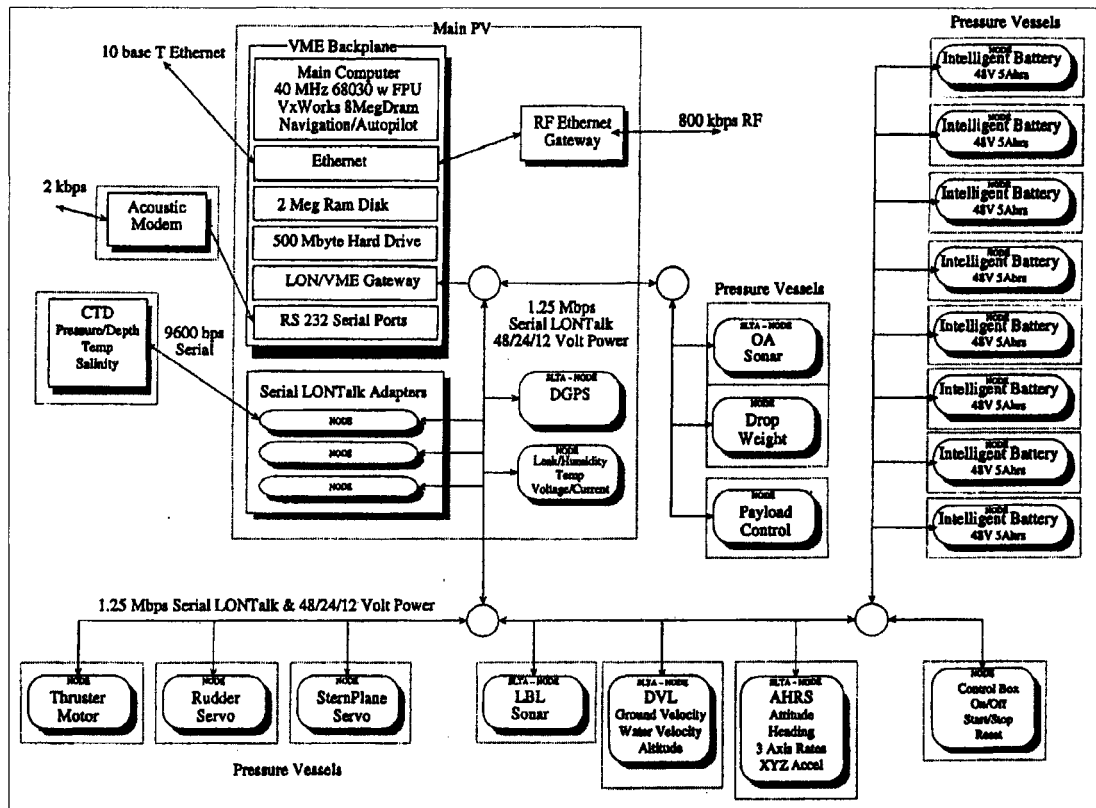


Figure 14: The OEX-B Intelligent Distributed Control Network

The advantages that IDCS brought to the AUVs are [44]:

- It allows a faster and easier reconfiguration, through the use of “plug and play” devices. Changing a particular sensor for instance, is totally transparent for the system. Only the local node has to be modified so that the data coming from the sensor are parsed to fit the previously defined NV interface.
- Modifications are limited: adding devices mainly consists in plugging in the network and writing the software that uses the device. Software rewrites are limited because the communication protocol, based on the seven layer ISO/OSI model, provides high-level functionalities.

- The robustness is improved. When a single node crashes, the rest of the system hangs, and only a limited part of the system has to be changed, which can be done easily thanks to the use of standard components.
- The main computer performs less tasks and hence can be solely devoted to high-level control.

#### **II.3.2.1.5. Parts**

From the hardware point of view, the nodes used on the OEX-B, and therefore on the OEX-D first time, are either based on PSG10, LTM10 or HPSN. Both PSG10 and LTM10 devices are off-the-shelf products available from Echelon. The HPSN is a node custom made by the Ocean Engineering Department's Electronics Laboratory.

The PSG-10 is an embedded Programmable Serial Gateway (PSG) used to build gateways between LonTalk network and a system with RS232C interface [47]. It consists of a Neuron Chip, Random Access Memory (RAM), application Programmable Read Only Memory (PROM), a First In First Out (FIFO) buffered serial interface, and a port to connect a transceiver towards a LonWorks channel. The important thing about this part, as far as the development is concerned, is that the application memory segment is in a UltraViolet Erasable PROM (UV(E)PROM). To program such a device, the UVPROM has to be erased using UV light, then programmed with a PROM burner.

The LTM-10 LonTalk Module (LTM) consists of a miniaturized board hosting a Neuron Chip, flash memory, RAM, and connectors for application I/Os and transceiver [47]. The programming is much more convenient than that of PSGs, since the application is stored on a flash memory, or Electrically Erasable PROM (EEPROM). This allows to program

the memory without having to take it out of the device.

The HPSN (High Performance Standard Node) is a compact, multi-purpose, low power board that implements a custom node based on a Neuron Chip, on which health and environmental sensors, serial interfaces, other I/Os and regulated power supply come as standards.

All these devices are attached to the network through a TPT/XF-1250 Twisted Pair Transceiver (TPT). The TPT/XF consists of a transformer isolated differential Manchester encoding communication transceiver and connectors for power supply, Neuron Chip Communication Port (CP) and TP network lines [47].

The main computers are connected to the LonWorks network through a Microprocessor Interface Program (MIP) board. The model used on OEX-C/Morpheus, a MIP-P50, is a standard product available from IEC Intelligent Technologies as a PC104 Adapter [29]. It is a firmware that transforms a Neuron Chip into a communication co-processor for any host processor. This makes it easy to create host applications that communicate using the LonTalk protocol and run on processors other than the Neuron Chip, so as to add processing power to a LonWorks network [48]. The MIP-P50 communicates data with the host processor through the PC104 bus, and an optional uplink interrupt line can reduce the latency on incoming traffic [49]. On the other side, it is attached to the network using a TPT/XF-1250 transceiver. The connection is buffered to ensure data integrity during network traffic burst [29].



### **II.3.2.2. The OEX-B Sensors and Actuators**

Thanks to this LonTalk network, the mandatory sensors and actuators, coming from the OEX-B tail, are connected together. A brief description of each device and its corresponding node is given hereafter.

#### **II.3.2.2.1. Main Health System**

The Main Health system is in charge of monitoring the health and controlling the power supply of the main pressure vessel. It constantly monitors the temperature, humidity, pressure and checks for a leak. This system is custom built by the Electronics Laboratory. It is based on health sensors (temperature, leak, voltage, current and so on), connected to a LTM10 device, on which the application performs the health monitoring and power control tasks.

#### **II.3.2.2.2. Global Positioning System (GPS) and Differential GPS**

This subsystem is in charge of the GPS/DGPS positioning of the vehicle. It consists of a topside dual antenna, a Radio Frequency (RF) splitter to separate GPS and DGPS signals, and a DGPS/GPS receiver. The GPS receiver node is based on a PSG10 that parses the GPS/DGPS data coming through the serial interface, and sends the information to LonTalk. It also implemented a Virtual RS232 link (VRS232) that allows an operator to directly communicate with the GPS from a terminal emulator on the main computer [41].

#### **II.3.2.2.3. GPS/DGPS Antenna**

This subsystem, although called GPS/DGPS antenna, is rather the motor that controls the rising and lowering of the topside antenna. The antenna is motorized so as to be about

30cm atop the vehicle when deployed, to improve the reception of RF signals, and, once retracted, merge with the surface of the vehicle body, to reduce the drag. The node is based on a LTM10 running the application that monitors the antenna motor health, and controls the motor according to oncoming commands to rise or lower the antenna.

#### **II.3.2.2.4. Conductivity Temperature Depth (CTD) Sensor**

The use of this sensor is dual. Its primary use is for navigation purposes: measuring pressure and conductivity, it can compute the depth, and using also the temperature, can compute the sound velocity, used to correct the measurements obtained from other sonic navigation sensors. On the other hand, the CTD data may be of interest for oceanographic studies. The node is based on a PSG10 that parses the CTD data coming through the serial interface, and sends the information to LonTalk. It also implements the VRS232.

#### **II.3.2.2.5. Doppler Velocity Log (DVL)**

This sensor essentially measures the relative velocity of a group of scatterers with respect to the body, by measuring the Doppler shift of signal transmitted along four beams. The DVL can measure its ground velocity, or the velocity of a water volume. Gating the incoming echo with respect to time, that is distance, the DVL can measure the velocity of different volumes, or bins, of water. In such a case the DVL profiles the current, and acts as an Acoustic Doppler Current Profiler (ADCP). This sensor is primarily used for navigation, measuring the velocity of the vehicle and its altitude. It can also be used as an oceanographic payload profiling the current under the vehicle. The node is based on a PSG10 that parses the DVL data coming through the serial interface, and sends the information to LonTalk. It also implements the VRS232.

#### **II.3.2.2.6. Attitude and Heading Reference Sensor (AHRS)**

The AHRS is an Inertial Measurement Unit (IMU), made of 3-axis accelerometer and rate gyro. The data can be used for navigation, and sometimes to estimate the noise due to the vehicle motion and vibration in data recorded by scientific payloads. The node is based on a PSG10 that parses the AHRS data coming through the serial interface, and sends the information to LonTalk.

#### **II.3.2.2.7. TopSide Acoustic Modem (TSAM)**

The AUV employs an acoustic modem for data telemetry and remote command and control of the vehicle [31]. Multiple modems can be used, at least, one being mounted on the AUV, another on the support vessel. This allows the support crew to monitor a few essentials pieces of data from the vehicle, to command the vehicle to start/stop missions, and possibly have it perform predefined macro instructions. The node is based on a PSG10 that bridges the TSAM serial interface and LonTalk.

#### **II.3.2.2.8. Thruster**

The thruster is a Direct Current (DC) motor that, mounted with the propeller blades, pushes the vehicle through the water. The motor can be controlled either in terms of current (torque) or voltage (rpm). It can spin in both directions, pushing the vehicle forward or pulling it backward. To enable close loop control on the rpm, a sensor measures the actual motor rpm. The node is based on a HPSN that interfaces LonTalk with a motor controller board, and monitors the health.

#### **II.3.2.2.9. Fins**

Four fins are mounted near the back of the AUV: two rudders, one on top, the other on the bottom, and two sternplanes, one on each side. They control the direction of the vehicle, in terms of heading (rudders), and pitch (sternplanes). Each fin is mechanically independent from the others, but both rudders and both sternplanes are coupled in terms of command. They are commanded by DC motors to angles between -20 and 20 degrees of the vehicle forward axis. To enable close-loop control, four sensors measure the actual position of each fin. Each of the four nodes is based on a HPSN that interfaces LonTalk with a motor controller board, and monitors the health.

#### **II.3.2.2.10. Dropweight**

This is a very important safety feature of the AUV. It is an additional weight that can be released in case of emergency. Once the weight is released, the vehicle, previously neutral, becomes positively buoyant, and surfaces. The dropweight monitors the whole vehicle network for critical errors indication, and fires an emergency abort and surface command when necessary. It has its own battery that allows the weight to be released even in case of main power outage. The node is based on a HPSN that interfaces LonTalk with a motor controller that commands the dropweight shaft motor, monitors the health of the dropweight and controls the charging of its battery.

#### **II.3.2.2.11. Batteries**

The AUV tail encompass eight battery cans, each providing approximately 48VDC when fully charged, and having a capacity of 5.4Ah, for a total of 43.2Ah, or approximately 2kWh. They provide power to the whole AUV. Each battery node is based on a

customized LTM10 running the application that monitors the battery health, controls the charging, reports information about the available power, and command the batteries to turn on or off. Each battery can charge itself from the power bus. They can be charged while mounted in the vehicle thanks to the health monitoring which triggers a slow charge mode with low duty-cycle when the temperature of the battery is too high.

#### **II.3.2.2.12. Control Box**

The control box is the interface a diver or operator has with the system. It consists of 15 LED indicators that provide information about the state of the vehicle, and three Hall and Reed switches to turn the vehicle on/off, start/stop a mission, or reset the whole system. Attached to the control box is also a circuit-breaker, which is a switch enabling an isolation between the batteries and the other part of the vehicle so that the batteries can be charged by external power without the vehicle being powered on. The node is based on a customized LTM10 running the application that monitors the control box health, monitors the switches and sends corresponding commands on the network, and control the LED display based on oncoming network informations.

#### **II.3.2.2.13. Compass**

This sensor is a magnetic compass, that is a “smart” three axis magnetometer. It measures the Earth magnetic field along three axis, and using incorporated tilt sensors, converts the magnetometer readings to the magnetic heading of the vehicle. It is possible to configure the compass to correct the readings for local deviation so as to output the true heading. The node is based on a PSG10 that parses the Compass data coming through the serial interface, and sends the information to LonTalk. It also implements the VRS232.

### II.3.2.3. The Morpheus/OEX-C Host Software

Now the Host computer software of the OEX-C/Morpheus, that is to be used on the OEX-D, will be described with more emphasis on the developer/integrator point of view.

#### II.3.2.3.1. System Overview

The supervisory control algorithms consist of a group of function controllers termed managers which are responsible for scheduling and monitoring of individual functions or behaviors [41]. There also exist a set of low-level daemons responsible for basic I/Os and regulatory functions. Figure 15 summarizes the main functional groups of the software. A quick description of each of these groups follows.

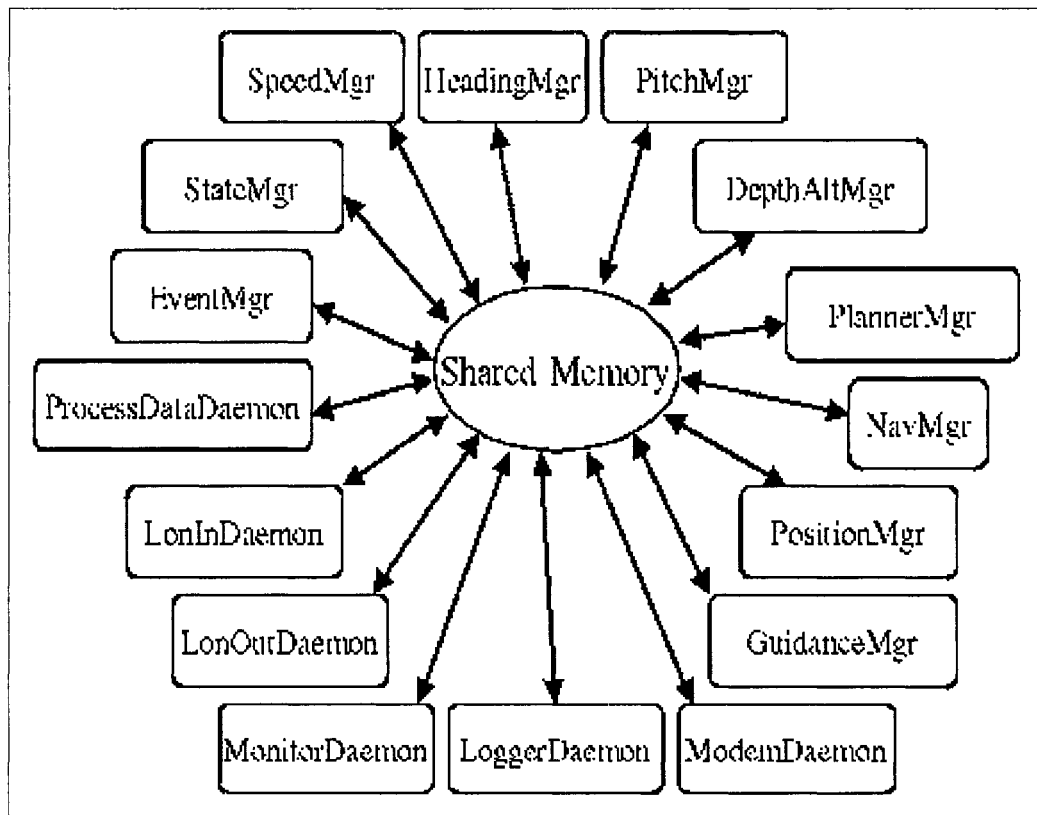


Figure 15: Software Overview: Main Functional Groups

The planner manager is responsible for interpreting the mission description from the input file, and generating navigation commands. The navigation manager is responsible for the supervisory execution of a command. It fetches commands issued by the planner in the navigation queue, converts it into settings for the vehicle autopilot, monitors the completion of the current command and removes it from the queue upon completion (success or timeout). The commands normally enqueued in the planner queue can be overridden by the modem queue for remote control or the error queue for emergency control. The guidance manager takes individual commands from the navigator and distributes them to the lower level behaviors it controls. The position manager estimates the current position of the vehicle. The state manager is responsible from translating measurements from sensors into a description of the vehicle state. SpeedMgr, HeadingMgr, PitchMgr and DepthAltMgr are individual managers responsible for a single control. They may rely on multiple behaviors to perform a control. The event manager monitors events, or conditions, that occur in the system and perform appropriate predefined actions. The LonDaemons are in charge of communications with LonWorks. The monitor daemon makes it possible to monitor and edit shared memory variables. The logger demon regularly copies selected variables of the shared memory into a log file for post processing. The modem daemon handles communication through TSAM. The processData daemon is responsible for the translation of raw data coming from the sensors into converted engineering data.

Among all the processes that run on the main computer so as to have the vehicle achieve its mission, only a few will be described in details here since they are the ones we have to deal with for the development of the OEX-D. These processes are those that interface

with the LonTalk network, sensors and actuators, which differ from one vehicle to another. Indeed, many of the high-level processes only interact with the shared memory and do not really care about what is connected on the outside.

### **II.3.2.3.2. Shared Memory**

The shared memory constitutes the core of the software. It defines a database of variables that any process can access. It basically consists of a common memory segment mapped onto the local context of each process attached to the shared memory. To ensure data integrity, shared memory units are protected by the use of MUTual EXclusion (MUTEX) techniques for critical sections of code. Since the share memory is already working, we do not need much knowledge about the way it is programmed. As we will mainly have to add or remove variables, we only need to look at the shared memory constructor.

The share memory definition is based on the shmem.in file (Figure 16), which lists all the variables to be declared, along with their types, size if necessary, default values, and information about the way they are to be interpreted, such as the unit of their content.

```
GPSHealthIn
{
  busVoltage      "dV"      short = 0;;
  busCurrent      "cA"      short = 0;;
  temperature     "dC"      short = 0;;
  leak            "leak"    short = 0;;
  humidity        "RH"      short = 0;;
  pressure        "kPA"     short = 0;;
};;

AntMotorHealthIn
{
  temperature     "dC"      short = 0;;
  leak            "leak"    short = 0;;
  humidity        "RH"      short = 0;;
  motorCurrent    "cA"      short = 0;;
  antennaPosition "n/a"     short = 0;;
};;
```

*Figure 16: Shmem.in Content Example*



There exist naming conventions, restrictions on the type of variables that can be defined, and grammar rules that have to be respected for that file. More details about these topics, as well as some guidelines about when to group variables in structures or declare them separately, can be found in [41].

A library of shared memory manipulation functions has been written, which allows a process to perform various operation on the shared memory. These functions can be used to read/write the content of a variable, get a local copy of a variable or set a global variable based on a local one, get the address, identification, type, size, dimension, or timestamp of a variable, and so on. Details about these functions can be found in [41].

#### **II.3.2.3.3. LonDaemon Module**

LonDaemon provides a software interface between the application running on the host and the LonTalk network. It basically creates a bridge between the network and the shared memory.

On a regular basis, the LonDaemon checks the shared memory for variables related to outgoing NVs and checks the network for incoming NVs. In case an update has occurred in shared memory, the daemon takes the shared memory variable, translates its format to a LonTalk one and drives the MIP to send the NV on the network. In case a NV update has occurred, the daemon drives the MIP to get the actual value of the NV, converts it from the LonTalk format to the local QNX format, and copies it to the shared memory.

The LonDaemon mainly consists of four processes: lonDaemonRx and LonDaemonTx are drivers that handle the transmission, and lonDaemonNv provides the synchronization between internal variables and NVs. A fourth process, LonDaemonDiag is a utility used

to provide a management interface with the network nodes, and is described later.

The main process we have to deal with to interface the network is LonDaemonNv. Indeed, the LonDaemonTx and LonDaemonRx drivers work by themselves, based on the information provided by LonDaemonNv. In this process, a table lists all the network variables to manage, along with their characteristics such as direction, size, necessity to acknowledge the communication, and the name of the functions that check, update or poll each variable (Figure 17).

{	"MdmToAuvCmds",	31,	NV_IN,	UNACKD,
	LdNvUpdateModemToAuv,		LdNvPollModemToAuv,	NULL },
{	"AuvDataToModem",	28,	NV_OUT,	UNACKD,
	NULL,		LdNvPollAuvToModem,	LdNvCheckAuvToModem },

*Figure 17: LonDaemonNv Table Content Example*

For each oncoming variable, it is necessary to write a function that polls the network, and a second one that updates the share memory with the data obtained from the network. For each outgoing variable, no update function is necessary, but a function that checks the share memory for new value is required, as well as a function that polls the network to send that value.

#### **II.3.2.3.4. ProcessData Module**

ProcessData is responsible for the translation of raw data coming from the network into converted engineered data. Because of the limitations of the Neuron Chips capabilities, and in order to optimize the computation and data throughput of the nodes, only integer values are manipulated on the network. Therefore, a data is often sent in several parts, in formats that are not directly usable. Moreover, processing of multiple oncoming pieces of data is sometimes required to derive other information. ProcessData is dedicated to

handling these tasks. On a regular basis, it checks the shared memory for update of raw sensor data. When an update occurs, it performs the required conversion and processing, and writes the results in separate shared memory variables.

#### **II.3.2.3.5. Logger**

The logger is the process responsible for regularly checking the shared memory and logging selected variables in a file, for debugging and post processing. The variables to be logged are listed in the text file lgr.in, along with various information about how the variables are to be logged (Figure 18). Options are provided in terms of logging frequency of a variable, and conditions for logging such as “Always”, “Once”, “If Updated”, “If Changed”, and so on.

```
auvHeading 8 IfChanged;  
AuvDepth.depth 8 IfChanged;  
AuvDepth.depthRate 8 IfChanged;  
AuvAlt.altitude 4 IfChanged;  
AuvAlt.altitudeRate 1 IfChanged;  
auvSpeed 4 IfChanged;  
auvRpm 1 IfChanged;  
auvTorque 1 IfChanged;
```

*Figure 18: Lgr.in Content Example*

The process is scheduled using a timer. Each times it executes, it scans the shared memory for the desired variables. If the condition for logging is satisfied (frequency, update,...), the variable is copied in the log file.

#### **II.3.2.4. Tools**

Various tools are used with the AUV, which can be classified as development tools, monitoring or checking tools, and pre and post-mission processing tools.

#### **II.3.2.4.1. Development Tools**

The two most used tools for the development are LonWorks tools, namely NodeBuilder and LonMaker. The NodeBuilder Development Tool is available from Echelon, and according to them, “provides an integrated hardware and software development environment running on an PC-compatible host computer. The tool includes components required for the rapid development of LonWorks devices” [45]. NodeBuilder is a device-level development tool that allows to program and debug individual LonWorks devices. It mainly consists of a wizard that assists developers in defining devices templates, a text editor for creating Neuron C source files, a cross-compiler for creating Neuron Chip object code, a linker to build final applications from object code and various libraries, and finally a cross-debugger that helps developers debugging an application running on a connected Neuron device.

As far as the modification of the AUV is concerned, we mainly use NodeBuilder to compile the Neuron C code that is the source of Neuron device application. Then it builds the application so that it can be loaded in a specific type of node defined by its template. A template essentially lists the various memory segments, along with their size and address, as well as other information such as neuron processor used, clock frequency, type of channel and transceiver.

The LonMaker Integration Tool is also available from Echelon, and according to them, “is a multipurpose LonWorks network engineering tool that runs on a PC under the Windows OS and uses Visio as a graphical interface” [50]. It essentially consists of:

- A network design tool that allows the design of a network, with or without being connected to it.

- A network installation tool that allows an off-site designed network to be installed. The engineered device definitions are associated with their corresponding physical devices, which can then be managed in various ways (load, reset, browse...).
- A network documentation tool. The Visio based graphical interface creates a drawing that accurately represents the network, and can thus be used for documentation.
- A network maintenance tool that allows the devices, routers, channels or subsystems to be modified, replaced and tested to support system maintenance.

Using the graphical interface, we add nodes to the network. If the computer is actually connected to the network, it is then possible to load an application into the device (provided that the memory segment for the application is Flash or EEPROM-based). Then the device can be “winked” in order to check the communication. A few other test methods are available. Adding a functional block related to a device, other management possibilities are offered, especially to browse in real time the network variables defined for that device, which is very useful during the development phase, particularly in terms of timing debugging. Many other features available on NodeBuilder and LonMaker are not described here. The reader interested in detailed explanations related to these tools should refer to Echelon documentation such as [43], [45], [50]. Various other tools are necessary, among which many of them do not need to be named here. Nevertheless, a few other tools are mandatory, namely the WSI compiler, the BP PROM burner and the Watcom C compiler. The WSI compiler is used to translate or “compile” an output file of NodeBuilder into a bitstream file that can be transmitted to a Programmable Logic Device (PLD). In our case, we need it for programming the UV(E)PROM used on PSG10 Neuron devices. The BP prom burner is used with the above to actually program the PROM of a

PSG10 with a bitstream. It drives a PROM burner, which is basically a mounting adapter for the PROM, and a logical circuitry that places the PROM in programming mode, and counts the address to which each byte is to be written. The Watcom C QNX-based compiler is used to compile and build the C language Main Control software. Not much knowledge about this tool is required for common development, since the OEX-C/Morpheus software comes with a number of makefiles that help the developer building the application.

#### **II.3.2.4.2. Monitoring Tools**

The monitoring tools mostly used for development and pre-mission check are the Monitor, the LonDaemonDiag, MissionCheck and ReadyMission tools.

The Monitor client is a tool written in Python programming language, that enables a computer connected to the AUV through TCP/IP sockets to browse and edit the content of any shared memory variable. On the vehicle side, a monServer daemon attached to the shared memory handles the request coming through TCP/IP for browsing or updating a particular variable. On any computer, a TCP/IP client connects to the monServer and, through a menu-based interface, allows the operator to browse variables previously listed in text files.

The LonDaemonDiag utility, developed by the AUV Laboratory, provides network management and diagnostic possibilities. It is related to the LonDaemon processes, which handle the LonTalk communications between the main computer and the network. LonDaemonDiag, as a diagnostic tool, allows the operator to list and browse the nodes connected on the network, to display and edit the NVs, and perform basic operation such

as resetting a node. As a management tool, it allows the developer to bind the network, that is to logically connect each NV to its writers and readers nodes. For more details about the LonDaemonDiag tool, refer to [41].

The MissionCheck tool is a utility that performs various tests on the vehicle. It exercises and monitors all the actuators and sensors of the vehicle. The following actions are performed:

- Check for errors flagged by any node on the network.
- Check the health monitoring devices and report summarized informations. If the health information for a node is not updated a warning is issued.
- Test each actuator, by sending commands and checking their executions. An error is reported whenever a command is not satisfactorily executed.
- Test each sensor, and report the most recent readings.
- Test the logger. Scan the lgr.in file and compare it to the shared memory. Return a warning in case of syntax error, and return the number of variables to be logged.

The ReadyMission utility simulates the mission currently defined in the mis\_plan input file. The file is parsed and corresponding navigation commands are generated. Then the response and position of the vehicle are estimated. The results are stored for post-processing, and any errors are logged. The output generated by this utility is analyzed with a Matlab script that displays important data, allowing the developer to check that the mission will be performed as expected.

#### **II.3.2.4.3. Processing Tools**

A certain number of processing tools have been developed, ranging from fairly simple, such as FTP macros, to much complex, such as post-mission data extraction. A few FTP macros have been developed to simplify common FTP operations. Among others, these macros are used to send mission files to the vehicle, get output files or reports generated by pre-mission checks, and so on. Various Matlab scripts exist to perform common data display and analysis on the data extracted from the vehicle logger. The extract tool extracts selected variables from a mission logger file using various available options. For post-processing, the operator can extract any combination of variables history. For commonly extracted data sets, scripts are written that call the extract utility with the required parameters. In addition to the data, headers with variables name and unit are also created. For more details about the extract tool, refer to [41].



## **III. Upgrading the Ocean Explorer**

With the knowledge summarized in the previous chapter, the Ocean Explorer D was built as an upgrade of the existing vehicles. Once again, the goal was to restore an existing OEX-B and improve it by addition of the more convenient high-level programming and operation interface used on both the OEX-C and the Morpheus. This choice makes all FAU vehicles have the same Operating System (OS), facilitating common operating procedures. Specific steps to reach this goal were:

- Implementation of a new computer on an OEX-B,
- Installation of the OEX-C/Morpheus main computer OS and software,
- Interfacing of the main computer and distributed nodes softwares,
- Tests and various fixes.

Each of these steps is summarized in this chapter.

### ***III.1. Implementation of a New Computer***

As explained before, the main computer software we wanted to install runs on the QNX RTOS, available for PC platforms. Therefore, the Motorola-based computer had to be replaced by one which was PC compatible. Moreover, the choice was dictated by the need to reduce the power requirements. Finally, it is better to use a proven technology on which the software to run has already been tested. For these reasons, the same computer

as on the OEX-C/Morpheus, a Pentium-based computer in a PC104 form factor, was selected in order to take advantage of the PC104 architecture, the peripherals were chosen in that form factor whenever possible. The essential peripherals are: hard-drive, MIP LonTalk adapter, and Ethernet controller. Moreover, for first step developments, it is desirable to have a console (keyboard and display) directly connected to the computer, as well as a floppy drive. The chosen parts are listed in Table 2:

<i>Component</i>	<i>Manufacturer</i>	<i>Reference</i>	<i>Comments</i>
Motherboard	Electronic Equipment Production & Distribution GmbH (EPPD)	CPUT6VEF	PC104, 133MHz Pentium MMX, LCD/VGA display controller, 64Mb SDRAM, Intel 82559 fast Ethernet 10/100BaseT controller, hard-drive and floppy controllers.
Hard Drive	Toshiba	MK3017GAP	30Gb, 2.5in, ATA interface.
LonTalk MIP interface	IEC Intelligent Technologies	MIP-P50 PC104 Parallel Adapter	
Miscellaneous	EPPD GmbH	Cable kit for PC104	Cables and connectors for VGA, keyboard, floppy, and so on.

*Table 2: Main Computer Components*

The main computer itself was easily assembled, since the PC104 cards stack one on top of the other, using the PC104 bus for communication. The hard-drive, being stand-alone, was placed on top of the stack, attached to a PC104-size mounting board. The LonTalk MIP board was configured so that its base address is set to  $(360)_{16}$  (360 hexadecimal), and it uses the interruption number 5. Every peripheral I/O was wired to a standard connector accessible from outside the stack for convenience.

The computer then had to be integrated in the main pressure vessel, both mechanically and electrically. As described in II.3.1.1.2, that pressure vessel mainly consists of a VME backplane. A new PC104/VME adapter was designed by the Electronics Laboratory. It

consists of a standard VME card on which the whole stack, as well as its I/O connectors, are attached. This adapter also provides the computer with power and LonTalk, which are the only lines it has to share with the backplane. The integration of the new computer in the main pressure vessel was completed by connecting its Ethernet port to the embedded 4-channel 10BaseT hub.

Figure 19 summarizes the integration of the new computer stack in the main pressure vessel.

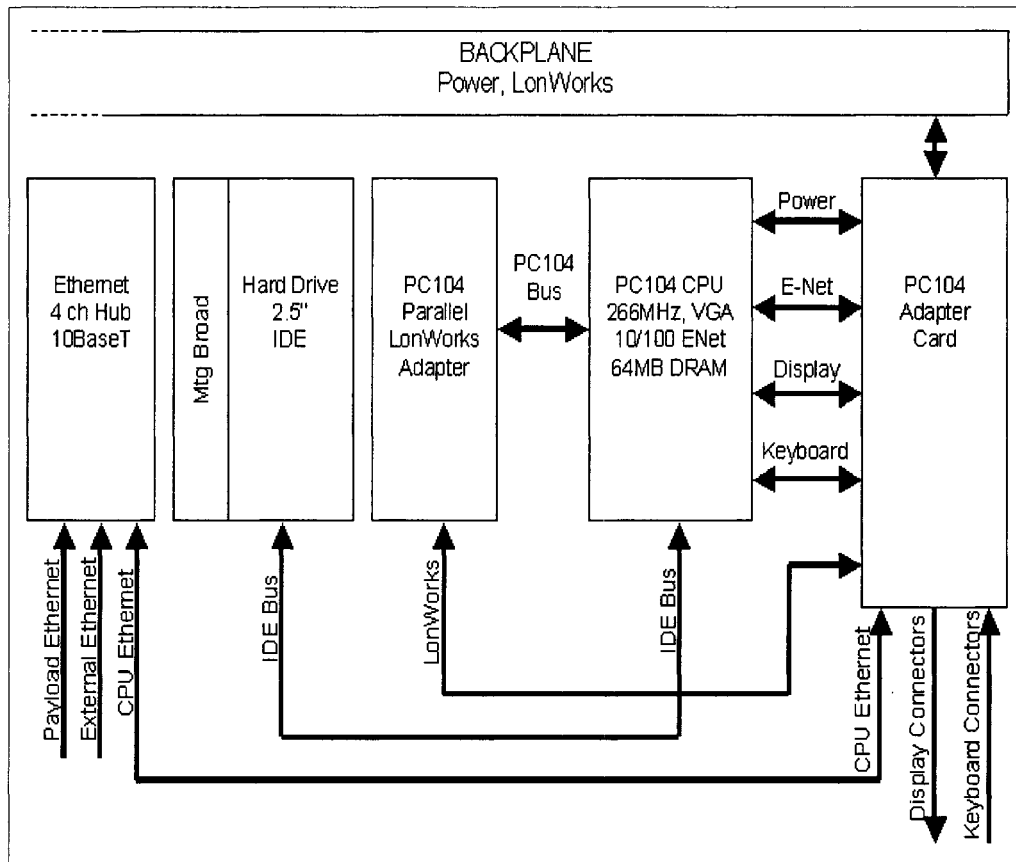


Figure 19: Integration of the New Main Computer Stack

## ***III.2. Operating System and Software Modifications***

This part involved installation of the QNX OS and the OEX-C high-level software, and the task of interfacing between the main computer and the distributed nodes softwares.

### **III.2.1. Operating System and High-Level Software Installation**

Before installing the OS, we first configured the Basic Input Output System (BIOS) of the computer: we mainly disabled the Plug-and-Play (PnP) controller, set various parameters, and configured the Hard Drive (HD) controller. The HD we mounted can be used in two addressing modes: either classical geometric addressing, or Logical Block Addressing (LBA) mode. The geometric addressing mode, which is the only one available with the selected version of the OS, was chosen. This allows the HD to be configured to a 8.4Gb capacity. The BIOS was configured for that mode. We installed the OS, formatted the HD to create a primary bootable QNX partition, and mounted it as the filesystem root. Then we installed the QNX4.25 patch D that fixes several bugs, and a new Ethernet controller driver for the integrated Intel 82559 fast Ethernet chip.

To enable FTP, TELNET and RLOGIN access, we installed the TCP/IP 4.23 support, and set the Hostname and Internet Protocol (IP) address for the computer. Various definition files were modified to properly configure the TCP/IP package. This mainly consisted of the configuration of the host name resolver, settings for the Domain Name Servers (DNSs) and gateways, and various restrictions for distant access.

Next, we added a few tools such as Sysmon (a system resources monitor), the tcsh shell, the Watcom C compiler and libraries, and the Mkdep utility (that forces rescanning of all files dependencies when building a project code). We also installed and configured the

driver for the LonWorks MIP adapter.

Finally, we changed the system configuration and initialization, to automatically load the Network, Pipe and Queue managers, and start the TCP/IP socket processes upon startup. We also changed the Process Manager configuration in order to enable the use of up to 600 semaphores. It was thus necessary to rebuild the kernel.

The OS was then ready for the installation of the software. We created a directory structure with three directories for the AUV software, one for the executable code, one for the OEX-C software source, and one where the source would be moved as it is integrated. We then copied the software from the OEX-C repository. We finally modified the system initialization so as to perform a vehicle initialization whenever the computer starts. This mainly launches the required daemon processes, to have the AUV ready to start a mission every time the computer is powered up.

### **III.2.2. Software Integration**

Then, we had, physically connected together, all the Neuron nodes with their application, and the new main computer with its software. We had to integrate the whole software together. This integration step is relatively complex to describe, for it was repetitive, and required numerous tests and fixes after a few modifications were done. Therefore, a basic description of the integration task is given, followed by a thorough, yet simple, example of the interfacing of a Neuron node. Then a few more complex tasks are described based on another specific example. Finally, some other modifications undertaken to bring some improvements to the system are summarized.

### **III.2.2.1. General Method**

The following method was used for the software integration: First of all, we identified all the nodes on the network, and tried to communicate with them. Then we planned the Network Variables (NVs) interface so that the declarations on each side (all nodes and main computer) matched. Once this was done, the interface was coded: First the Neuron nodes softwares were modified to define the correct NVs, and configure each node in a state suitable for automatic binding with LonDaemonDiag. Then the interface was defined on the main computer side, through modification of the LonDaemonNv NV table and associated functions, and the declaration of the corresponding shared memory variables. When necessary, a function to convert the raw data coming from the network was added to the processData process. In such a case, the additional shared memory variables to store the converted data were defined. Then, the network was bound, defining the binding information table, and using the LonDaemonDiag tool. Finally, the shared memory variables created or modified were reviewed in the logger input file and monitor variables lists to ensure that the correct name was used for each variable. Throughout this integration process, many tests were performed whenever a single modification was completed.

The software integration was done in several steps. First, we tried to have the whole main pressure vessel working. After that, we added the DVL, GPS antenna and dropweight, then the batteries and control box, and finally, the thruster and fins.

### III.2.2.2. An Example of Software Integration Tasks: Integration of the GPS

To explain the software integration tasks, the integration of the GPS is given as an example. This node is chosen because it is relatively simple to implement. The GPS essentially communicates with the main computer to provide GPS fix information.

The GPS node on the OEX embeds an application that mainly parses the GPS data coming through a serial port, and sets accordingly several fields of a network variable. Moreover, the application listens to the network for update of a command NV that tells the node to perform a few specific operations such as changing the configuration of the GPS or polling a data set. On the other side, the main computer needs to get the raw data from the GPS and convert it into a convenient format. It also has to set the required command codes to control the GPS. This data flow is summarized in Figure 20.

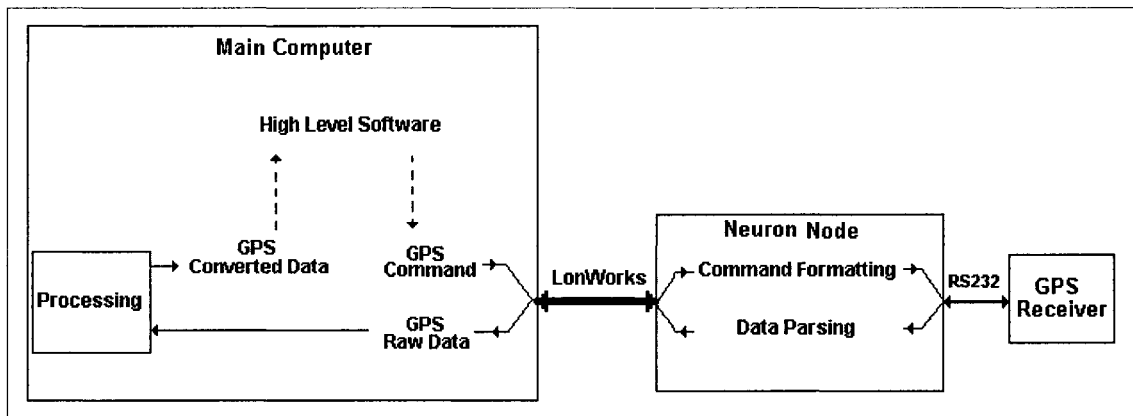


Figure 20: Simplified GPS Data Flow Chart

We had to bring together the interface between both sides. The first step was to identify the node on the network. We located the GPS node board in the VME crate and checked its connections. Then we identified the type of Neuron device, in our case a PSG10, and located its service pin, a switch that makes the Neuron node send an identification

message on the network. Using the LonMaker tool on a development computer attached to the network, a new device was added on the network drawing. Its properties were set to the best of our knowledge (type of node, transceiver, and so on). When we depressed the service pin, LonMaker identified various properties of the node, especially the Neuron Identification (NID), which uniquely identifies the node on the network. Then the second step was to clearly plan the variable interface. The network variables needed are:

- A multi-fields (structure) NV sent by the GPS to the main computer to communicate the raw GPS fix data. This NV is also sent to the control box node so that it is able to display the GPS status (GPS, DGPS or nothing at all) through a defined LED pattern.
- A simple (integer) code issued by the main computer to command the GPS to the various configurations it can be set.
- Moreover, another simple variable is used by the main computer for controlling the antenna motor to rise/lower the antenna. This NV is also monitored by the GPS so that it knows when the antenna is supposed to be up or down, indicating when GPS signals are likely to be received.

On the main computer side, these NVs are to be matched with shared memory variables. Two “command” and one “raw incoming data” variables are required. Moreover, it is necessary to process the raw GPS data and convert it into a convenient format. This is done by ProcessData, which requires another shared memory variable to store the converted data.

All the variables were named according to the naming conventions in use in the OEX-C/Morpheus software. The overall variable needs and logical connections are summarized in Figure 21.



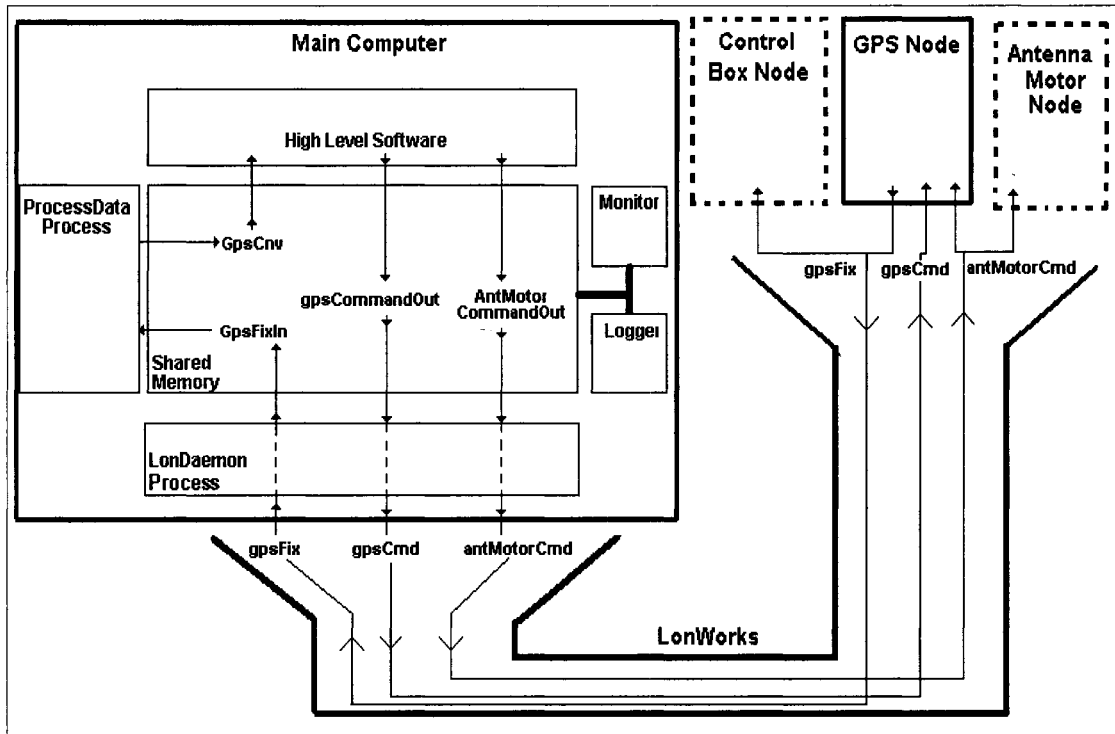


Figure 21: GPS Variables and Logical Connections Synoptic Diagram

Once the integration task was thoroughly planned, we modified the node software. First, we had to allow the automatic binding of the network using the LonDaemonDiag tool. This was done through the addition of two preprocessor directives (or “pragmas”) in the source file:

- #pragma set\_node\_sd\_string "GPS"
- #pragma enable\_sd\_nv\_names

The first pragma allows the node to broadcast a self-documentation (sd) string on the network, that can be used instead of the NID to identify the node by its name. The second enables the node to provide the name of the NVs in addition to their identification code. Then, the NV interface was redefined according to our plan (Table 3).

<i>NV</i>		<i>Comments</i>
<i>Type</i>	<i>Name</i>	
Network output GPS_FIX <sup>(1)</sup>	gpsFix	Output GPS data (latitude, longitude,...)
Network input long integer	gpsCmd	Set mode or request data form GPS
Network input long integer	antMotorCmd	Indicate when the antenna is risen/lowered

*Table 3: Network Variables Interface for the GPS Node*

(1) *GPS\_FIX* is a custom defined type. It consists of a structure of 14 integers to store each piece of data forming a GPS fix.

Then we went through the node source code to verify it, and especially check the consistency of the constants or codes that are supposed to be defined in other files. Once everything had been verified, the source code was recompiled and the application was built, both using NodeBuilder. Then, using the WSI compiler and the BP PROM burner, a new UV PROM was burnt and placed in the Neuron device. The device was replaced in the LonMaker drawing so that the tool takes the Neuron node modifications into account, and the node was rapidly tested. Finally, using LonDaemonDiag, we checked that the node was seen on the network from the main computer side, and was correctly identified by both its NID and self-documentation name. We also checked that the NVs were correctly seen by the main computer.

The next step was to logically connect the GPS node with the other nodes it has to communicate with. Here, as described previously, the node principally communicates with the main computer host application for commands and data exchange. It also somehow communicates with the antenna motor node, since both devices receive the command to rise the antenna when a GPS fix is to be taken. The GPS node also communicates with the control box node, since this one monitors the GPS fix NV in order to display the GPS status. The logical connections are defined in the

BINDINFO.TBL table (Table 4), in which each variable is listed, with its readers and writers. This binding information table file is used as an input for the LonDaemonDiag utility when it performs the binding.

<i>NV Name</i>	<i>Writer(s)</i>	<i>Reader(s)</i>
gpsFix	GPS	Host application, control box
gpsCmd	Host application	GPS
antMotorCmd	Host application	Antenna motor, GPS

Table 4: Binding Information Table for GPS Network Variables

Then we had to write/check the interface of the antenna motor, the control box and host application. For the antenna motor and control box, it was just necessary to check that the incoming NVs antMotorCmd and gpsFix were correctly defined, with the same names and types, and used as expected. Especially, we checked that both the GPS and antenna motor nodes had the same convention for the codes used to rise or lower the antenna.

Then came the interface with the host application. The first thing to do was to enable the main computer to gain access to the GPS NVs through LonDaemon. In the LondaemonNv process, the NVs table (Table 5) was modified to declare these GPS NVs, specify various properties and list the addresses of the functions used to check, update and poll each variable.

<i>{“NV name”, Update function,</i>	<i>Size,</i>	<i>Direction,</i>	<i>Acknowledgment, Check function},</i>
{"gpsFix", LdNvUpdateGpsFix,	28, LdNvPollGpsFix,	NV_IN,	UNACKD, NULL },
{"gpsCmd", NULL,	2, LdNvPollGpsCommand,	NV_OUT,	ACKD, LdNvCheckGpsCommand},
{"antMotorCmd", NULL,	2, LdNvPollAntMotorCmd,	NV_OUT,	UNACKD, LdNvCheckAntMotorCmd},

Table 5: LonDaemonNv Table Section for GPS Network Variables

The corresponding functions were then declared and written, which is not detailed here. To summarize, these functions check for update of NVs or shared memory variables, perform the required format manipulations (swapping Least and Most Significant Bytes (LSB/MSB)) and synchronize copies of variables between network and shared memory. The corresponding shared memory variables were created to store the copy of each NV (Table 6).

<i>Name</i>	<i>Unit</i>	<i>Type</i>	<i>Initialization</i>	<i>Comment</i>	<i>Corresponding NV</i>
GpsFixIn	n/a <sup>(1)</sup>	n/a <sup>(1)</sup>	n/a <sup>(1)</sup>	GPS data	gpsFix
gpsCommandOut	“n/a” <sup>(2)</sup>	integer	0	0: Off, 1: On, 2: Fix, 3: Idle ...	gpsCmd
AntMotorCommandOut	“n/a” <sup>(2)</sup>	integer	0	0: Nothing, 1: Down, 2: Up.	AntMotorCmd

*Table 6: Shared Memory Variables Declaration for raw GPS Variables*

(1) The variable *GpsFixIn* is a structure of 14 fields, not detailed here. For that reason, the unit, type and initialization value are reported here as *n/a*, although these are set for each field of the structure in the software.

(2) Here, “*n/a*” means that there really is no unit for these variables, and that is the information that is set in the software.

At that point, the NVs were interfaced with the shared memory. To send a command to the GPS, it is only necessary to write the required code value in the shared memory variable. But it was not that simple for the data coming from the GPS. GPS-fix data is written in shared memory using a format that is convenient for the network communication, but not for direct use in the main computer software. It is received in several pieces giving the latitude and longitude in degrees, minutes and fractions of minutes, and so on. Moreover, all variables are integers only. In order to have this

information represented in a convenient format, it is necessary to process the data and perform the required conversion. This task, among others, is handled by `processData`, in which a piece of code was written so as to perform the tasks summarized hereafter (Figure 22). This also gives an example of the kind of conversion performed in `processData`.

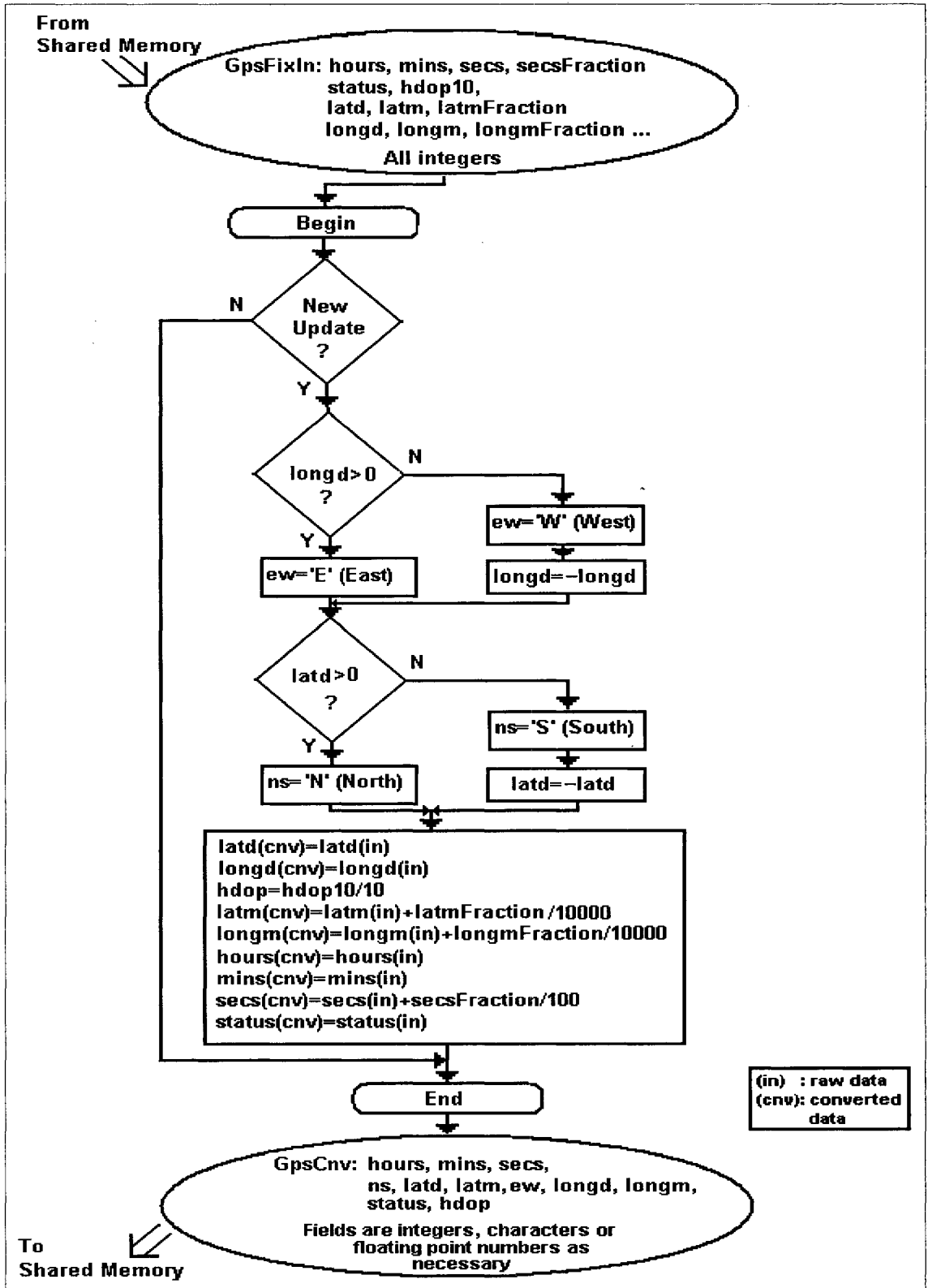


Figure 22: GPS Data Processing and Conversion Algorithm Summary

The corresponding shared memory variable where the processed data is to be written was then declared in the shared memory definition (Table 7).

<i>Name</i>	<i>Unit</i>	<i>Type</i>	<i>Initialization</i>	<i>Comment</i>
GpsCnv	n/a <sup>(1)</sup>	n/a <sup>(1)</sup>	n/a <sup>(1)</sup>	Converted GPS data

*Table 7: Shared Memory Variables Declaration for Converted GPS Variables*

(1) The variable *GpsCnv* is a structure of 11 fields, not detailed here. For that reason, the unit, type and initialization value are reported here as n/a, although these are set for each field of the structure in the software.

The integration of the GPS was then finished. It was just necessary to check that every process that asks for an access to the GPS variables in shared memory is looking for the right variables and is not using different names. Finally, the names of the shared memory variables were modified or added in the logger input file (Table 8).

<i>Shared Memory Variable Name</i>	<i>Log Frequency</i>	<i>Log Condition</i>	<i>Comment</i>
GpsFixIn	1 Hz	Change	Most fields are not actually logged <sup>(1)(2)</sup>
gpsCommandOut	1 Hz	Change	Not actually logged <sup>(1)</sup>
AntMotorCommandOut	1 Hz	Change	Not actually logged <sup>(1)</sup>
GpsCnv	8 Hz	Update	Some fields are not actually logged <sup>(1)(2)</sup>

*Table 8: Logger Input File Section for GPS Variables*

(1) Usually, all variables are listed in the logger input file. Once tested, the variables that are not essential to log are commented but kept in the file so that any required modification of the logger configuration is easier.

(2) For complex variables made of several fields, each field is considered as a single variable. Therefore, some fields can be logged while some other not, or the logging frequencies and conditions can be different among the fields of a same structure.

The integration was approximately as described above for most of the nodes. Nevertheless, the integration of some nodes was more difficult either because:

- The data received from some sensors require more complex processing. That is the

case for instance for the DVL, whose data are processed with that describing the AUV attitude and that coming from the CTD, to provide reliable information to the navigator. In such cases, the data coming from the network is interfaced with the shared memory through LonDaemon as usual, but processData makes a more complex processing, using several shared memory variables, to convert the data to a convenient and meaningful representation.

- The nodes used on the OEX-D are significantly different from what is used on the OEX-C, therefore, more software rewrites were required. That was the case for instance for the batteries, which are using different nodes and are managed differently on each version of the OEX, or the control box, which doesn't exist on the OEX-C. In such cases, some modifications of the nodes applications were required, and the integration with the high-level software was a little more complex.

### **III.2.2.3. A More Thorough Example**

A more thorough example can be given with the integration of the batteries, for which more software modifications were required. The main reason is that, where the OEX-D (as well as the OEX-C and Morpheus) now uses the LonDaemonDiag utility to bind the network, that of the OEX-B was bound using the LonBuilder tool. Because binding the network with LonBuilder was a time consuming process, the binding was usually not modified on the fly. Nevertheless, it was necessary to be able to add or exchange batteries depending on specific mission needs. But it was, and is still, necessary that a node, the control box for instance, be able to communicate with any battery on the network. In order not to have to rebind the whole network every time batteries are exchanged or



added, communication had to be done without having to a priori know the number of batteries and each battery Neuron ID (NID). To do so, explicit messaging as well as some other features not described here were used. Concerning the OEX-D, since explicit messaging is complex and not managed by LonDaemonDiag, it was decided to use classical NVs instead. Doing so, when batteries are exchanged or added, it is necessary to bind the network again, which only takes a couple of minutes with LonDaemonDiag.

Thus it was necessary to modify the application code of the batteries as well as the control box and rewrite the appropriate software interface on the main computer side. All explicit messages were removed and replaced by one simple variable to turn on or off all the batteries, and two variables per battery to report battery information and health. When the vehicle is turned on through the control box, the battery command NV is set to ON, and then each battery can decide to turn on or off depending on its state and that of the bus.

The other difficulty was that at least eight batteries are to be used on the vehicle. Every battery runs the same application, but needs to be uniquely identified so that the main computer is able to store each battery information. It is impossible to use the neuron self-documentation name, since this attribute, defined in the neuron application, is the same for every battery. But when the application of a node is loaded, it is possible to set an attribute called "location" identifier, which is then specific to an association node-application. It can thus be used to identify a specific battery. The neuron code was modified so that each node application retrieves its location ID, and uses it in network communication. Battery information and battery health NVs, that have to be specific to a single battery, were modified so as to be some structures in which one of the fields is a copy of that location ID. This method is summarized in Figure 23.

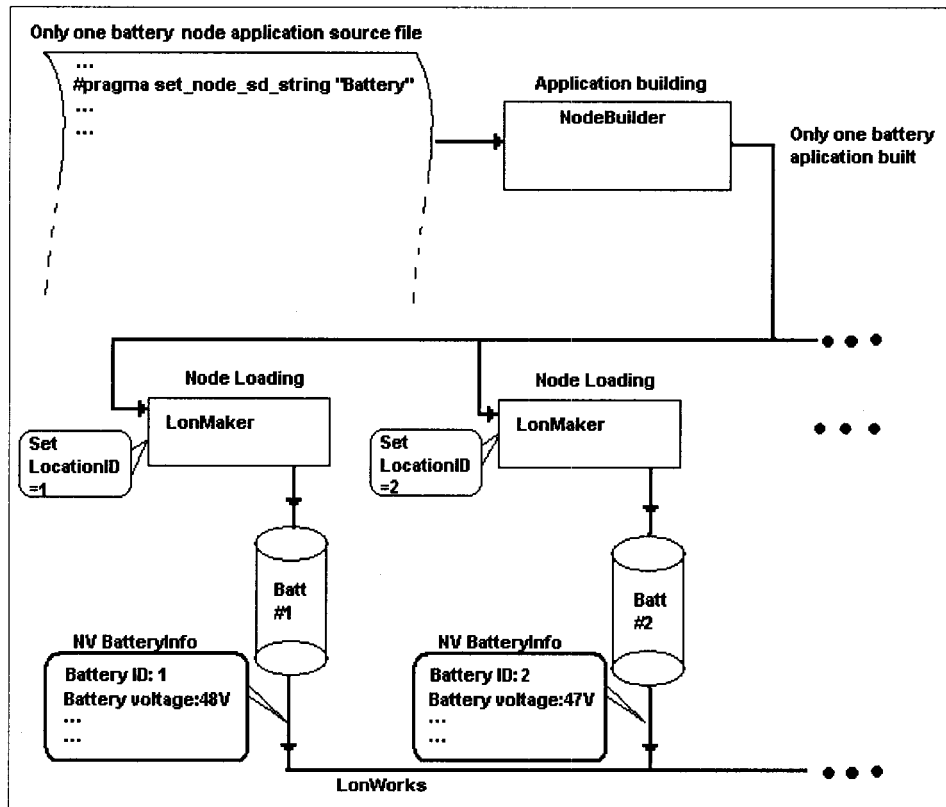


Figure 23: Multiple Batteries with one Single Application

Once the neuron software was ready, and the required modifications had been done in the control box node (that communicates with the batteries), the batteries were interfaced with the main computer using the same procedure as described in the previous example of the GPS. The batteries NVs were added in the BINDINFO.TBL table, each NV was added in the lonDaemonNv table, along with the corresponding functions, the shared memory variables for raw data were created, raw data conversion functions were reviewed in processData, and the shared memory variables to store the conversion results were created.

Once the batteries had been integrated with the vehicle software itself, a few tools had to be modified. The most significant modification was made in the missionCheck tool. In

this utility, run before every mission, a procedure checks the state of each battery. MissionCheck was supposed, on the OEX-C and Morpheus, to display for each battery its location ID, state, voltage, current going out, current coming in during charging, total energy available, used, and remaining. But on the OEX-D, the batteries are not equipped to measure the current coming in during charging. As it is impossible to display that value, the corresponding field was removed from the display. But moreover, without measuring the current input, the energy information field cannot be updated during charging, which is critical. Each battery itself reports its energy available and used based on the current output history since last reset. But it was necessary to take into account the charging. The missionCheck utility was modified so that each time it is run, it checks for batteries reporting a “FULL” state, for which the energy used field is non-zero. For each such battery, it asks the operator if he wants to reset the energy of that battery. In case the answer is yes, that information has to be sent to the corresponding battery. A new NV was then added to send that command. The NV has a field indicating which battery is the target of the command. The neuron code for the batteries was modified so that when an update occurs on the reset energy NV, the node compares the target ID in the NV with its own ID, and if they match, it resets its energy used to zero, and energy available to a predefined constant. This NV also had to be integrated in the shared memory, lonDaemonNv and BINDINFO.TBL in the manner previously described for all other NVs. This battery energy gauge reset procedure is summarized in Figure 24.

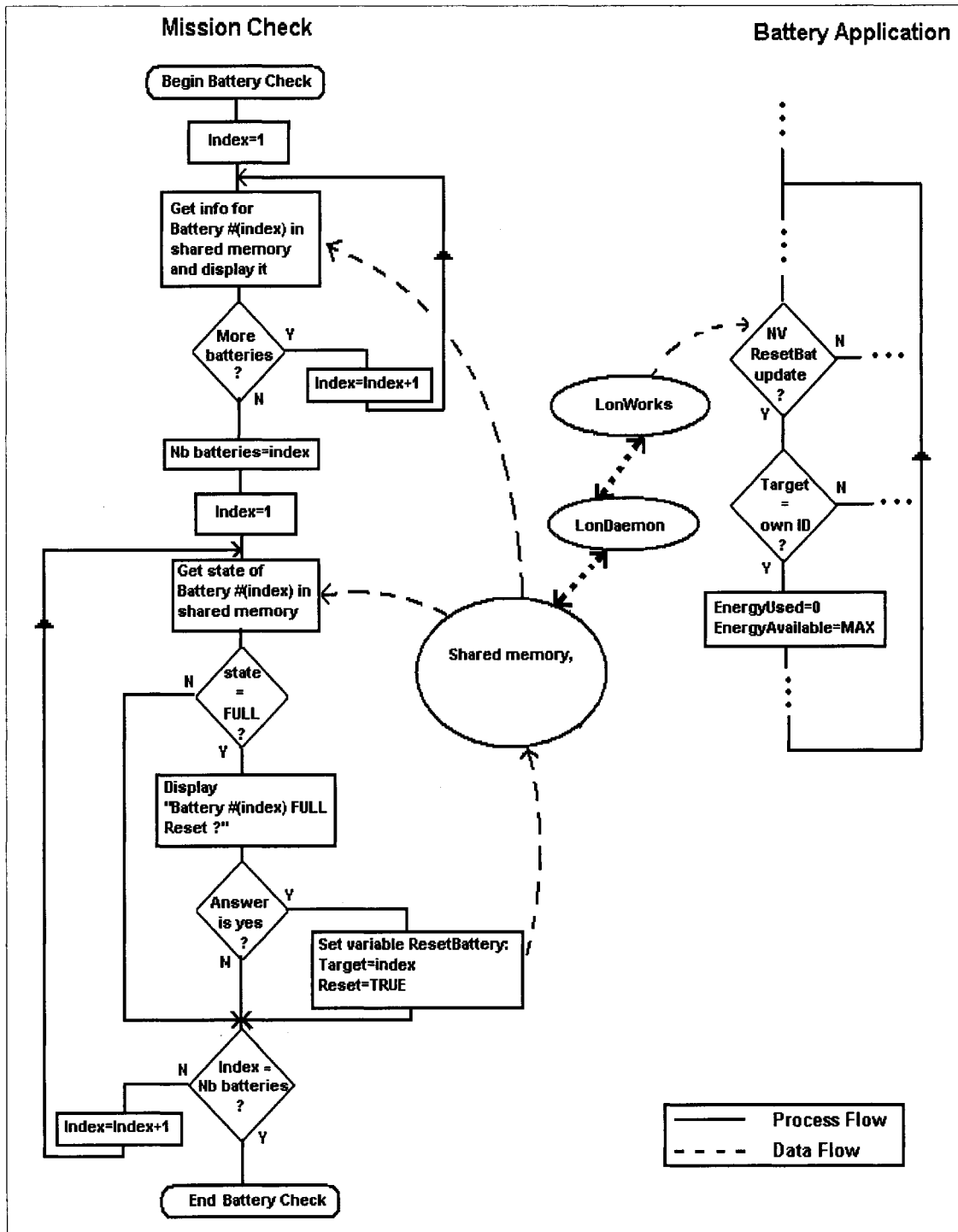


Figure 24: Batteries Energy Gauge Reset from MissionCheck

As usual, the variables list in the input files for the logger and monitor were modified to match the newly defined variables. That completed the integration of the batteries.

#### **III.2.2.4. Other Software Modifications**

Numerous other tasks specific to a node or a piece of code were undertaken, but their description is beyond the scope of this chapter. The purpose here is only to give an overview of the methods used. In addition to the integration task described so far, a few improvements or fixes of the software were performed when, while interfacing some pieces of code, some bugs or ways of improvements were discovered. In some cases, these modifications were also added to the Morpheus software.

One of the major improvements is that added to perform a clean shutdown of the main computer. The way it worked before on each of the OEX and Morpheus was that whenever the on/off switch on the control box or console was activated, two NVs were set. The first one was directed towards the MainHealth so that it turned off the power to the main computer. After a delay, the second one, which on the OEX-B was actually an explicit message as explained before, was sent towards the batteries to turn them off. The problem with that procedure is that it performed a hard power shutdown on the main computer. The computer was thus unable to terminate the current processes, close the open files and so on. This was likely to create some errors on the hard drive, mainly on the files that were being modified. A simple, yet significant improvement was brought to that procedure: the NV sent to the MainHealth to turn off the computer power supply is now also sent to the main computer. When it receives the NV update indicating that the power is to be turned off, a software procedure performs a clean system shutdown, enabling the computer to properly terminate the current processes and close all open files. In the MainHealth node application, a delay was added between the reception of the NV update and the actual power down procedure. Doing so, the computer is properly shut

down before the power is turned off. Since this improvement is simple to implement and is thought to bring a significant enhancement to the long-term reliability of the main computer, it has also been implemented on the Morpheus.

Another improvement in the OEX-D software was the implementation of the Virtual RS232 (VRS232). A similar functionality was available on the OEX-B software but it has now been standardized according to the VRS232 used on both OEX-C and Morpheus. It allows an operator logged on the main computer to communicate through LonWorks with sensors having a serial interface, as if the sensors were directly connected to some of the serial ports on the computer. This is mainly used to test sensors, or configure them using specific settings not implemented in the neuron node applications. For instance, this is used every time the compass has to be calibrated: the command to begin/end the internal calibration is sent through VRS232.

Once the software integration is completed, the AUV can be seen as the distributed control network given in Figure 25.

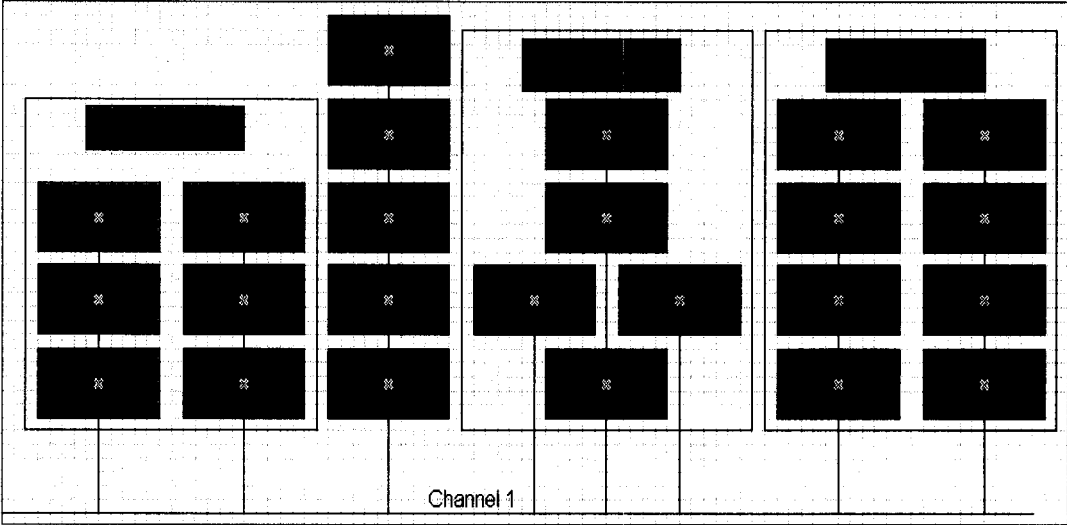


Figure 25: OEX-D Distributed Control Network

Then the vehicle had to undergo detailed tests to ensure its working and identify the problems that needed to be fixed in order to have a vehicle ready for real missions.

### ***III.3. Tests***

Numerous laboratory tests were performed during the development of the OEX-D, but are not reported here. Once the vehicle was deemed to be ready as far as possible to tell from laboratory tests, a major pool test was conducted, followed by the first at-sea test.

#### **III.3.1. Pool Test**

The objectives of the pool test were first to check the health of the system, then exercise all sensors and actuators in water, not only to check that everything was working satisfactorily, but also to derive some information about the functioning, such as power consumption of some subsystems, or the time required for certain operations. To do so, three missions were run.

##### **III.3.1.1. Mission Planning**

The purpose of the first mission was to log various data while the vehicle was idle. The mission was then written so that the vehicle did nothing during 10 minutes. The purpose of the second mission was to exercise the fins. The mission was written so as to successively command the rudders and sternplanes to various setpoints. The third mission aimed at exercising the thruster. The mission was written so as to successively command the thruster from 10% to 100% of the maximum rpm. For all three missions, the logger was configured to log 438 variables.

### **III.3.1.2. Mission Execution**

On October 14<sup>th</sup>, 2002, these missions were run in the pool at Seatech. Because three of the batteries had problems and were under investigation, only five batteries were available for that test. A missionCheck was run to test the whole vehicle, except the acoustic modem (TSAM) because the vehicle wasn't yet in the water. Everything was working, and a DGPS fix was reported in about 20 seconds. The vehicle was then lowered in the water and tighten to the pool walls, while, using a monitor client, we checked the health variables for indication of leak. Once the vehicle was in the water, a second full missionCheck was run, that reported a 7% error on the thruster rpm. The three missions were successively started through TSAM. During the whole test, we kept monitoring the health variables, in order to be able to abort the mission and take the vehicle out of the water should a problem occur.

### **III.3.1.3. Mission Analysis**

The missions output files (consoles, and loggers) were analyzed, according to the purpose of each mission. Various analyses were performed, of which only the significant results are summarized here.

#### **III.3.1.3.1. Mission 1**

The console didn't report anything unusual. The health variables were extracted from the logger file, and are summarized in Table 9.



<i>Variable</i>	<i>Value</i>	<i>Comment</i>
Leak	4087-4096 counts	Maximum value, no leak
Humidity	0-8 %	Where measured
Temperature	25-30 °C	Most nodes <sup>(1)</sup>
	35-40 °C	Control box, dropweight and main pressure vessels <sup>(1)</sup>
Bus voltage	58-55 V	Slowly decreasing
Bus current	<80 mA	Most nodes
	600 mA	Main pressure vessel
	1.3 – 30 A	Antenna motor <sup>(2)</sup>
Battery voltage	56-57 V	Except battery #1: 0 V <sup>(3)</sup>
Battery current	300 mA x 4 batteries	Battery #1 off <sup>(3)</sup>
Error	0	None

*Table 9: Vehicle Health Data Summary for Mission 1 on 10/14/02*

*(1) The control box, dropweight and main pressure vessels were significantly warmer than the other nodes.*

*(2) There obviously was something wrong in the measurement of the antenna motor node current.*

*(3) Battery #1 remained off during the whole mission.*

The temperature in the pressure vessels would have to be monitored to ensure that it does not exceed allowable limits. Concerning the antenna motor, a 1.3 A current was measured at rest, and when rising/lowering the antenna, the current exceeded 30 A, which is not realistic. The current at rest should be that consumed by the LTM10 node, a few milliamperes. There probably was a mistake in a conversion factor. Battery #1 remained off during the whole mission, which may be explained by the fact that it was significantly less charged than the other batteries and the power requirements weren't as high as to require battery #1 to turn on. Otherwise, there may be a malfunction in that battery node. Then the "mission" variables, describing the AUV state, were analyzed, and are summarized in Table 10.

<i>Variable</i>	<i>Value</i>	<i>Comment</i>
Relative position (East, North)	(0,0) → (1,1.4) m	Small drift <sup>(1)</sup>
Heading	350 → 320 °	Slow variation <sup>(1)</sup>
Pitch	-1.7 – -1.9 °	Static pitch around -1.7° <sup>(2)</sup>
Roll	7.8 – 8.5 °	Static roll around 8° <sup>(2)</sup>
Depth	9 – 13 cm	
Altitude	598– 61 cm	

*Table 10: Vehicle State Data Summary for Mission 1 on 10/14/02*

*(1) The small, slow variations in position and heading are mainly due to the drift of the AUV before it was tighten to the pool walls.*

*(2) There exist a static pitch and roll since the vehicle had not been trimmed before the pool test.*

The primary conclusion is that the health of the vehicle seems correct, although the temperature in a few pressure vessels was a little high. It would be necessary to check that the battery #1 was off because it wasn't charged enough, and not because of another problem. The bug in the antenna motor current conversion had to be fixed.

### **III.3.1.3.2. Mission 2**

The console didn't report anything unusual for mission 2, but we discovered that the console output for the third mission has been appended at the end of the second mission console. The health and actuators variables were extracted from the logger. Again, humidity and leak checks were fine. The temperature kept increasing slowly. The battery voltages kept decreasing from 56 V to 55.5 V. We then particularly analyzed the data related to the fins. Whenever a command was sent to a fin, it reacted with a 100-200ms delay, and then reached the setpoint in 350ms. Small overshoots were sometimes evident, but there was no steady-state error in the fin position. When exercised, the fins motor consumed a current below 1.5A. The fin test reported everything working normally.

### III.3.1.3.3. Mission 3

As said before, there were no console output file, and the output was appended to the previous mission console. The same health and actuators variables were analyzed. The temperature in the pressure vessels kept rising. The thruster temperature increased with the rpm, from 29°C to 32°C. The batteries voltages dropped while the rpm increased, from 55.9V to 53.4V, and eventually went back to 55V when the thruster stopped. Again, leak and humidity checks were fine. As far as the actuators are concerned, it is interesting to look at a comparative plot of the commanded and actual rpm (Figure 26).

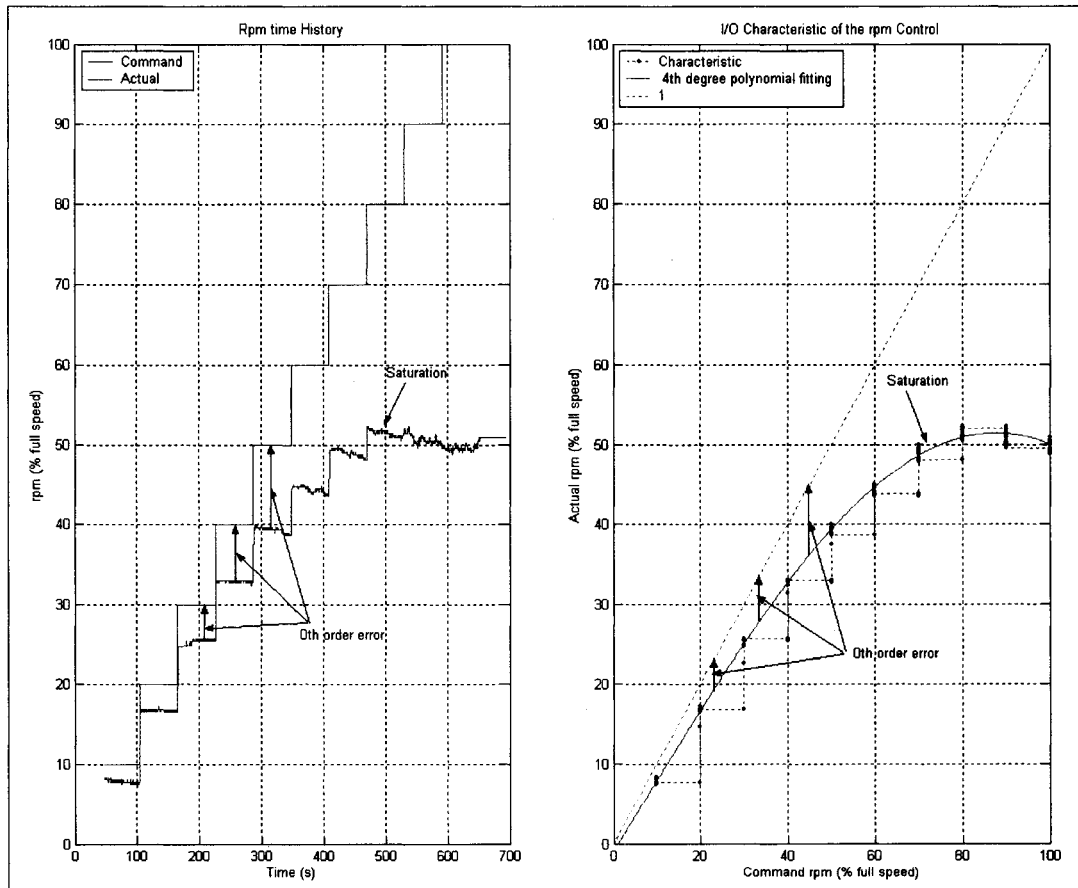


Figure 26: Thruster Test Results

There exists a non-zero steady-state (zeroth order) error, that seems to increase with the

rpm. This may be explained by a gain error in the feedback loop of the motor controller. A saturation is obvious for commands above 70%, which can be explained by the fact that only five batteries were present. It was thus not possible to deliver enough current for the thruster to keep increasing rpm. When the actual rpm was around 52%, the current in the thruster was around 9A. Each of the batteries was delivering 2A, except battery #1, which was delivering only 1.5A, probably because it was not fully charged. This confirmed that almost all the current coming from the batteries was consumed by the thruster. When the batteries were unable to output more current, the thruster saturation occurred. Aside from this steady-state rpm error, the thruster seems to be working satisfactorily, although the same kind of test should be performed with all batteries fully charged.

#### **III.3.1.4. Conclusion**

The pool tests showed that:

- The vehicle is healthy, there is no sign of leak or humidity problem. Nevertheless, the temperature in some pressure vessels was quite high. This may be explained by the fact that the pool water wasn't cold enough to sink the heat from the pressure vessels.
- Overall, the vehicle seems so far to be working satisfactorily, although a few things require a fix or more investigation:
  - The antenna motor current measure probably has a scaling problem,
  - Battery #1 needs to be checked, to ensure that its behavior during that test was due to an incomplete charge and not a malfunction,
  - The thruster has to be tested at high speed when all batteries are available,

- The steady-state error problem in the thruster rpm needs to be investigated,
- The problem of a mission console being appended to a previous console file has to be solved. This problem has been seen from time to time with both OEX-C and Morpheus.

The antenna motor current scaling problem has been fixed. Indeed, the factor used for the conversion from NV value to shared memory value was wrong. All available batteries have been fully charged, and the behavior of battery #1 has been monitored. There didn't seem to be any problem. The vehicle was then ready for an at-sea test.

### **III.3.2. At-Sea Test**

This experiment was intended to be a general test for the vehicle. Its major capabilities had to be tested so as to ensure the general working of the vehicle and identify the points that needed more work. Five missions were planned, so as to test the overall navigation capability of the vehicle, then more particularly test the depth control, altitude control, and finally the speed capability in straight run.

#### **III.3.2.1. Mission Planning**

The first mission was intended to be a general test of the vehicle. This mission was planned as a surface mission, for safety reasons, and in order to be able to compare the vehicle dead-reckoning navigator data with GPS data. The mission was written as a 300m x 300m surface box pattern. The vehicle was controlled to a 3kt speed, was using homing control on the two first legs, and tracking on the two others. The GPS wasn't used for navigation, but was logged for later comparison.

The second mission was the exact copy of the previous one, except that the vehicle was controlled with thruster rpm instead of speed. Indeed, some problems had previously occurred with the speed controller of the Morpheus, which uses a similar high-level software. Therefore, it was likely that the speed controller would not work here either. Therefore, only one mission was written with speed control to test its working, and the other were written with rpm control. So this mission was also a 300m x 300m surface box, without use of GPS for navigation. The rpm was set to 50% of the full speed, which was assumed to approximately correspond to a 3kt water speed.

The third mission was planned as a 300m x 300m box pattern with a depth command set to 3m. The vehicle was controlled to 50% rpm, and was only using tracking control. The maximum safety depth was set to 15m, and the minimum safety altitude to 3m. The use of the GPS for navigation, when available, was enabled.

The fourth mission was planned as a 200m x 200m box pattern with an altitude command set to 5m. The vehicle was controlled to 50% rpm, and was only using tracking. Since the maximum safety depth was set to 15m, the water column had to be shallower than 20m so that the vehicle could achieve its mission. The minimum safety altitude, which had to be less than 5m, was kept to 3m.

The fifth mission was planned as a straight run with a constant true heading command towards South. The rpm command was increased from 30% to 100%, with a 10% increment every minute. The depth command was set to 3m.

As usual, GPS fixes were added at the beginning and end of each mission.

### **III.3.2.2. Mission Execution**

These five missions were run off Dania Beach on October 16, 2002. After a missionCheck that reported no error, the vehicle was launched and the first mission was started through the modem. Right after completion of the GPS fix, the thruster spun at full speed, then stop, start again and so on. The mission was aborted as it appeared clearly that the speed controller wasn't working, as had been previously seen with the Morpheus. The second mission was then started. The vehicle performed its surface box satisfactorily. It was only noted that on the southbound leg, the vehicle, running leeward, was regularly submerged by waves. Therefore, it was likely that some GPS fixes would be missing on that leg. The mission ended normally, and the third mission was started. The vehicle dove, and, according to the tracking system, it correctly performed its box pattern, while reporting a depth of 3m through the modem. The vehicle then surfaced, and the fourth mission was started. The vehicle dove, performed its box pattern around 5m altitude and surfaced. The last mission was started, and the vehicle dove towards the South for its speed run. Through the modem, the vehicle reported a measured water speed of 2m/s, but a saturation occurred, and the following higher rpm command did not make the vehicle go faster. At the end of the 1000m run, the vehicle surfaced and was recovered.

### **III.3.2.3. Mission Analysis**

Various analyses were performed on the data obtained from the missions loggers. Only the important results are summarized here.

### **III.3.2.3.1. Mission 1**

This mission wasn't successful because of the speed controller problem. To get an idea of the reasons for that failure, certain speed controller related variables were extracted from what had been logged before the mission was aborted. To summarize our investigations, we observed:

- During the mission, the speed controller rpm output was oscillating between 5 and 20% of full speed. The actual rpm remained 0 until, after the GPS fix, the autopilot rpm command was set to 60%. At that point, the actual rpm approximately followed the oscillating rpm command outputted by the speed controller.
- The measured AUV speed remained between  $\pm 0.1\text{m/s}$  during the GPS fix, and increased to  $\pm 0.3\text{m/s}$  when the thruster started spinning.

It is likely that there is a problem in the speed controller, since:

- This problem has previously been seen with the Morpheus,
- The feedback providing the actual AUV speed seems to be working.

Therefore, this is probably a high-level software issue, rather than a software integration mistake.

### **III.3.2.3.2. Mission 2**

Since this was the first mission at sea, the health data were observed with much attention. Nothing particular was observed, except a quite high temperature in the main pressure vessel ( $33^{\circ}\text{C}$ ) and the control box ( $37^{\circ}\text{C}$ ). Then the navigation data was analyzed, comparing the position estimator and the GPS data, in order to estimate how well the vehicle computed its position by dead-reckoning (Figure 27).



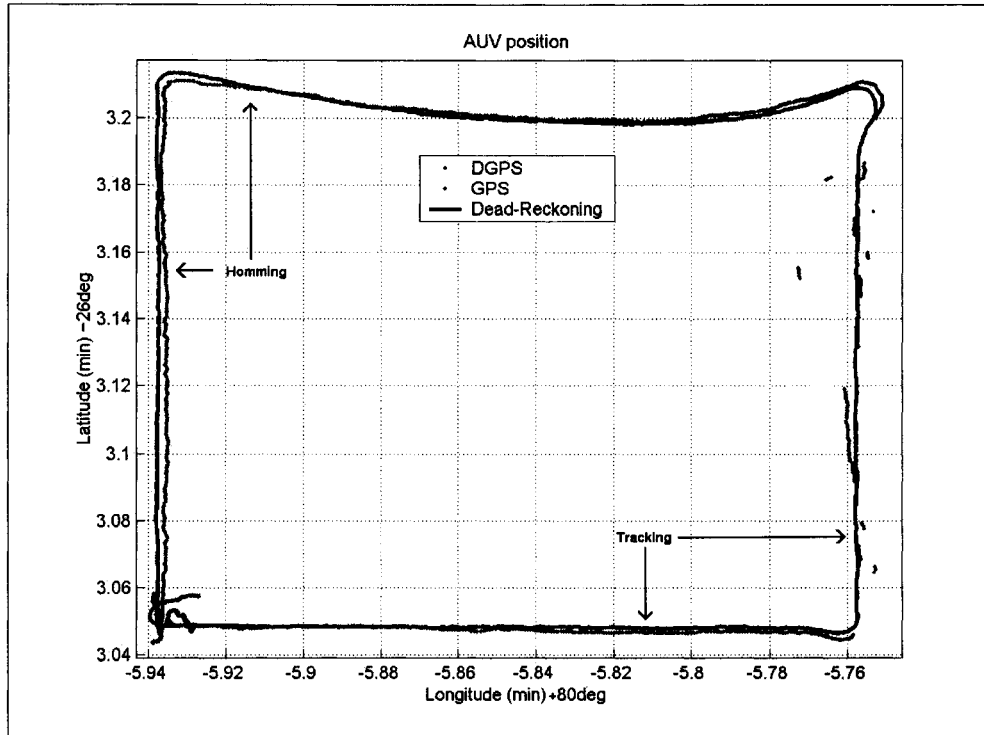
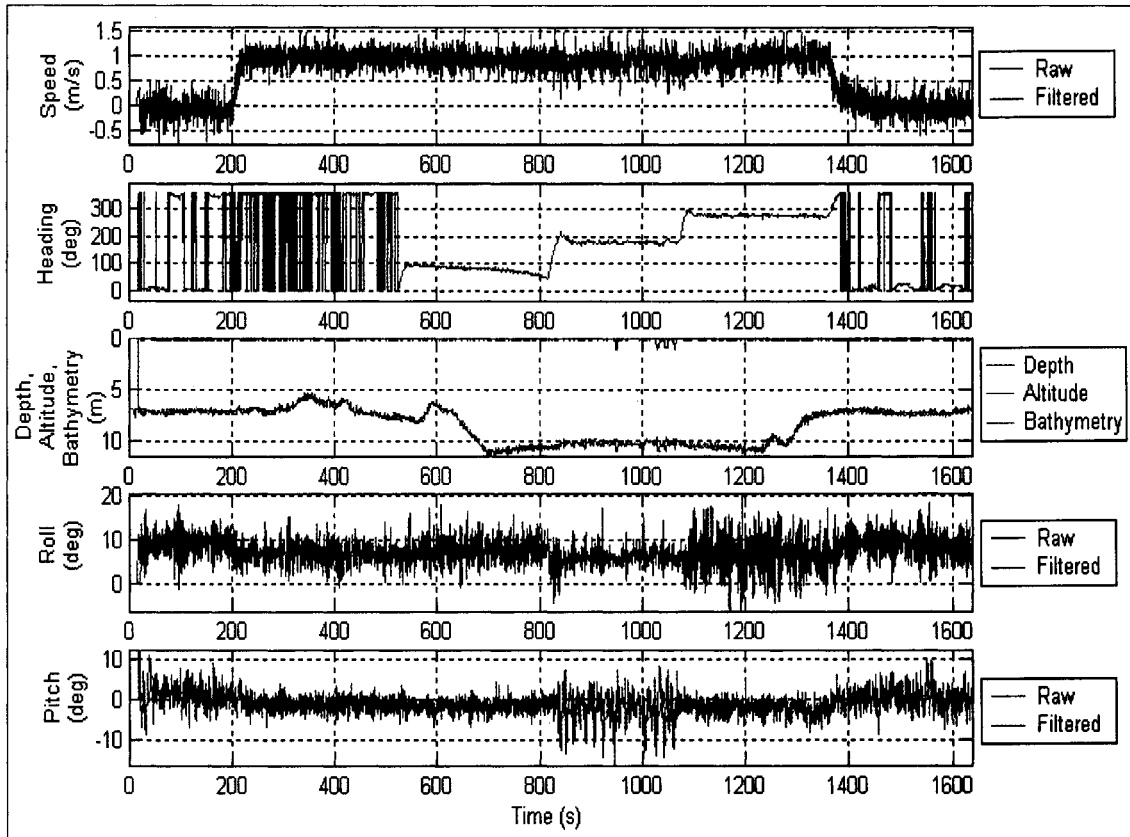


Figure 27: Dead-Reckoning Navigation Compared to GPS Position

The navigation was surprisingly accurate, especially given that the compass had not been recalibrated. During the whole mission, the difference between the position estimated with dead-reckoning and the GPS data remained under 4m. As expected, the GPS data on the southbound leg was of poor quality because of the waves that regularly submerged the antenna and caused the GPS receiver to lose track of satellites. On the above plot, the difference in the navigation between tracking and homing control appears clearly, especially along the East-West legs. Since a significant current was directed towards North, the vehicle was more likely to drift when crossing that current. On the second leg (Northwest to Northeast corners), the vehicle was using homing, and although it was drifting with the current, it was always pointing towards the waypoint, which it eventually reached after a curved run. On the fourth leg (Southeast to Southwest corners), the

vehicle, using tracking, was trying to stay on the direct track joining the previous and next waypoints. Other interesting information is summarized in Figure 28.

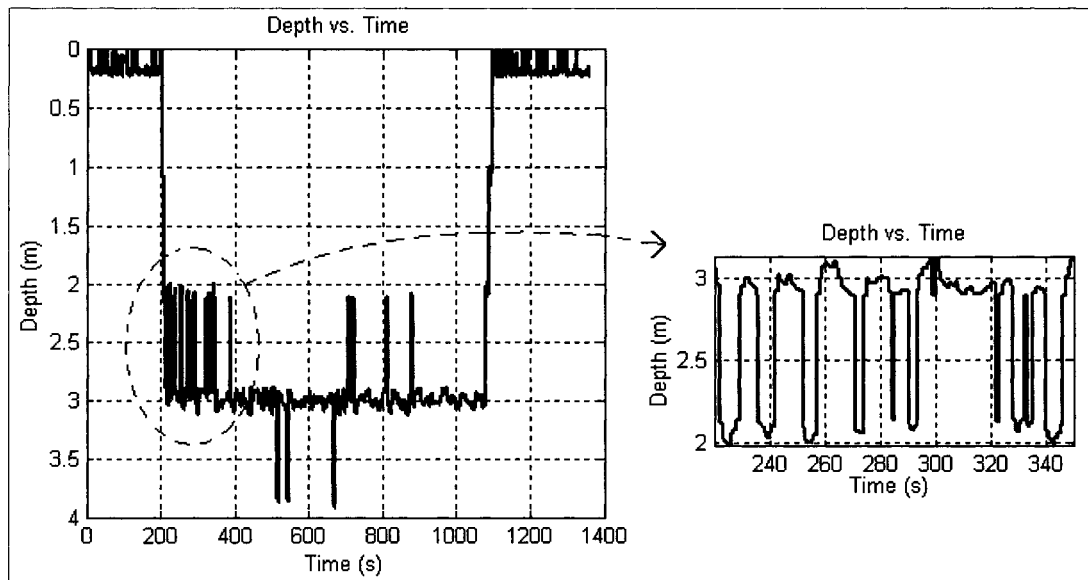


While cruising, the AUV reported a speed of around 1m/s. This confirms that the problem seen previously with the speed controller does not seem to come from a lack of feedback. We can check that altitude and bathymetry increased as the vehicle went towards the East. Some little jumps appear in the depth measurement on the southbound leg, since the vehicle was regularly submerged by waves. A static roll around  $10^\circ$ , and a static pitch around  $2^\circ$  are evident. The AUV was oscillating a lot because of the waves when it was idling during GPS fix, but became more stable as it began to move. We can observe more important oscillations in the pitch readings during the southbound leg. Since the waves

were coming from behind, the wave encountering frequency was reduced, and was then probably approaching the natural frequency of the vehicle. Overall, the mission ran as expected.

### **III.3.2.3.3. Mission 3**

In this mission, the main concern was the depth control. Again, the health data was analyzed, and no problem was found. The navigation was satisfactory, and the AUV correctly performed its box pattern. We then had a closer look at the depth time history (Figure 29).



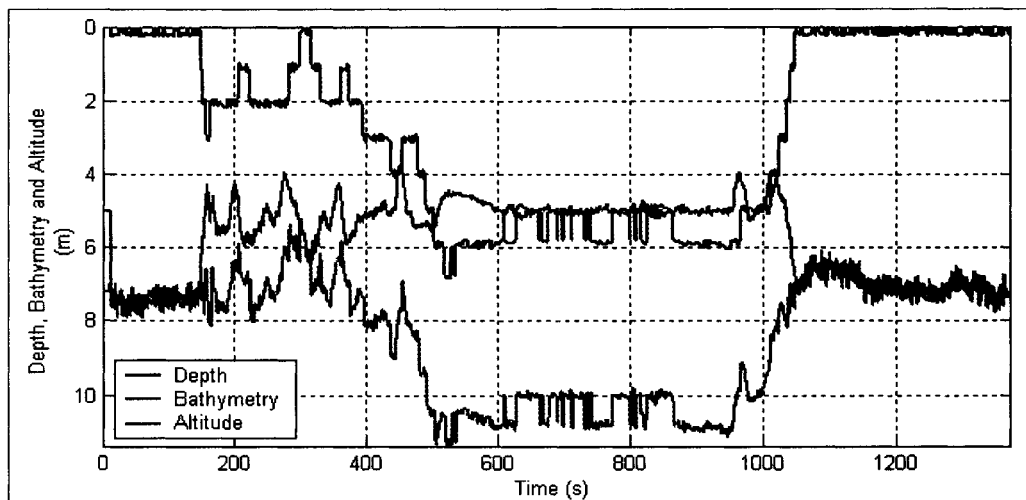
*Figure 29: AUV Depth during Depth-Controlled Mission*

We can observe a lot of jumps, with a significant magnitude. Successions of jumps as large as  $\pm 1\text{m}$  in a few second often occur, which is not realistic at all. Indeed, even if the vehicle may be able to dive at  $1\text{m/s}$ , it is clearly unable to dive and then climb at such a high velocity without transition. These jumps were thought to be related to some noise in the CTD measurements, due to the acoustic modem transmitting nearby, as had

previously been observed with the OEX-C. Nevertheless, it seems odd that the jumps did not occur regularly, with a frequency related to that of the modem transmissions. Besides, since the vehicle was controlled based on the depth measured from the CTD, it tried to correct for the measured errors due to the CTD noise. Therefore, the activity of the sternplanes was increased, and correlated with the CTD noise. This reduced the smoothness of the trajectory, and increased the power consumption. Otherwise, in the absence of such noise, the depth controller maintained the AUV within  $\pm 15\text{cm}$  of the commanded depth.

#### **III.3.2.3.4. Mission 4**

In this mission, the main concern was the altitude control. Again, the health and navigation data were analyzed, and no problem was found. We then had a closer look at the altitude time history (Figure 30).



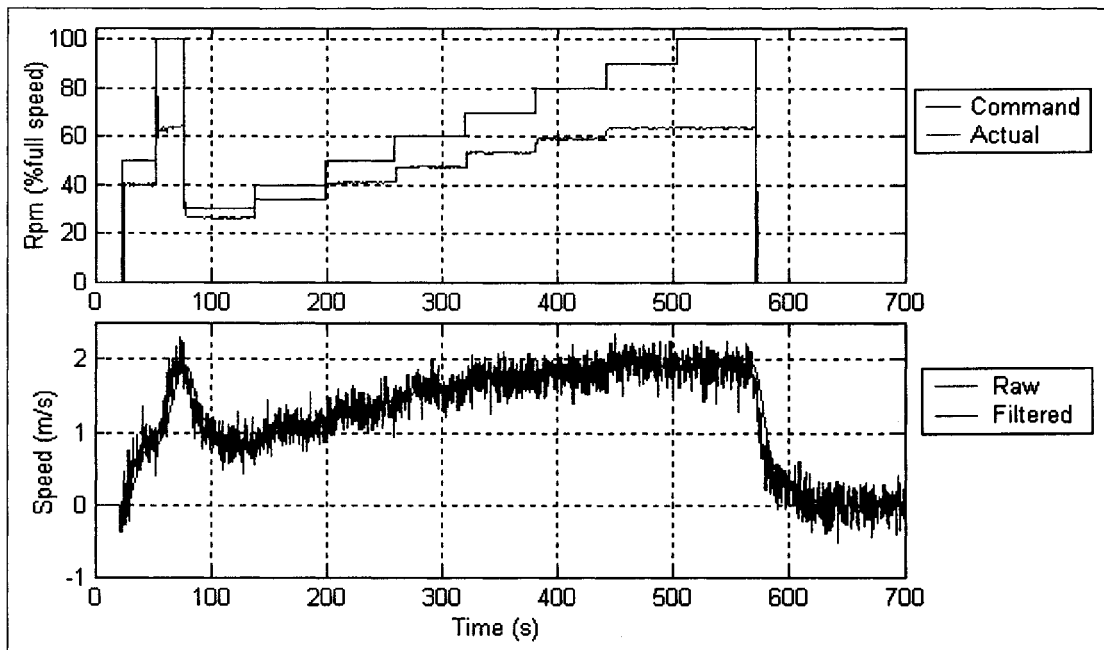
*Figure 30: AUV Altitude, Depth and Bathymetry during Altitude-Controlled Mission*

We can verify that the controller maintained the vehicle within  $\pm 1\text{m}$  of the commanded altitude, and even within  $\pm 20\text{cm}$  during the second half of the mission. These 1m

oscillations between 200 and 400s may be explained by the variations of the seabed while the vehicle was traveling towards the North. These variations are evidenced in the above plot of the bathymetry, although this plot must be considered carefully since the bathymetry estimation is perturbed by the noise in the depth measurements. Since the seafloor variations were quite fast, the vehicle may have been unable to react quickly enough to maintain a 5m altitude. However, once the seabed became smoother during the second half of the mission, the altitude control was much efficient. Overall, the altitude control worked satisfactorily. We can still observe a lot a jumps in the depth measurements.

### **III.3.2.3.5. Mission 5**

This mission was a speed test. It is then interesting to look at the vehicle speed and thruster data (Figure 31).



*Figure 31: Speed Test Results (Rpm and Speed Data)*

The steady-state error problem we saw during the pool test still persists. Again, a saturation occurred because only five batteries out of eight were used. Particularly, the fastest speed achieved was 2m/s, for a commanded rpm of 90% (achieved: 63%). At that point, one of the batteries ran out of energy and turned off. The remaining batteries were then outputting their maximum current, which wasn't enough to maintain a 2m/s speed. The vehicle slowed down to 1.9m/s, and the following command to increase rpm could not be satisfied.

Looking at the fins data, we observed a lot of oscillations in their position. The magnitude of these oscillations increased with speed, until, at full speed, the fins kept going “bang-bang” by as much as 20°. This can be explained by the mission plan we used, in which the commands given were a 180° heading and a 3m depth, without tolerance. Then the fins always tried to correct the slightest deviation. Moreover, as noticed before, the sternplanes activity was increased by the jumps in the depth measurements.

Looking at the attitude angles, we observed that the roll, which was statically around 10°, decreased towards 1° as the speed increased. Thus, a vehicle perfectly trimmed would likely roll by as much as 10° at full speed, probably because of the torque applied by the water on the blades.

Otherwise, the vehicle almost ran a perfect straight line towards South, only drifting about 10 meters towards East in a 900m long run, which approximately corresponds to a 0.7° heading error.

### **III.3.2.4. Conclusions**

The conclusion of this at-sea test is that, overall, the vehicle performed well. According to the AUV engineers present during that mission, the OEX-D seems even more robust than the previous versions of Ocean Explorer. Indeed, the five missions were run successively with one minute interval without any problems. The vehicle did not abort any mission by itself, dropping the ballast weight, as had regularly been seen with the other OEXs. The navigation was surprisingly accurate, even without the compass having been recalibrated for years.

Aside, there remain a few things to fix or investigate. For instance, the jumps in the CTD readings, thought to be due to the modem transmitting in its neighborhood have to be investigated in order to try and find a way to reduce that noise that undoubtedly has an influence on the actuators activity when the vehicle is controlled on a depth command. The thruster rpm steady-state error still needs to be fixed, as well as the speed controller. Another piece of information derived from this experiment is an estimate of the power consumptions of the vehicle without payload. Based on the three box patterns run, and considering only the part of the missions when the vehicle was actually cruising (at 2 knots), a mean power consumption of the order of 150W was measured.

### ***III.4. Various Fixes and Conclusion***

After the development of the vehicle in the lab, a few other fixes were performed based on observations from the pool test and first at-sea mission. These fixes mainly consisted of :

- Fixing the antenna motor current reading,
- Trying a way to fix the speed control. It seems that the speed arbiter was not configured correctly, and thus was unable to perform a speed control. This configuration has been modified. In order to assess the efficiency of that fix, a speed test was later conducted, but unfortunately still proved not to be working. This malfunction is currently under investigation.
- Trying to reduce the jumps in the CTD measurements. To do so, the sensor was replaced by another of the same model, which had been modified by the Electronics Laboratory at Seatech so as to implement a lowpass filter designed to reduce the modem-generated noise.

Overall, the vehicle is safe and efficient. There doesn't seem to be any major software or electronic issue. The navigator, position estimator and most of the controllers seem to be working satisfactorily, although the depth controller is not as efficient as it should be, and no closed-loop speed control is available.

It was then decided that the OEX-D was an operational vehicle, ready to be used for performing real missions, that would undoubtedly reveal a few other problems to be solved.



## **IV. Mapping the Thermal Structure of a Water Column**

This chapter describes the experiments in which the OEX-D was used to map the thermal structure of a water column. Three missions were run in December 2002 and March 2003 off Dania Beach. The scientific goal of these experiments was to observe the structure of a vertical water slice, mainly around the upper part of the thermocline layer, in terms of its thermal structure. To this end, the AUV carried a CTD sensor along a predefined pattern throughout the chosen water slice.

The motivations for these missions were twofold: Firstly, they were intended to gather measurements to improve our knowledge about the thermal structure of a water slice. Particularly, the emphasis was on the order of magnitude of the temperature variation with depth within and outside the thermocline, as well as the variations of the temperature profile and the thermocline over the horizontal distance. Secondly, the aim was also to acquire data that, through post-processing, would enable the reconstruction of a temperature map of a water slice. Such a map of the temperature of a water slice can later be used as an input for a thermocline tracking simulation. Because of the primary goal of the thesis, we were mainly interested in acquiring temperature. Nevertheless we also wanted to measure the water current profile in order to estimate how the variation of the current may be related to the thermocline.

## ***IV.1. Requirements and Assumptions***

In order to perform exhaustive sampling, temperature data was acquired with the OEX-D. Indeed, as already discussed in Chapters 1 and 2.1.4, the use of an AUV for such purposes is more efficient than several shipboard CTD casts. To enable thorough sampling of a vertical water slice while allowing the AUV to travel in an efficient way, the vehicle was programmed to perform a “vertical lawn-mower pattern” consisting of several horizontal legs vertically spaced.

To reduce the mission time, in order to prevent temporal variations from being mistaken for spatial variations, the AUV depth variation should be monotonic. The AUV should thus either run the deepest leg first, followed by subsequent shallower legs, or the other way around. The leg separation distance is also a parameter influencing both the spatial resolution of the survey and the mission duration.

In order to enhance the compromise between resolution and instantaneousness of the survey, the pattern should be finalized only when more information about the water column is available.

The survey location should be chosen where a significant thermocline is likely to be found. Moreover, the mission path extent should be oriented so that the horizontal variation of the temperature profile is maximized. A water slice oriented along the East-West direction, across the Florida Current was selected. The reason for this choice are threefold: Firstly, previous experiments, such as the “Cross-Shelf CTD experiments”, have shown that no significant thermocline was likely to be found closer to the shore. On the other hand, at that distance, a thermocline, shallow enough for the AUV to sample it,

is likely to be found. Secondly, at that location, the seafloor variation is important, which is believed to have a significant influence on the horizontal variation of the temperature profile. In order to see important variations in the seafloor, and hence possibly in the thermocline, the survey legs have to be sufficiently long, say 1000m. Finally, it is of interest to run across the Florida Current, since, where the current is significantly strong, some variations of the current profile related to the thermocline may more probably be observed.

In order to map a single water slice, the AUV has to remain in the same water column throughout the whole mission. Assuming that the Florida Current is mainly oriented towards North, and considering the current variations to be sufficiently small along the East-West direction, the vehicle, crossing the current and drifting with it, would remain in the same water slice.

## ***IV.2. Method of Survey***

Based on the goals and requirements described above, the following method of survey was chosen: The AUV was programmed to run several horizontal 1000m long legs across the Florida Current, along the East-West direction. The deepest leg was run first, and then the AUV depth was reduced for each subsequent leg. In order to maximize the amount of information obtained from the survey, the parameters, such as number of legs and separation distance between legs, were decided at the very last moment, based on knowledge acquired at that time with shipboard CTD casts. Therefore, the AUV mission had to be written in a way that allows these parameters to be modified on the fly.

The data recorded were:

- CTD temperature,
- ADCP current profile,
- 3-Dimensional AUV position (horizontal position from the position estimator, and depth from the CTD).
- AUV attitude and velocity (to post-process the ADCP data).

Moreover, we also wanted to take advantage of these missions to acquire other data provided by the sensor payload referenced to as the Turbulence Package, described later. To enable the Turbulence Package to acquire meaningful data, the water speed of the AUV should exceed 0.5m/s. A water speed of 1.25m/s was chosen.

In order to ensure that the vehicle remained in the same water slice and drifts with the current, we had to prevent it from correcting its navigation for any drift it may sense. This was actually fortunate because, given the mission location, the water column was between 100 and 200m deep. Thus, during most of the mission, the seafloor was out of range of the DVL, which prevented the vehicle from measuring its ground velocity. Therefore, the vehicle was only able to estimate its motion with respect to a reference layer of water. When the vehicle exactly drifted with the water, it measured a zero velocity, and therefore did not sense the drift and did not try to correct it. But this also means that the horizontal path of the vehicle was a-priori unknown since was dependent on the current. For this reason, it was necessary to have a reliable tracking system to determine the absolute position of the AUV.

The planned mission path, projected along the East-West and vertical directions of the water slice, is presented in Figure 32. In 3-D referenced to the ground, this pattern would

spread horizontally, mainly towards the North, because of the current.

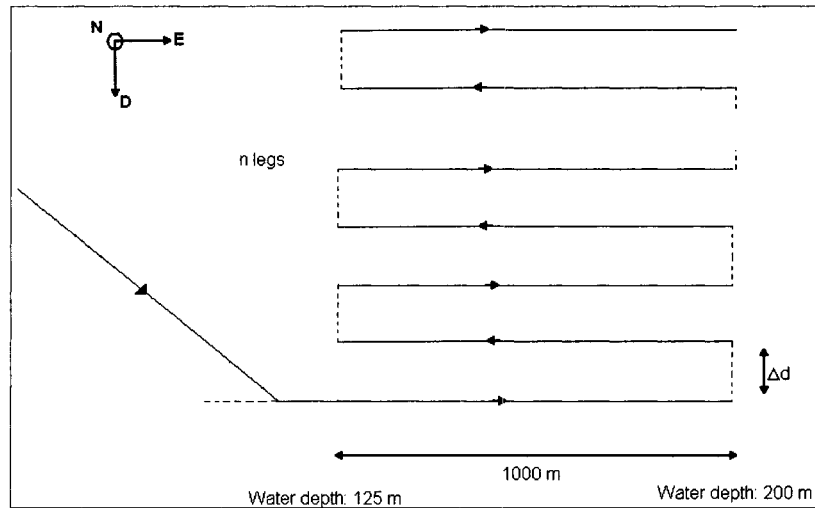


Figure 32: Vertical Lawn-Mower Pattern

Finally, the mission location was chosen to be in the South Florida Ocean Measurement Center (SFOMC) range, off Dania Beach, in the vicinity of the South Florida Testing Facility (SFTF) 600ft deep ADCP, because that range corresponds to the needs described above, and the SFTF ADCP would provide complementary current data. The ADCP is moored at  $26^{\circ}3.8' N$ ,  $80^{\circ}3.6' W$ . To remain in its vicinity, the launch point for the AUV was chosen at  $26^{\circ}3.7' N$ ,  $80^{\circ}4.0' W$ .

### ***IV.3. December 2002 Missions***

The idea was to run two missions in December 2002 within a few days interval in order to allow appropriate modifications to the mission plan in case of problems revealed by the first mission, so as to be able to achieve the second one satisfactorily. As described hereafter, we planned to prepare one mission and run it twice. The data from both missions were then analyzed separately.

### **IV.3.1. Preparing the Missions**

The mission plan for the two missions run in December 2002 was designed according to the goals and guidelines detailed above. To prepare the mission, the following tasks had to be performed: payload integration, sensor calibration, pool test and vehicle trimming, and mission writing.

#### **IV.3.1.1. Payload Integration**

For these two missions, the only sensor which was not already a part of the vehicle was the Turbulence Package, a payload designed by the Center for Hydrodynamics and Physical Oceanography. Moreover, although the vehicle DVL had been previously integrated, it was not yet possible to use it as an ADCP to record the water current profile under the vehicle. A few modifications were required to enable such a use of that instrument on the AUV.

##### **IV.3.1.1.1. Turbulence Package Integration**

The Turbulence Package payload comes as a small cylindrical appendage that sticks out of the vehicle nose. The cylinder houses the acquisition system that records the data coming from the sensors mounted on the front end of the cylinder. The package is powered by a separate battery can. Since the Turbulence Package is self contained, no real integration with the vehicle was necessary. The sensor package was mounted at the tip of the vehicle nose. The battery can was secured inside the nose, which also housed a dropweight and a light strobe.

#### **IV.3.1.1.2. Enabling the ADCP Mode of the DVL**

The DVL itself can be configured to operate as an ADCP that outputs the current profile data. What needed to be added was the possibility to switch the DVL to water profiling mode by updating the appropriate shared memory variables from the mission plan, and record the water profile data in the main computer logger.

Since this DVL had previously been used as an ADCP on an OEX-B, there already was the code in the Neuron node application that performs the following tasks:

- Read a LonWorks NV command to change the DVL mode to water profiling and send the appropriate DVL command strings through the serial port,
- Read another NV parameter to set the water bin size for the profiling mode and send the appropriate DVL command strings through the serial port,
- Read the water profile data coming from the serial port and translate it into NVs content to be placed on the network.

What was necessary was to interface these NVs with the main computer shared memory and logger. This kind of task has been detailed in the previous chapter, and is therefore not described further here. Once the integration is completed, the water current profile relative to the AUV, over 16 bins under the vehicle, can be recorded in the logger file.

#### **IV.3.1.2. Sensor Calibration**

The sensors used for these missions were the CTD, the DVL and the magnetic compass. The depth offset of the CTD was measured, and entered in the shared memory reinitialization file, so that it would be removed from the raw measurements. Possible offsets in the temperature and conductivity readings are not critical since we are mainly

interested in the variations of these quantities.

As far as the DVL is concerned, the only possible calibration is that of the heading sensor. But the DVL is currently configured to output the data in a frame referenced to the instrument. The heading sensor of the DVL is not used and thus does not need to be calibrated.

The vehicle magnetic compass calibration, required to achieve an accurate navigation, is done in two steps: Firstly, the internal calibration procedure of the sensor is run. Secondly, a deviation table is computed to correct the sensor output, as detailed in [51]. The vehicle is slowly rotated in different attitudes, so that the internal calibration procedure can estimate and correct the effects of local magnetic disturbances. Then, to acquire the data required for the computation of the deviation table, the vehicle is slowly rotated horizontally while the output from the compass and gyro are logged. Integrating the gyro readings and comparing them against the compass heading, a deviation table is built and the residual error is estimated.

#### **IV.3.1.3. Pool Test and Vehicle Trimming**

As before any mission, the vehicle was taken to the seawater pool at Seatech for a test and trimming. The purpose of the pool test is to ensure that the vehicle is safe and operational. When the vehicle was lowered into the water, its health was monitored during a couple of minutes to check that there was no leak. Then, a test more specific to the mission needs was performed. In our case, we had to ensure that the CTD and DVL were working properly, and that the water current profile was recorded. To do so, we ran a short mission in the intracoastal waterways, enabling the ADCP mode of the DVL, and logging the



CTD and ADCP data. In order to estimate the importance of the modem-related noise on the CTD measurements, the acoustic modem was configured to request navigation data every 30 seconds. A quick analysis was performed on the CTD data (Figure 33 and Table 11).

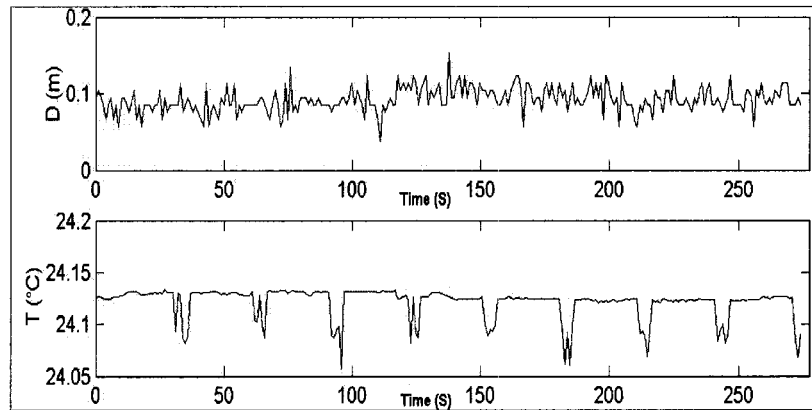


Figure 33: CTD Noise Characterization from Pool Test Data

	<b>Resolution</b>	<b>Min.</b>	<b>Max.</b>	<b>Mean</b>	<b>Range</b>	<b>Std. Deviation</b>
<b>Depth (cm)</b>	$\delta d=1$	3.7	15.3	9.2	11.6	$1.7=1.7 \delta d$
<b>Temperature (°C)</b>	$\delta T=0.001$	24.070	24.130	24.120	0.077	$0.015=15 \delta T$

Table 11: CTD Noise Characterization from Pool Test Data

It appeared that the filter that had been added to the CTD efficiently removes the modem-related noise on the depth channel. Nevertheless, a significant and regular noise appears every 30 seconds on the temperature measurements, and is undoubtedly caused by the modem. Because the magnitude of that noise is less than  $0.1^{\circ}\text{C}$ , and because reducing that noise would require the filter to be redesigned, with the drawback that it would reduce the bandwidth of the sensor, it was decided to keep the sensor as is, and to avoid using the modem during the mission, except when really necessary.

Then the water current profile from the DVL/ADCP was analyzed. Only the current

component along one axis is plotted in Figure 34 as an example.

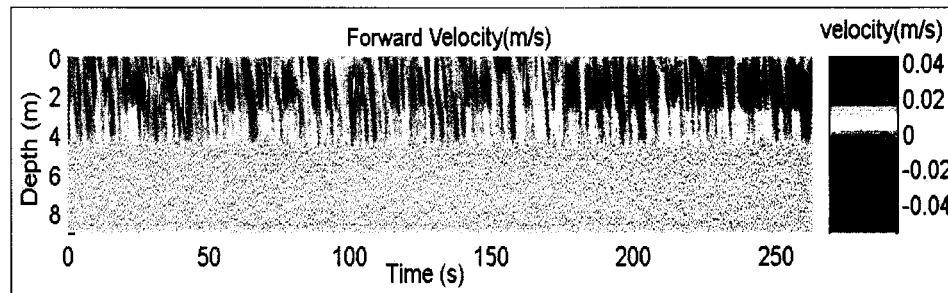


Figure 34: ADCP Data from Pool Test

The velocity is non-zero in the first 3-4 meters, as expected in a 3m deep water column. The same kind of profile was obtained along the two other axis. This confirmed that the ADCP output is logged during missions.

The vehicle was then trimmed so that it is as little positively buoyant as possible, and it has a zero pitch and roll at equilibrium. The aim of this trimming is to ensure that it requires as little actuators action as possible to have the vehicle on a steady flight, that the vehicle doesn't sink, and that it surfaces once the dropweight is released. That trimming is achieved by the use of the appropriate combination of foam and lead.

#### IV.3.1.4. Writing the Mission

The mission was written based on the considerations discussed in 4.1.

The idea, in order to be able to modify the mission on the fly is to write most of it as a macro function for which only a small number of parameters would have to be defined.

One way to write such a macro is to use relative waypoints.

A macro "VerticalLeg L D1 D2 D3" was then defined as follows:

- (1) Go to waypoint L meters East at depth D1,
- (2) Go to waypoint 50 meters East at depth D2,
- (3) Go to waypoint 50 meters West at depth D2,
- (4) Go to waypoint L meters West at depth D2,
- (5) Go to waypoint 50 meters West at depth D3,
- (6) Go to waypoint 50 meters East at depth D3.

This macro instruction is presented in Figure 35.

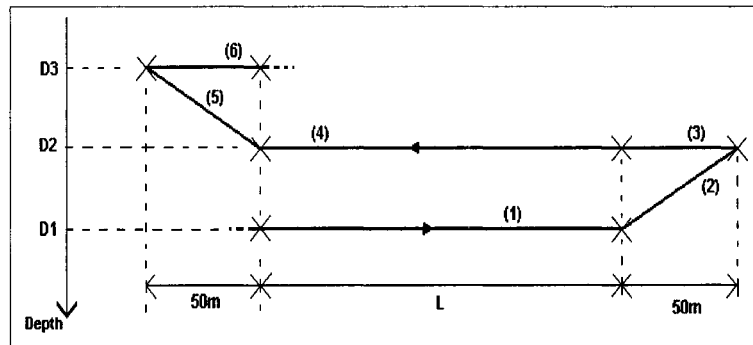


Figure 35: Macro Instruction "Vertical Leg"

Then programming the whole path mainly consisted of as many calls to that macro as necessary, with different depth parameters. The mission was written as summarized below and presented in Figure 36.

- (1) Start mission,
- (2) Configure DVL for water profiling mode with 1.5m water cells,
- (3) Get GPS fix,
- (4) Dive to first waypoint L1 meters East, at depth D1 (L1 is computed once D1 is set, so that the average slope is around 15°),
- (5) VerticalLeg 1000m D1 D2 D3,

- (6) VerticalLeg 1000m D3 D4 D5,
- (7) ...
- (8) VerticalLeg 1000m, Dn-2,Dn-1,Dn,
- (9) Get GPS fix (which first makes the vehicle surface),
- (10) Stop mission.

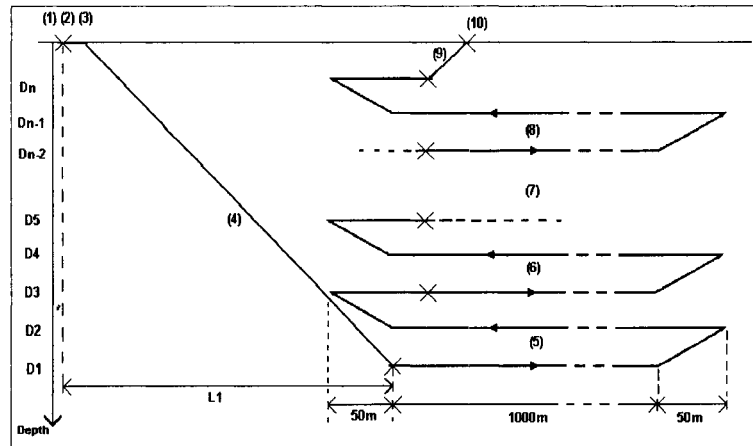


Figure 36: Whole Mission Plan Summary

Finally, the logger input file was checked and modified so that all the required variables would be logged.

### **IV.3.2. Mission Execution**

The first mission, planned to be run on December 13<sup>th</sup>, 2002, had to be rescheduled because of malfunction of the AUV thruster. The thruster motor controller board was replaced and tested, and the mission was rescheduled for December 16<sup>th</sup>, 2002.

After a thorough test of the AUV, we left the dock and proceeded to the mission location. We first performed a CTD cast at the easternmost edge of the mission area (26°3.7'N, 80°3.0'W), then went back to the western launch point (26°3.7'N, 80°4.0'W) to perform a

second cast. A quick analysis of the data obtained from both CTD casts (Figure 37) gave the parameters of the mission.

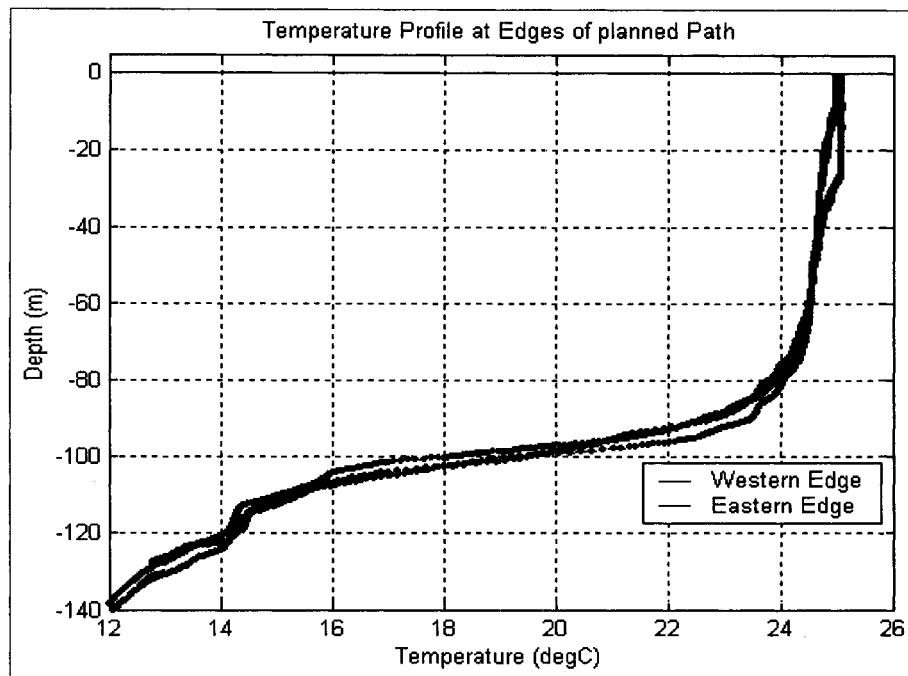


Figure 37: CTD Casts at Edges of Mission Location before Launching the Vehicle

Unfortunately, the main thermocline was quite deep, and its variation over the horizontal distance was not as significant as expected. Indeed, the main thermocline seemed to be almost constant between 80 and 120 m, with only a few tenth of a degree difference between eastern and western edges. A weak shallow thermocline seemed to develop between 10-20m at the western edge, and dive to 30-40m at the eastern edge. In order to map the upper part of the main thermocline, and capture the shallow thermocline variations, we decided to run 8 equally spaced legs between 90m and 20m depth.

The AUV was launched and the mission started. The AUV was unable to dive, and the reason was found to be related to the bottom safety procedure. It seemed that when the bottom was out of range of the DVL, the altitude was reported as zero, which prevented

the vehicle from diving. Given that on the deepest leg the AUV would still be more than 20m above the seafloor, the bottom safety was removed and the mission restarted. As far as it was possible to determine at that time, the mission ran according to our expectations, except that the tracking system was not working satisfactorily. It only gave a few points from time to time, showing significant scatter. The end of the mission also showed the vehicle sinking, and we sent a command to release the dropweight so that the vehicle surfaced. It was found that the battery can powering the Turbulence Package payload had been flooded.

After the vehicle was recovered, we performed a last CTD cast at the recovery point ( $26^{\circ}7.3'N$ ,  $80^{\circ}3.3'W$ ), for later processing aimed at estimating the time-dependent variation of the temperature.

A rapid mission analysis was performed, which gave the results summarized in Figure 38.

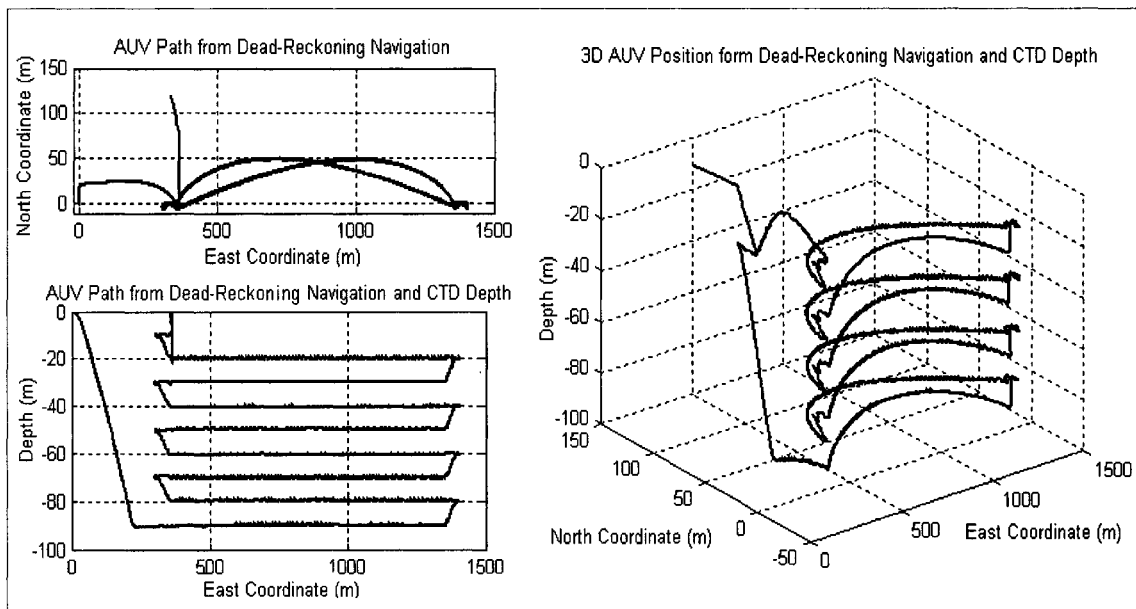


Figure 38: Mission Summary (Navigation)

As that seemed to correspond to our expectations, we decided to run the exact same mission on December 18<sup>th</sup>, 2002. The only required modification was to fix the tracking system. We found that there was a grounding fault and battery malfunction in the tracking system mounted on the tail of the vehicle. To correct these problems, we replaced the whole tracking system on the AUV.

Then we went back at sea on December 18<sup>th</sup>, 2002. We went to the easternmost edge of the mission location (26°3.7'N, 80°3.0'W), performed a CTD cast, went back to the planned launch point (26°3.7'N, 80°4.0'W), and performed a second CTD cast. The data obtained from these casts are presented in Figure 39.

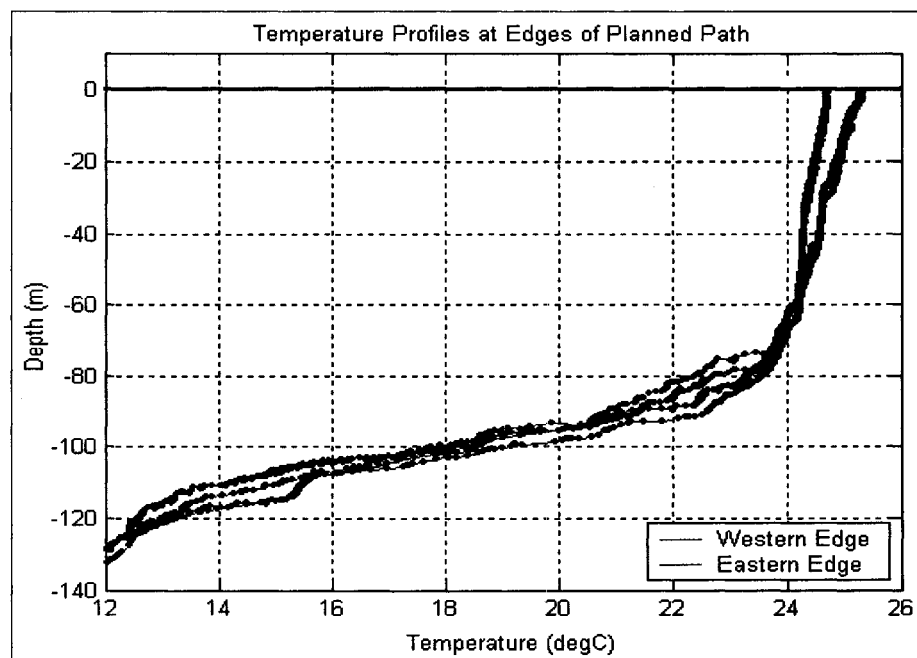


Figure 39: CTD Casts at Edges of Mission Location before Launching the Vehicle

We parametrized the mission so as to run the deepest leg at 90m and the shallowest at 10m. In order to accurately map the regions showing important temperature variations over depth and distance, closer legs were run between 70 and 90m. The mission pattern is presented in Figure 40.

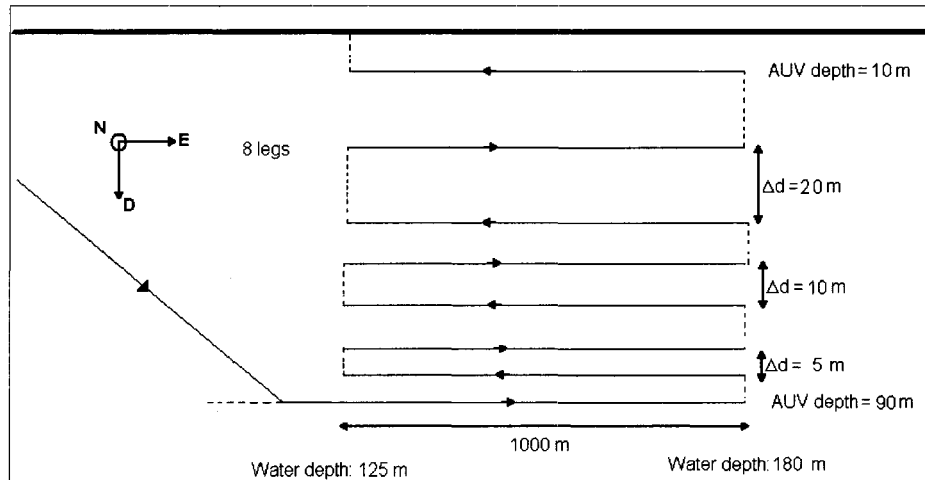


Figure 40: Mission Plan for December 18<sup>th</sup> Mission

We launched the AUV and started the mission. Again, we were unable to obtain good tracking. It took some time to locate the vehicle once it surfaced at the end of the mission, but the vehicle was finally recovered. Again, a third CTD cast was performed at the recovery point (26°5.5'N, 80°3.7'W).

### **IV.3.3. Data Analysis and Results**

For each of the two missions, each piece of data was analyzed separately.

#### **IV.3.3.1. December 16<sup>th</sup> Data**

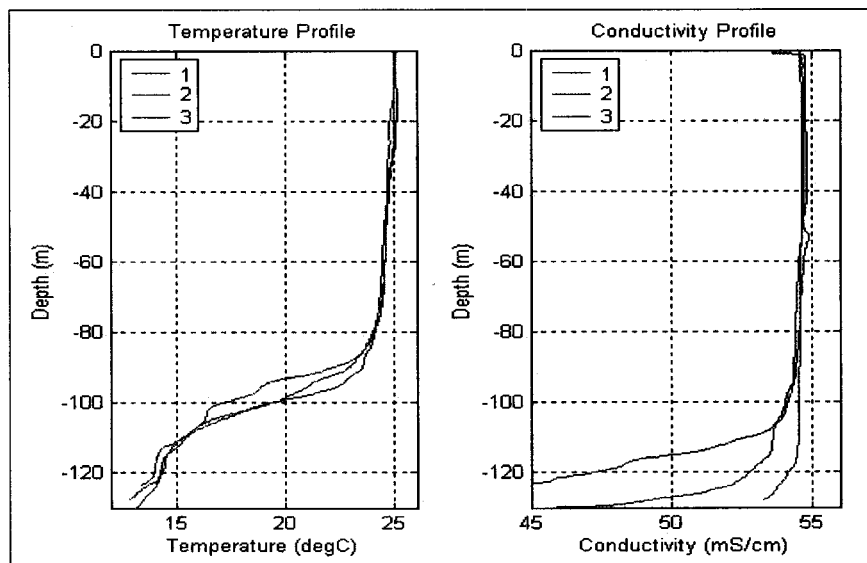
The shipboard CTD data was first analyzed. Then the AUV mission data was examined, to estimate how the AUV performed the pattern it had been programmed for. Then the CTD measurements acquired by the AUV were studied. Finally, the water current profile measured from the vehicle was considered.



### **IV.3.3.1.1. Shipboard CTD Data**

A detailed analysis was performed on the shipboard CTD data, of which only the significant results are summarized here. Both the temperature and conductivity as functions of depth were analyzed. The motivation for the analysis of the conductivity profile was to see if it could provide a complementary way to locate the thermocline.

Only the part of the data corresponding to the CTD going down was kept because it is considered more reliable, since, when the CTD is going up, the probes are more likely to be in a turbulent wake of the instrument. After having been sorted based on the depth measurements, all three variables were filtered using a Moving Average (MA) filter (Figure 41).



*Figure 41: Temperature and Conductivity Profiles (December 16<sup>th</sup>)*

*(1): 26°3.7'N, 80°3.0'W, 14:30 UTC, (2): 26°3.7'N, 80°4.0'W, 15:00 UTC,*

*(3): 26°7.3'N, 80°3.3'W, 22:00 UTC*

Once the data was filtered, the derivatives of temperature and conductivity with respect to depth were taken (Figure 42).

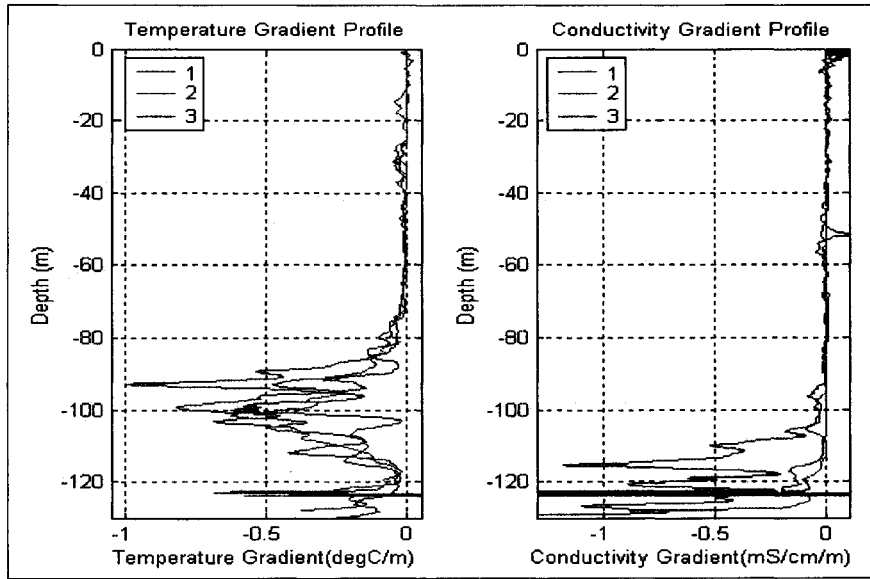


Figure 42: Temperature and Conductivity Vertical Gradient Profiles (December 16<sup>th</sup>)  
 (1): 26°3.7'N, 80°3.0'W, 14:30 UTC, (2): 26°3.7'N, 80°4.0'W, 15:00 UTC,  
 (3): 26°7.3'N, 80°3.3'W, 22:00 UTC

Figures 41 and 42 show a strong deep thermocline at around 100m depth, showing a temperature gradient of a few tenth of a degree per meter over a layer of 20 m thickness, with a maximum gradient as large as 1°C/m. In the mixed layer above the thermocline, the vertical temperature gradient was less then 0.02°C/m. Unfortunately, there was no significant temperature profile variation in the main thermocline over the 1000m horizontal extent of the mission. The small shallow thermocline noticeable on the temperature profile is weak, with a temperature gradients around 0.07°C/m, but the plot of the temperature gradients emphasizes the depth variation of that shallow thermocline between the two edges of the survey location. It seems that the deep thermocline is a few meters shallower on the last CTD cast. Nevertheless, it is impossible to tell whether this difference is due to a time-dependent variation, or is a spacial variation due to the fact that this last CTD cast was performed 3.5nm North of the two others. In such a case, this

would indicate that a more significant horizontal variation of the thermocline can be found along the North-South direction, rather than along the East-West one, as expected. Finally, these plots show that the conductivity decreases with the temperature, and that a layer of important variation in conductivity seems to exist below the top of the thermocline, but this is not sufficient to formally relate the two phenomenon in a way that can help locate the thermocline in a systematic manner. The primary conclusions are that:

- So far, there isn't enough information to formally derive an obvious way to use the conductivity data to help locate the thermocline,
- The shipboard CTD data from this mission does not seem to show a significant horizontal variability of the thermocline. A 20m thick main thermocline exists around 100m depth, with vertical temperature gradient around  $0.5-1^{\circ}\text{C}/\text{m}$ .

#### **IV.3.3.1.2. AUV Mission Data**

In order to estimate how well the vehicle performed its mission, we looked at its trajectory, and compared it to what was intended. We first considered the vertical trajectory, obtained by projecting the 3D path in the vertical (East-Up) plane (Figure 43).

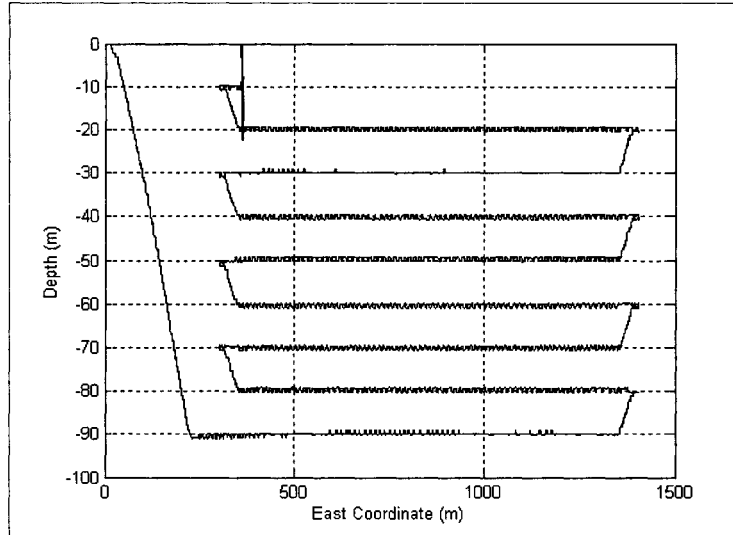
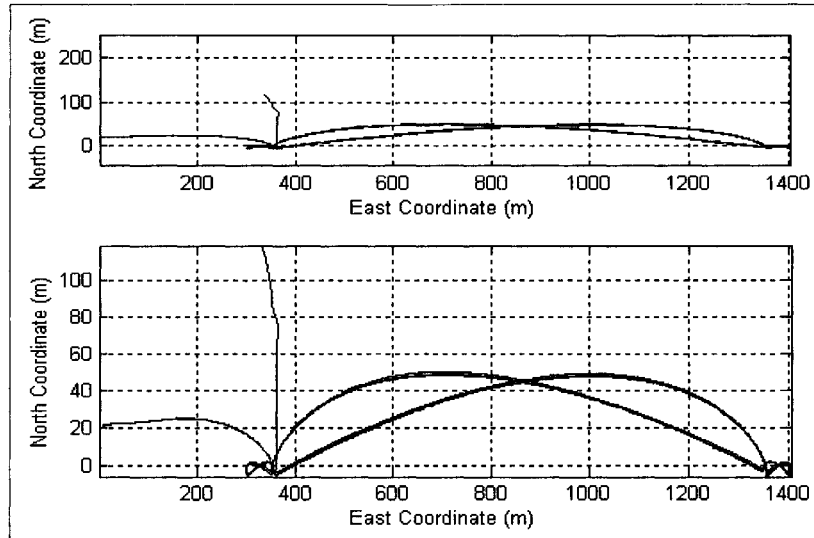


Figure 43: AUV Vertical Path from Dead-Reckoning Navigation and CTD Depth

Overall, this pattern corresponds to what was expected. Nevertheless, the thickness of the plot shows a significant amount of fluctuation on the depth, which may be due either to vehicle motion or to noise in the measurements. Further analysis showed that the fluctuation was rather noise in the measurements, which in turn had an influence on the fins activity and the vehicle motion because the vehicle was under depth control. Since this noise exists even when the modem is not transmitting, it cannot be a modem-related noise.

Figure 44 shows the horizontal trajectory in the horizontal (North-East) plane, as computed by the position estimator.



*Figure 44: AUV Horizontal Path from Dead-Reckoning Navigation  
(Two Representations: Equal Axis and Tight Axis)*

Although on the top figure, the horizontal path referenced to the water column seems to correspond to what was expected, the second plot, enlarging the scale of the North axis, shows that somehow the AUV sensed a drift and tried to correct it, as evidenced by the trajectory curved towards North in the middle of each leg. This needed to be modified for the next mission, in order to ensure that the vehicle remained in the same water slice throughout the whole mission.

#### **IV.3.3.1.3. AUV CTD Data**

The plot in Figure 45 shows the raw data for the depth and temperature versus time.

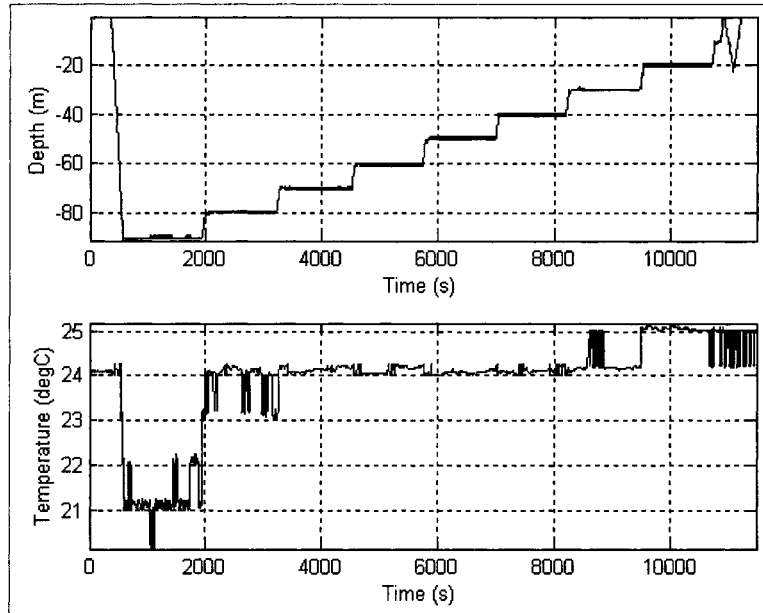


Figure 45: Raw Depth and Temperature Data from AUV CTD Sensor

The first plot confirms, as seen before, that the depth pattern performed by the AUV is as expected, but that the depth measurements are noisy. The same noise is more evident in the plot of the temperature. Even without considering the noise, the temperature seems wrong because the temperature profile it shows is not realistic and is moreover not what is shown by the shipboard CTD casts. Although it is possible that there is an important offset between the two sensors temperature probes, all shipboard CTD cast showed the same surface temperature. Therefore, even taking into account a possible offset, the AUV CTD should measure the same surface temperature at the beginning and end of the mission, which is not the case here. Moreover, the shipboard CTD data shows approximately a 1°C difference between the temperature at 30m and 80m, and the AUV CTD does not show such a difference.

The plot in Figure 46 shows the time derivative of the depth and temperature, and a tentative temperature profile from the AUV CTD data.

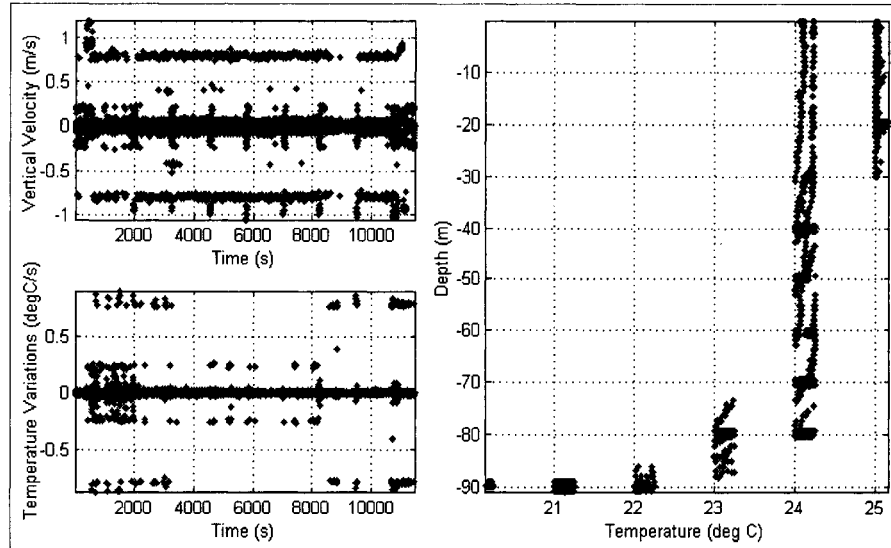


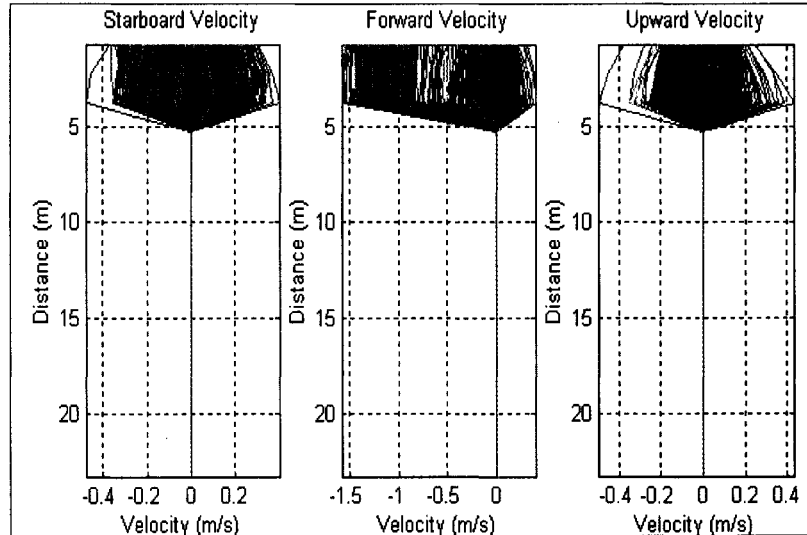
Figure 46: CTD Data Noise and Vertical Temperature Profile

It appears clearly on all plots that the difference between readings is most of the time either 0.2 (cm or °C), or 0.8 (cm or °C), which rather looks like a quantization error.

Indeed, later investigations showed a possible mistake in a data conversion which may have resulted in bits switching. Several methods to correct that were tried, but without real success. Indeed, it seems that at least two bits were switched instead of two others, and since there is no possibility to tell which four bits were concerned, it is impossible to correctly post-process the data to fix the problem. An approximate correction was possible for the depth data, given that the error is around 1m on a full range of 100m (1%), and that the depth can be assumed approximately constant along each leg. The same kind of correction was not possible for the temperature readings, since in that case, the error represents approximately 20% of the data range. Considering that the temperature variations we are trying to analyze to characterize the thermocline are around a few tenths of a degree per meter, it is much less than the noise we have to deal with. Therefore, no conclusion can be drawn from the analysis of this data.

#### IV.3.3.1.4. AUV DVL/ADCP Data

A plot of the raw water current profile data is presented in Figure 47.



*Figure 47: Raw Water Current Velocity in the Vehicle Body-Fixed Frame*

Only the three first bins are non-zero, which is not realistic at all. This rather looks like only the first three bins have been logged. We investigated the software and found the following explanation: The DVL Neuron node sends the data three bins at a time, along with an index that indicates which bin data is being transmitted. On the main computer side, the index was not correctly converted by LonDaemon, and was thus always read as 0. Then, every piece of data coming from the network was successively written in the fields for bins 1 to 3. Because there is no way to post-process the data to determine which data corresponds to which bin, the ADCP data is totally unusable, unless to get a rough idea of the average current over the 16 bins. Since this is not the primary purpose of the experiment, it is not discussed here.



### IV.3.3.2. December 18<sup>th</sup> Data

Again, we analyzed first the shipboard CTD data, then that acquired from the AUV.

Because of the problems discovered during the analysis of the data from the previous mission, no much detail is given here concerning the AUV data.

#### IV.3.3.2.1. Shipboard CTD Data

Again, a detailed analysis was performed on the shipboard CTD data, of which only the significant results are summarized here. Only the part of the data corresponding to the CTD going down was used, as explained in IV.3.3.1.1. All three variables (conductivity, temperature and depth) were sorted, then filtered using a Moving Average (MA) filter. The vertical profiles for temperature and conductivity are presented in Figure 48.

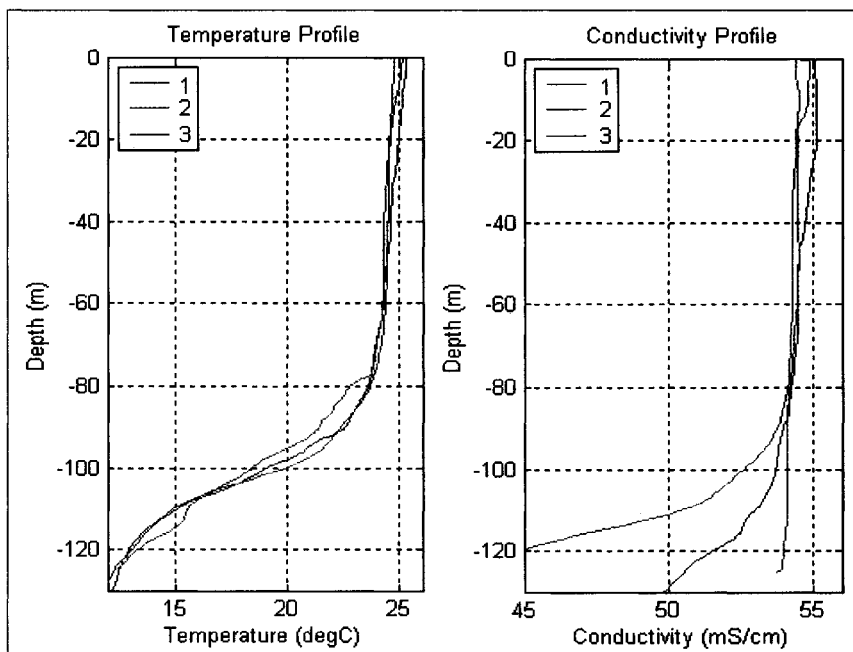


Figure 48: Temperature and Conductivity Profiles (December 18<sup>th</sup>)

(1): 26°3.7'N, 80°3.0'W, 16:00 UTC, (2): 26°3.7'N, 80°4.0'W, 16:30 UTC,

(3): 26°5.5'N, 80°3.7'W, 22:00 UTC

Once the data were filtered, the derivatives of temperature and conductivity with respect to depth were taken (Figure 49).

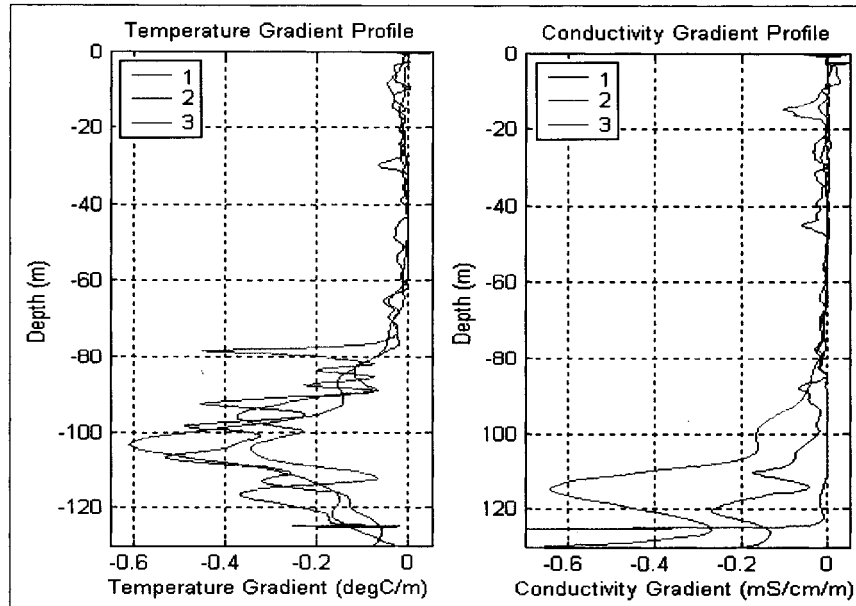


Figure 49: Temperature and Conductivity Vertical Gradient Profiles (December 18<sup>th</sup>)

(1): 26°3.7'N, 80°3.0'W, 16:00 UTC, (2): 26°3.7'N, 80°4.0'W, 16:30 UTC,

(3): 26°5.5'N, 80°3.7'W, 22:00 UTC

As in the case of the data of December 16<sup>th</sup>, a deep thermocline exists around 100m. This one is a little thicker but less intense than on December 16<sup>th</sup>. The temperature gradient in the mixed layer is a little higher than during the previous mission. The interesting thing is the plot from the third CTD cast that shows a very smooth profile, with a little shallow thermocline, a layer of very small gradient, and then the main thermocline with a vertical temperature gradient varying smoothly. Again, there is no significant variation of the temperature profile over the horizontal distance, with only a few tenths of a degree or a few meters difference between each profile plot.

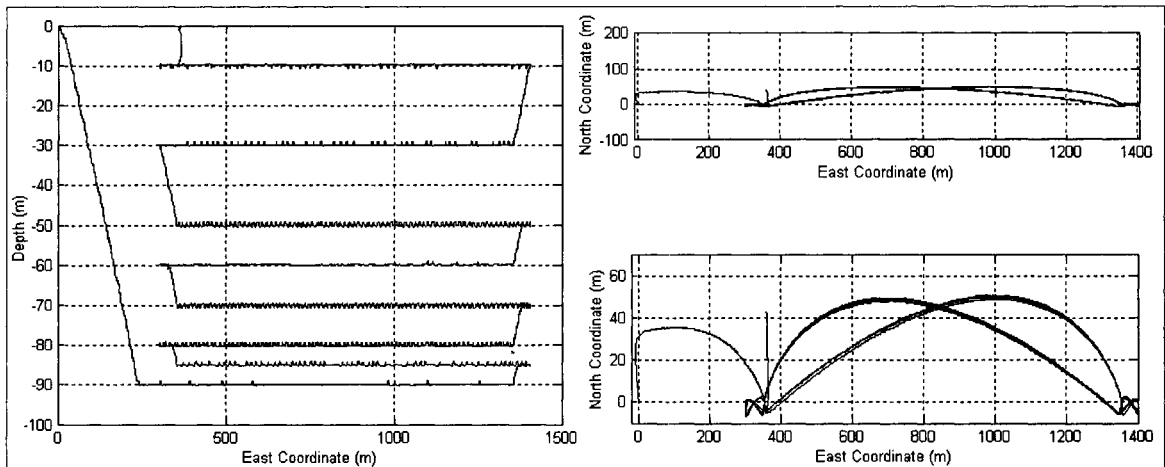
As previously, there is a layer with important variation in the conductivity just below the thermocline. It seems that a significant conductivity variation occurs even with a weak

thermocline, as can be seen on the gradient profile from the third cast, around 10-15m.

The conclusion that can be drawn from the analysis of this data set is the same as that observed with the data set from December 16<sup>th</sup>. More detailed conclusions are discussed later in this chapter.

#### **IV.3.3.2.2. AUV Mission Data**

Because the mission plan was exactly the same as the one used in the previous mission, except for the depth of each leg, the remarks concerning the navigation data are likely to be the same as that detailed in the previous mission analysis. Figure 50 shows the AUV trajectory in the vertical and horizontal planes.



*Figure 50: AUV Trajectory in both the Vertical and Horizontal Planes  
(from Dead-Reckoning Navigation and CTD Depth)*

As in the previous mission, the AUV correctly performed the pattern it had been programmed for, as far as its vertical path is concerned. The same kind of noise on the depth measurements is still evident. Again, it appears that the vehicle was somehow able to sense its drift and tried to correct it. As explained before, the way the mission plan is programmed using relative waypoints needs to be modified for the next mission.

#### **IV.3.3.2.3. AUV CTD and DVL/ADCP Data**

Because of the problems of data conversion of the CTD and DVL/ADCP measurements discussed in chapter IV.3.3.1.3 and IV.3.3.1.4 there is no use in detailing here the analysis performed on these pieces of data. As explained previously, we were unable to correct the errors in the temperature measurements, which was our main concern, and most of the ADCP data had not been logged.

#### **IV.3.4. Conclusions**

The conclusions and lessons learned from these two missions are that:

- The vehicle performed the mission it had been programmed for well, although the depth control was probably not as efficient as possible, because of the noise in the depth measurement.
- Unfortunately, because of many bugs in the data reading and logging, the data gathered by the AUV during these missions are of poor interest as far as the observation of the water slice properties is concerned. Moreover, as explained before, the thermal structure of the water slice was not as interesting as expected, because of the lack of significant variation in the thermocline.
- Several things were found on the AUV that needed fixing, namely the problems that caused the CTD data to be so noisy, and the water current profile not to be logged correctly.
- Ideally, the AUV software should be investigated to find the reason why the AUV altitude was reported as zero when the seafloor was out of range of the DVL. It would be better to set the altitude to the maximum range of the DVL in such a case.

- The tracking system really needs to be thoroughly tested, and either fixed or replaced, in order to obtain reliable tracking data on further missions.
- The mission should be written in a different way so that the vehicle wouldn't try to correct the navigation for any current it may sense. Indeed, we do want the vehicle to remain in the same water column, only crossing it along the East-West direction and not performing some strange patterns while trying to correct for a sense of the current that cannot be but wrong.
- We should also write the mission in a way that would make it easier to locate the AUV when it surfaces.
- Finally, for vehicle safety matters, we have to make sure that any new device integrated in the vehicle is fully pressure tested, and if possible, that every pressure vessel is provided with a leak sensor.

These conclusions were taken into account while preparing the next mission.

#### ***IV.4. March 2003 Missions***

The March 2003 Mission was the same as the previous ones in terms of motivations and requirements. The goal was to successfully acquire the data we were unable to acquire satisfactorily previously. Moreover, this time, we also wanted to log the data provided by an upward looking ADCP and a second CTD that were to be integrated in the AUV payload. The upward looking ADCP was used to measure the current profile above the vehicle, in order to double the ADCP range. The second CTD was used because it was believed to be more accurate and to sample at a higher frequency than the AUV CTD, and also because no really conclusive test of the AUV CTD had been performed. As for the

previous missions, the AUV carried the Turbulence Package. Hereafter are described the preparation and execution of the mission, followed by the data analysis.

### **IV.4.1. Preparing the Missions**

The preparation of the mission required firstly the problems discovered during the previous missions to be fixed, and an additional task: the integration of a SeaBird Electronics (SBE) CTD and an upward-looking ADCP. Both sensors had to be integrated with the AUV so that they take power from the vehicle batteries and communicate data to the main computer logger. The integration of these sensors is described hereafter, followed by a brief discussion of the pool test and vehicle trimming, the sensors calibration, and the writing of the mission.

#### **IV.4.1.1. Fixing the Problems Revealed by the Previous Missions**

The problems revealed during the previous mission had been fixed: The DVL data was then correctly read by the main computer since the NV conversion mistake was corrected. Similarly, the CTD data conversion problem was fixed and a few tests were performed. Unfortunately, since these tests had not been performed over the full depth range of the planned mission, we were not able to check in a conclusive manner that everything had been fixed. Finally, the tracking system was investigated and fixed, and the whole system was successfully tested.

### **IV.4.1.2. Sensor Payload Integration**

As explained previously, the only link between the AUV tail and the sensor payload is the 48VDC power supply and LonTalk network. That is, the tail sees a load consuming some power, and some Neuron devices that demand and provide a number of network variables. The integration, in terms of electronics and software, of the new CTD and ADCP then required:

- Conversion of power from the main bus to the ADCP and CTD,
- Logical integration of the sensors in the distributed software, which can be further divided into:
  - Local control of the sensors by Neuron nodes,
  - Integration of the sensor nodes in terms of software.

Both the physical connection and the logical integration tasks are described hereafter.

#### **IV.4.1.2.1. Physical Connection of the Sensors to the Payload Bus**

The upward looking ADCP and the SBE CTD had to be connected to the payload bus through a device that transforms the power supply and data channels as required. Both sensors communicate through RS232C ports. The ADCP requires a 48VDC power supply, while the SBE CTD needs an input voltage of 12VDC.

The upward looking ADCP had been previously used with the OEX-B which makes use of the same LonTalk network and power bus. Thus, there already existed a device that connects the ADCP to that bus. This device takes the 48VDC input, and powers the ADCP and a Serial to LonTalk Adapter (SLTA). This adapter bridges the LonTalk channel with the RS232C DVL data port. The device is housed in a pressure vessel that

has three bulkhead connectors. The third connector, previously unused, now connects the SBE CTD to the payload bus. The advantages are:

- The use of one pressure vessel instead of two reduces the occupied space and the chance of water leak,
- While we modified the electronics inside the pressure vessel, we could take this opportunity to add some health sensors,
- The inside connection eliminates the need for an external splitter.

The electronics inside that pressure vessel was rewired to include another SLTA for the SBE CTD. This SLTA is based on a High Performance Standard Node series 2 (HPSN2) developed by the Seatech's Electronics Laboratory. This part has been chosen because it is powered by 48VDC and communicates with LonTalk on one side, while on the other side, it offers, among others, the 12VDC regulated power supply and RS232C full duplex port we need. Moreover, the HPSN2 also includes temperature, pressure, humidity and leak sensors. The rewired circuitry, that physically links the sensors to the main bus of the vehicle, is presented in Figure 51.



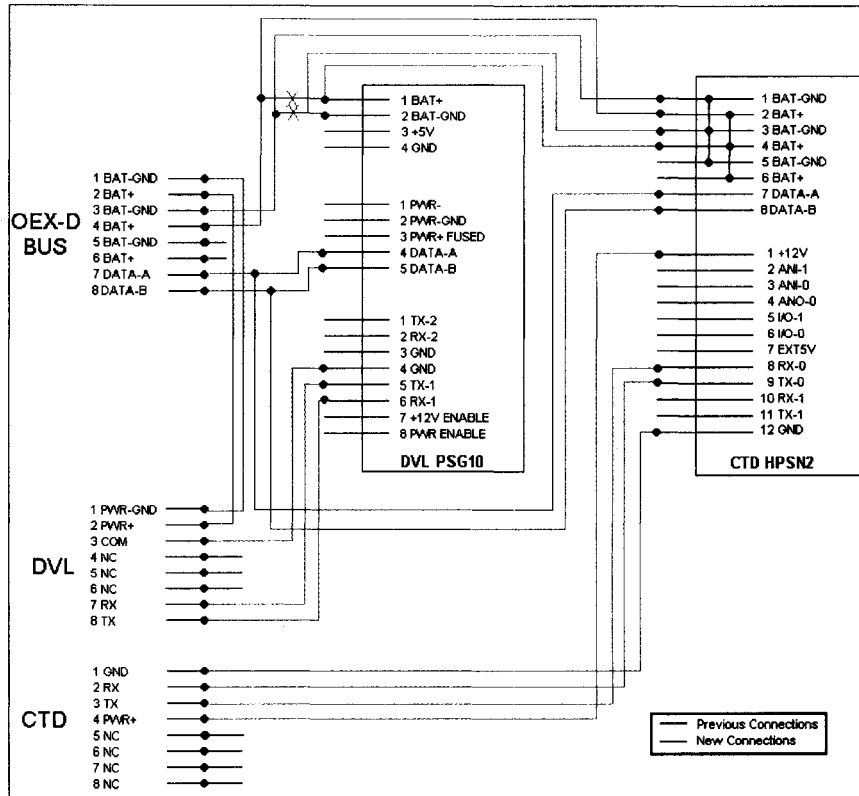


Figure 51: Upward Looking ADCP and SBE CTD Payload Interface Module

#### **IV.4.1.2.2. Logical Integration of the Sensor Nodes in the Software**

As previously described, the logical integration of the sensors required the implementation of the local logic in the sensor nodes, and the interfacing of each node with the main computer software.

Since the upward looking ADCP had previously been used on an OEX-B, there already existed the node application software. It was necessary to interface that node with the main computer. This task has already been discussed in details and is not reported again here. To summarize, the vehicle can now carry two similar DVLs that can both be switched to ADCP mode. The only difference is that the tail DVL data is also used for navigation, while that coming from the upward looking ADCP is simply logged.

Then, as far as the SBE CTD was concerned, the node application had to be written, then interfaced with the other nodes in the vehicle.

In the SBE CTD neuron node, the local logic application is responsible for:

- The monitoring of the health sensor and the assignment of corresponding NVs,
- The configuration of the sensor,
- The parsing of the data from RS232C strings to NVs values,
- The implementation of the VRS232.

To implement these functionalities, it would have been an easy solution to simply modify the Neuron C code used with the AUV CTD, but this was impossible, mainly because of too many differences in the CTDs interfaces. It was then decided to completely rewrite a new application code. The application for the CTD node works as follows:

When the node is reset, it turns on the power to the CTD, configures the serial port, and sends to the serial port the required command strings to configure the sensor. Then, whenever data is received on the serial port, it is either directly copied in the VRS232 NV if the VRS232 mode has been enabled, or parsed and copied in the CTD data NVs. Every second, the health sensors reading is triggered by a timer. The health sensors outputs are sampled, converted and copied in the corresponding NVs. Finally, some procedures are implemented to reset the CTD in case of communication problems. The node application is not described further here.

Once the neuron application was written, it offered the NV interface presented in Table 12.

<i>NV</i>	<i>Writer(s)</i>	<i>Reader(s)</i>	<i>Purpose and comments</i>
PayldCTDData	Payload CTD	Host application	CTD data
PayldCTDhlth	Payload CTD	Host application	CTD node health
error	Payload CTD <sup>(1)</sup>	Host application	CTD node error
dropWtDrop	Payload CTD <sup>(2)</sup>	Dropweight	Command to drop the weight in case of leak
PayldCTDModeChg	Host Application	Payload CTD	NVs used for the implementation of the VRS232
vrs232CharsOut	Payload CTD <sup>(3)</sup>	Host application	
vrs232CharsIn	Host Application	Payload CTD <sup>(3)</sup>	
mainExecTime	Payload CTD	None	Only used to debug the timing, later removed
hlthExecTime	Payload CTD	None	

Table 12: Payload CTD Network Interface

(1) This NV is also written by every node in which an error-checking procedure is implemented.

(2) This NV is also written by every node in which a leak-detection procedure is implemented.

(3) These NVs are read/written by every node in which the VRS232 is implemented.

Once the neuron node was configured, we checked its timing and found that the parsing of the serial data was approximately taking 10 to 20 ms, while the health monitoring was taking 40ms, which is acceptable.

The NVs were then interfaced with the other nodes using the method described in III.2. Particularly, on the main computer side, the raw CTD data is processed by ProcessData which converts it into real values using the appropriate formulas. Once each field is converted, the depth is computed using the equation [52]:

$$d(m) = p(dBar) \times 1.019716 \quad (\text{Eq. 4.1})$$

To check that the data reading was correct, we measured the temperature, and compared it against the content of the NVs, as browsed using LonMaker, and the converted value in shared memory as monitored by the AUV monitor server. We also monitored the health of the neuron node to ensure that the temperature inside the pressure vessel was not rising

too much, and that the pressure was not rising after the pressure vessel had been closed with half an atmosphere depression inside, which would indicate a leak.

In the end, from the operator point of view, the sensor is configured to automatically output its data at 2Hz, averaging 8 of the samples taken at 16Hz. Only the conductivity, temperature and pressure are outputted. The sensor is configured to wait thirty seconds after a conductivity threshold has been reached before turning on an internal pump that creates a water flow around the probes to improve their accuracy.

As everything seemed to be working, the CTD and the neuron node pressure vessel were mounted in the nose payload of the OEX, together with the upward looking ADCP, Turbulence Package, dropweight, light strobe, and a tracking beacon.

#### **IV.4.1.3. Sensor Calibration**

Since the vehicle had been modified by the addition of new sensors that are likely to create some magnetic perturbations, it was necessary to recalibrate the magnetic compass so as to achieve adequate navigation performance. This was done as described in IV.3.1.2. As far as the SBE CTD is concerned, it had been recently calibrated in factory, and had not been used since. No recalibration was therefore required. The calibration of the DVL and/or ADCP has been discussed in IV.3.1.2

#### **IV.4.1.4. Pool Test and Vehicle Trimming**

As before any mission, the vehicle was tested in the seawater pool at Seatech. After having checked the health of the AUV, we ran a short mission to check the logging of the two DVLs/ADCPs and the two CTDs data. Because there was a lot of oil in the pool, we

did not remove the cap of the the pipe leading to the conductivity probe and the pump of the SBE CTD. A quick analysis of the data showed that both DVLs had been successfully switched to ADCP mode, and that the current profile had been logged. As far as the CTDs are concerned, the AUV CTD did not seem to show any problem of noise, and the SBE CTD gave some meaningful depth and temperature reading that were comparable to that of the AUV CTD. The SBE CTD node health was also examined to ensure that everything was fine. Then the vehicle was trimmed, as detailed in IV.3.1.3.

#### **IV.4.1.5. Writing the Mission**

Since this mission was very similar to the previous one, it was written based on what was done last time. Nevertheless, considering the conclusions of the previous mission, we changed the way the navigation was programmed so that the vehicle would not try to correct its trajectory for any current it may sense. The macro instruction used to run a set of legs of the survey was modified so as to use heading commands and duration criteria instead of waypoints. The purpose of that modification was to ensure that even if the vehicle sensed a current, it would not try to correct its trajectory, because it would not be trying to reach a waypoint but to keep a constant heading. Doing so, the vehicle would always remain in the same water column, drifting with it.

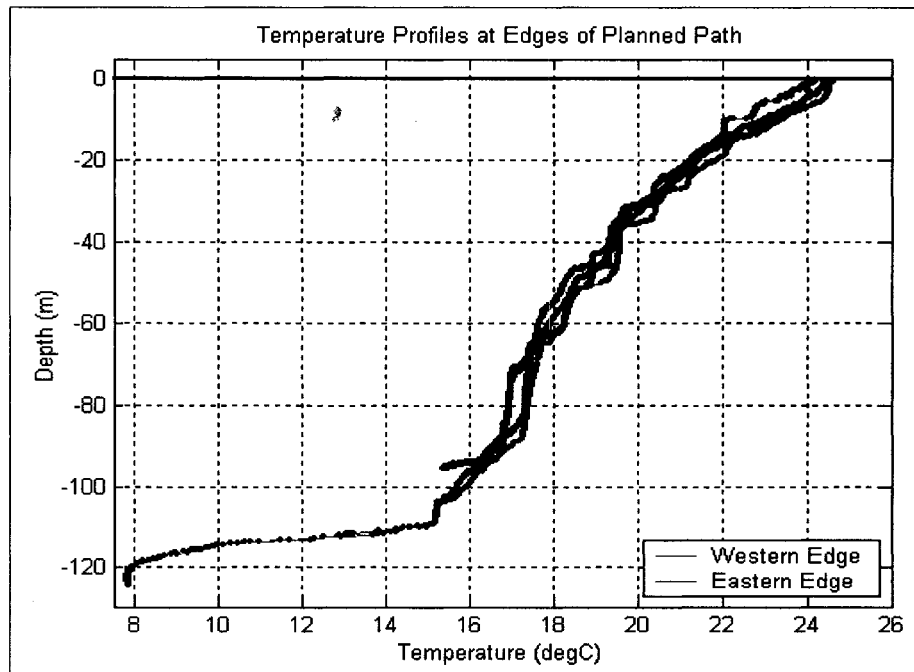
As described in IV.3.1.4 for the previous mission, the mission plan consisted of the required command to switch both DVLs to ADCP mode, a GPS Fix, several calls to the “Vertical Leg” macro, and a final GPS Fix. We checked in the logger input file that every variable to be logged was listed, with the appropriate logging parameters.

Because of the difficulties we encountered last time in locating the vehicle once it

surfaced, we modified the last GPS Fix macro used in the mission so that the vehicle keeps the antenna up and continues to update its estimated position with GPS fixes forever. A second mission was then written, the purpose of which was solely to lower the antenna. This mission would be started through the modem once the vehicle would be located.

#### **IV.4.2. Mission Execution**

The mission was run on March 19<sup>th</sup>, 2003. We left the dock and proceeded to the eastern edge of the mission location (26°4.4'N, 80°3.0'), where we performed a first CTD cast to get the temperature profile of the water column. Then we went back to the western launch point (26°3.6'N, 80°4.0'W) and performed a second CTD cast. Both temperature profiles are presented in Figure 52.



*Figure 52: CTD Casts at Edges of Mission Location before Launching the Vehicle*

Unfortunately, during the first CTD cast, the current acting on the rope prevented the CTD from going all the way down to the bottom. For the second cast, some weight was added to the instrument so that it was less sensitive to the current. Based on the information provided by these casts, it was decided to run 10 equally spaced legs between 100 and 10m, plus two more legs at 5 and 2.5m. Then the vehicle was launched, and the mission started through acoustic modem. This time, the tracking system was working correctly, but unfortunately, the computer used for tracking crashed, and the data was lost. The mission went on without any other troubles, and the vehicle surfaced 3 hours later. A third CTD cast was then performed at the recovery point (26°7.9'N, 80°2.1'W).

### **IV.4.3. Data Analysis and Results**

Every piece of data was analyzed separately first, then compared together where appropriate. First the data obtained from shipboard CTD cast was examined, then the data describing the AUV mission, the AUV CTD data, and finally the ADCPs current profile were considered.

#### **IV.4.3.1. Shipboard CTD Data**

Again, a detailed analysis was performed on the shipboard CTD data, of which only the significant results are summarized here. Only the part of the data corresponding to the CTD going down was used, as explained in IV.3.3.1.1. All three variables (conductivity, temperature and depth) were sorted, then filtered using a Moving Average (MA) filter. The vertical profiles for temperature and conductivity are presented in Figure 53.

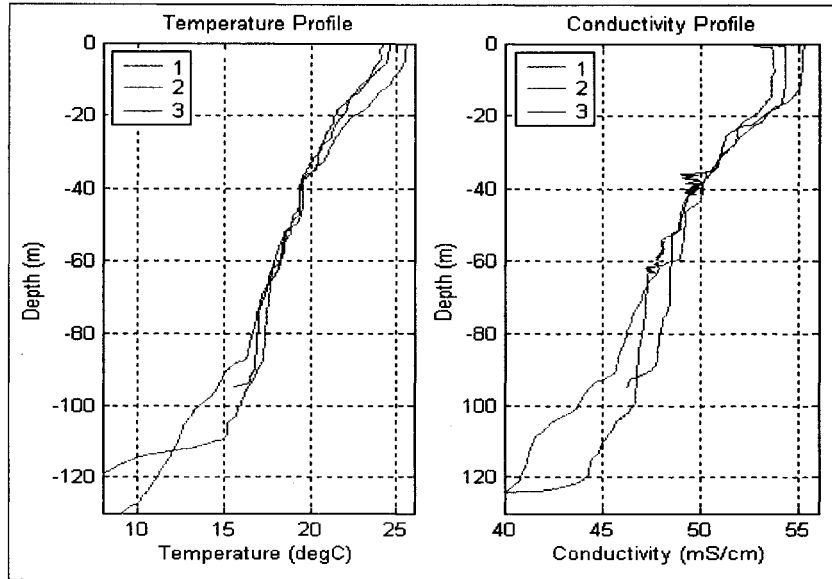


Figure 53: Temperature and Conductivity Profiles (March 19<sup>th</sup>)  
 (1): 26°4.4'N, 80°3.0'W, 16:30 UTC, (2): 26°3.6'N, 80°4.0'W, 17:00 UTC,  
 (3): 26°7.9'N, 80°2.1'W, 22:00 UTC

Once the data was filtered, the derivatives of temperature and conductivity with respect to depth were taken (Figure 54).

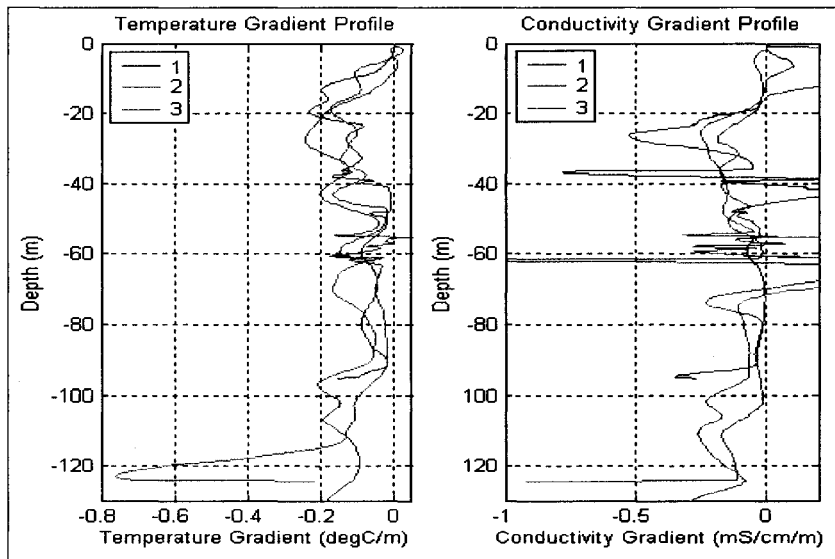


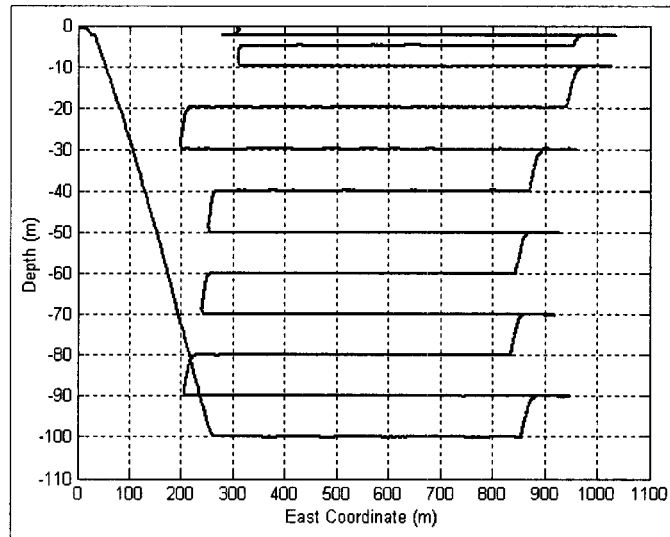
Figure 54: Temperature and Conductivity Vertical Gradient Profiles (March 19<sup>th</sup>)  
 (1): 26°4.4'N, 80°3.0'W, 16:30 UTC, (2): 26°3.6'N, 80°4.0'W, 17:00 UTC,  
 (3): 26°7.9'N, 80°2.1'W, 22:00 UTC



Unfortunately, these profiles are not as interesting as expected. Here, the whole top of the water column is almost a thermocline. There is not, as that was the case during the experiments we performed in winter, a small shallow thermocline over a thick mixed layer and then a strong main thermocline. Here, there only is a very thin mixed layer, due to the action of the wind on the surface, and then the shallow thermocline extends down to 40m, and combines with the main thermocline around 50m. This is confirmed by the plot of the gradient profiles, which shows relatively regular profiles, with no localized region of higher gradient. In a same manner, the conductivity gradient profile does not show any particular pattern comparable to the ones observed on the previous data sets. Moreover, once again, there doesn't seem to be any significant variation in the temperature profile with the horizontal distance, as far as the top 100m are concerned. The second cast seems to show a quite strong ( $0.8^{\circ}\text{C}/\text{m}$ ) thermocline around 110m, but this may be only due to the seafloor being close. The third plot rather shows a time-dependent variation, with the diurnal variations of the temperature in the first meters of the water column as it is warmed by the sun radiation. The profile around 110m differs significantly from that obtained from the second CTD cast, but again that may be explained by the seafloor being close to the bottom of the second cast. This profile is not as interesting as one showing a localized strong thermocline. Indeed, if we consider the tracking of the thermocline, it is impossible with such a profile to tell which part should be tracked. Overall, this profile shows some interesting things about the combination of different types of thermoclines, but unfortunately does not bring the kind of information we needed to improve our knowledge for thermocline tracking purpose.

### IV.4.3.2. AUV Mission Data

Again, in order to estimate how well the AUV performed the mission it had been programmed for, its recorded trajectory was analyzed, and compared to what was intended. First, we considered the vertical trajectory (Figure 55), obtained by projecting the 3D path computed by the position estimator into the East-Up plane.



*Figure 55: AUV Vertical Path from Dead-Reckoning Navigation and CTD Depth*

This time again, the pattern corresponds to what was expected, and what's more, there doesn't appear to be any significant noise on the depth measurement. Indeed, closer analysis of the depth measurements showed a few smooth variations of the order of 10cm, which is totally acceptable.

It is then interesting to look at the horizontal trajectory (Figure 56), as measured by the position estimator, with respect to a reference layer of water.

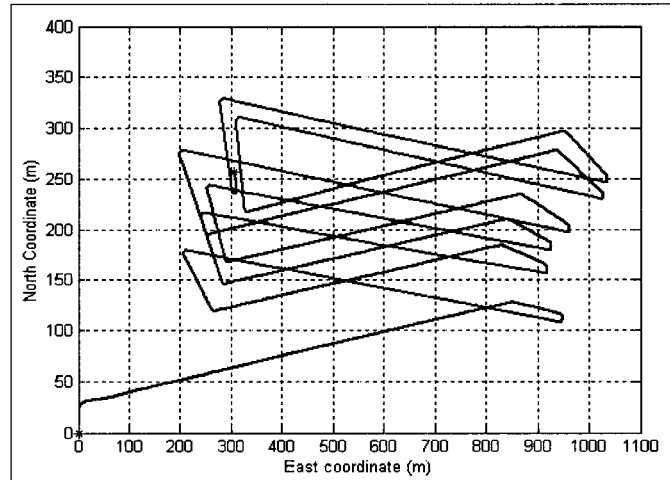


Figure 56: AUV Horizontal Path from Dead-Reckoning Navigation

The pattern shown here is much better than was obtained during the previous missions. Indeed, even if the vehicle still sensed a drift, the straight lines for each segment of the mission show that this drift was not taken into account for the navigation. The vehicle heading data shows that during each leg, the heading remained within half a degree of the commanded value. It is interesting to get an idea of the real path of the vehicle, taking into account the drift caused by the current. To do so, we assume that the current was approximately constant during the whole mission. Then the difference between the vehicle position estimated by dead-reckoning and its real position tells how much it has drifted since the beginning. The only point where both positions are available is when the vehicle surfaces. Just before it gets a GPS fix, the position is that obtained from dead-reckoning, which is corrected when the fix is available. The displacement between the two successive points is obviously neglected, given the logging frequency of the vehicle position (8Hz). Then the difference between positions gives the distance drifted, which divided by the duration of the mission in turns yields the mean drift velocity. This drift velocity can then be used to correct each position.

This is summarized by the following equations:

$$\vec{V}_{Drift} = \frac{\vec{X}(t=t_{GPSfix}) - \vec{X}(t=(t_{GPSfix} - \Delta T))}{t_{GPSfix}} \quad (\text{Eq. 4.2})$$

$$\vec{X}_{Real}(t) = \vec{X}(t) + t \cdot \vec{V}_{Drift} \quad (\text{Eq. 4.3})$$

Where  $\vec{V}_{Drift}$  is the drift velocity,  $\vec{X}(t)$  is the position at time  $t$  as computed by the position estimator,  $\vec{X}_{Real}(t)$  is the estimated real position at time  $t$ , corrected for the drift,  $\Delta T$  is the logging period, and  $t_{GPSfix}$  is the timestamp of the first GPS fix when the AUV surfaces.

Once  $\vec{V}_{Drift}$  is obtained, the mean current velocity and direction are known.

This gives the path presented in Figure 57.

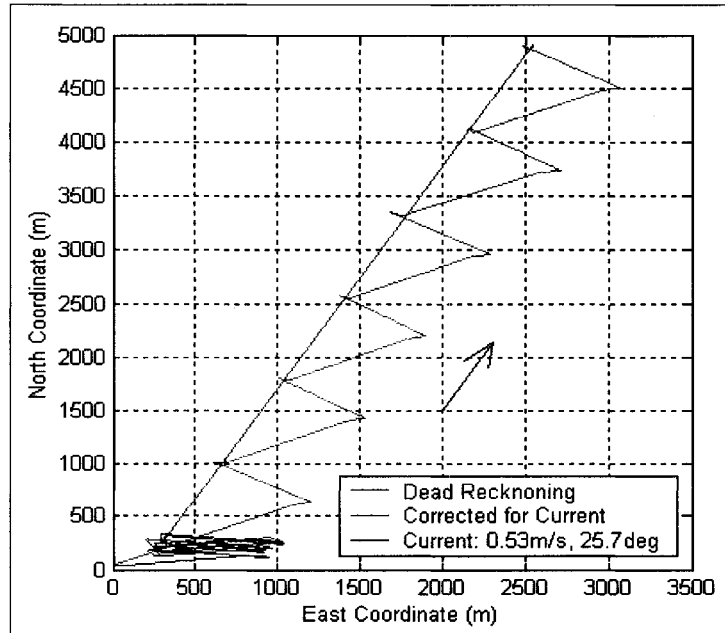


Figure 57: AUV Horizontal Path, Corrected for Current-Induced Drift

This plot indeed looks like what was seen on the tracking system computer before it crashed. Moreover, the mean current obtained by this method is really close to what was measured by the shipboard ADCP, from which a mean North component of the current around 0.5m/s was observed.

#### IV.4.3.3. AUV CTD Data

The plot in Figure 58 shows the raw data for the depth and temperature as functions of time.

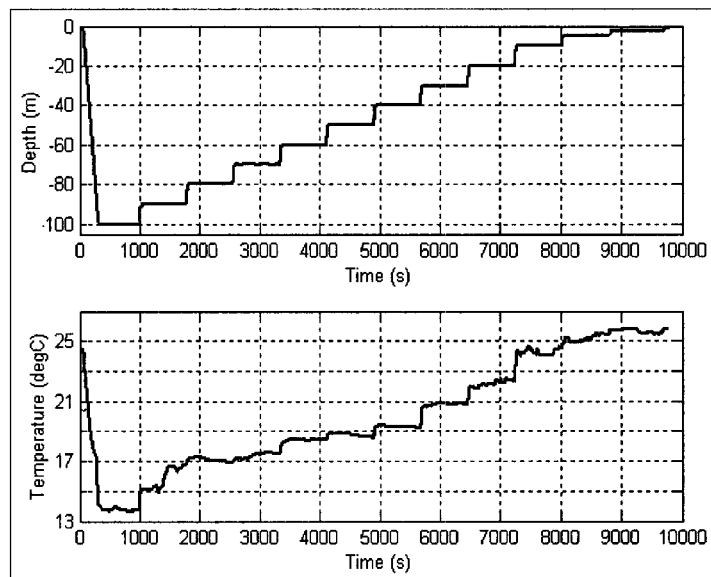


Figure 58: Raw Depth and Temperature Data from AUV CTD Sensor

Here the temperature data is much more reasonable than what was obtained during the previous missions, and is closer to what has been observed from the shipboard CTD casts. From this data it is then possible and meaningful to plot a temperature profile (Figure 59).

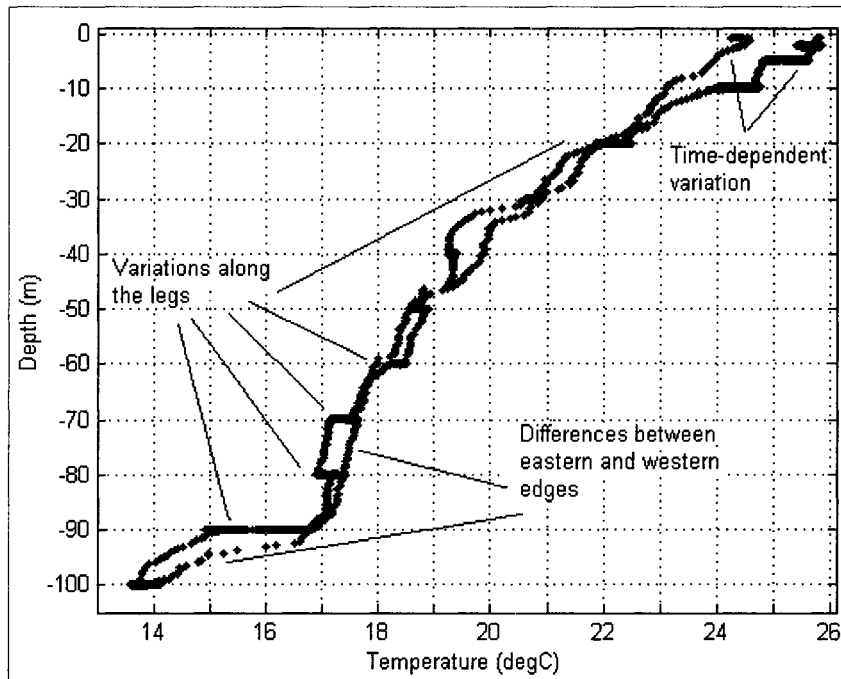


Figure 59: Vertical Temperature Profile of the Water Column from AUV CTD Data

This plot is interesting because it emphasizes the variation of the temperature with both horizontal distance and time. At each depth where a leg was run, a horizontal segment shows the temperature variation along the leg. Significant differences are noticeable on the segments joining the 100m and 90m legs, and 80m and 70m legs. Because these legs were run less than an hour after the first diving, the variations can not be time-related. Moreover, the vehicle climbs between these legs on the East side of the water column, while it has dived on the West side. The segment separation then really shows the temperature difference between the edges of the water slice. On the other hand, on the upper part of the plot, the separation between the first profile when the vehicle dives to run the deepest legs, and the other part of the profile when the vehicle climbs up shows the temporal variation, which is particularly obvious closer to the surface, since the time interval approaches 3 hours.

Now it is interesting to try to reconstruct a temperature map, using the vehicle position and the CTD temperature history. Creating a grid of position on an East-Up plane based on CTD depth measurements and horizontal coordinates from the position estimator, and interpolating the temperature measurements over that grid yields the temperature map of the water column in Figure 60.

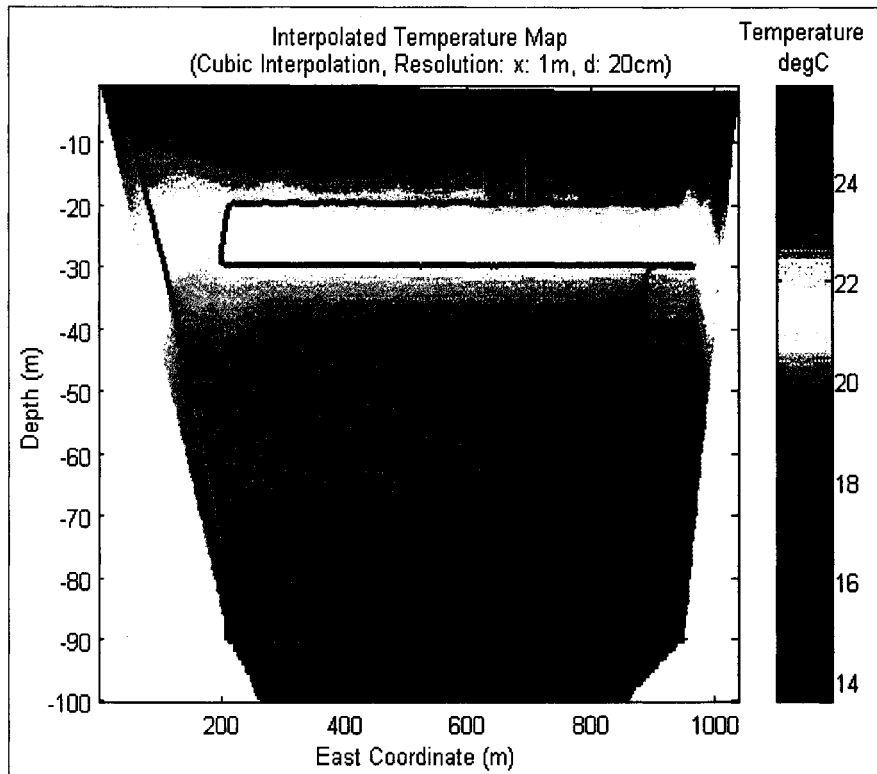


Figure 60: Temperature Map from AUV CTD Data

Although the side boundaries of the map are arguable, the middle of the plot shows small horizontal variations of the temperature, particularly above 20m and below 60m. This map shows a kind of compression of the temperature variation on the East side of the water column, where the cold water is shallower and the hot water deeper than on the West side.

Nevertheless, because no important variation was observable in that water column during that experiment, and because of the absence of a strong localized thermocline, the use of that map is limited. But this at least validates the method used for the mission and the processing technique.

#### IV.4.3.4. AUV DVL/ADCP Data

This time the whole profiles over twice 16 bins has been correctly logged. This gives the current profile of a 32 bins water column that extends 24m above and below the vehicle.

Figure 61 shows the total magnitude of the current, taking into account the raw magnitudes along three axis.

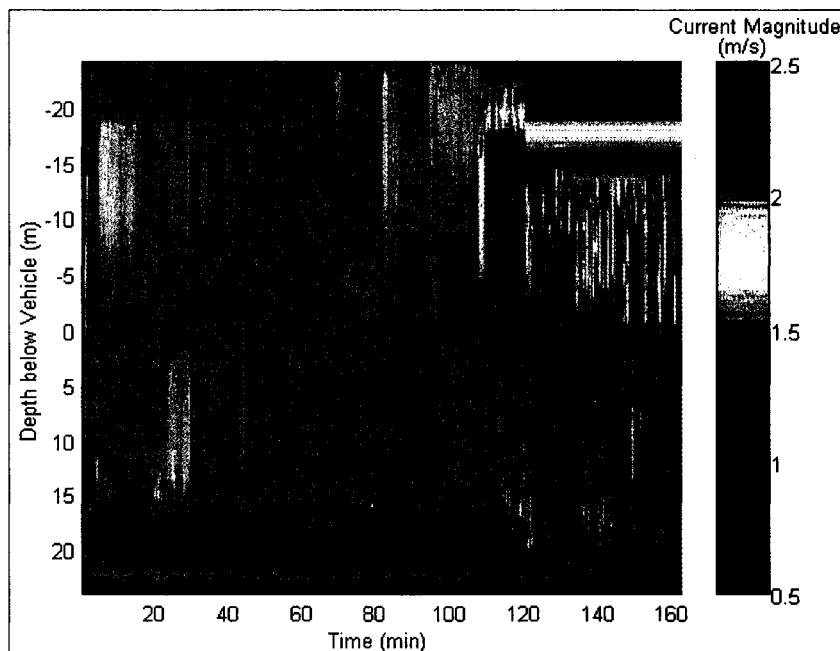


Figure 61: Raw ADCP Current Magnitude Profile as seen from the Vehicle

The current velocity indicated in the above plot is relative to the vehicle. To obtain the absolute ground velocity, the platform velocity has to be subtracted. The magnitude of the platform motion is computed from the two components of the horizontal velocity



estimated as detailed in IV.4.3.2. The vertical component of the vehicle velocity is neglected since it is zero most of the time.

Moreover, the dataset has to be filtered to remove wrong samples. This filtering is performed using the error information provided by the ADCPs. Based on that error indication, a simple binary filter is designed to remove the wrong samples, by comparison of the error with a chosen threshold, which produces the mask presented in Figure 62.

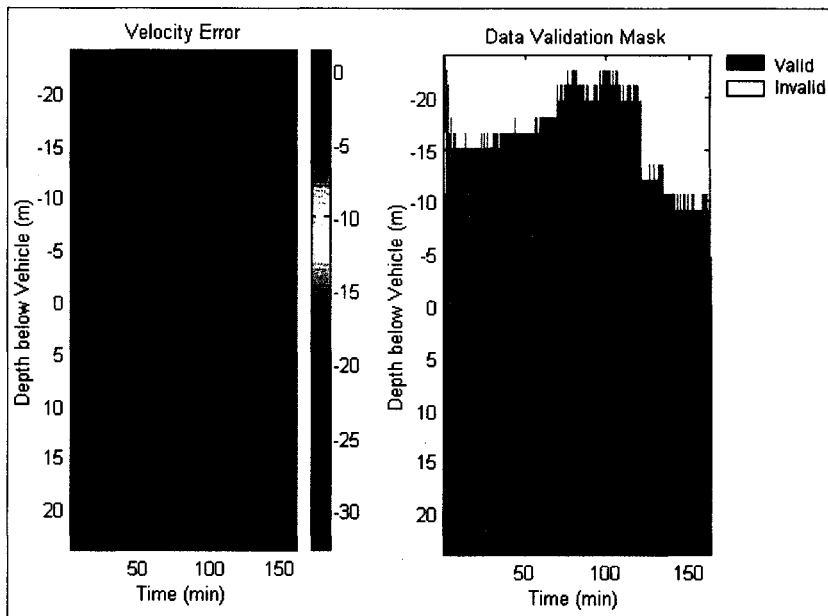
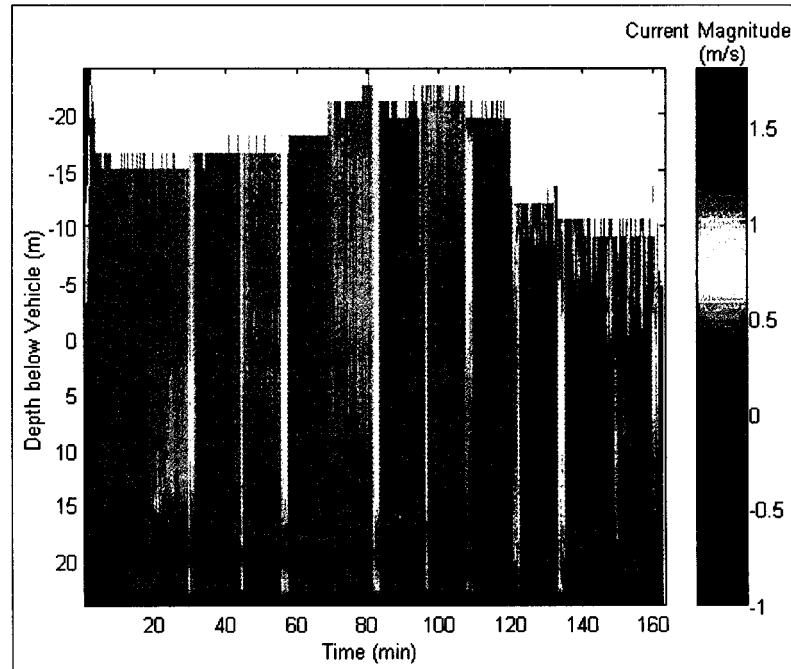


Figure 62: ADCP Velocity Error and Corresponding Validation Mask

The above figure shows that over most of the profile, the velocity error estimated by the ADCPs is relatively small and constant. Only the upper part of the profile presents a more significant error. On the right hand side of the plot, the error is due to the acoustic reflections of the side lobes on the nearby surface. Around 110 minutes, the surface is 20m above the upward-looking ADCP, the range of which is then decreased to 18m (12 bins), then 9m (6 bins) when the vehicle is at 10m from the surface (around 130 min), and almost zero after. On the other hand, there is no such explanation for the error in the 7 top

meters of the profile during the first 100 minutes.

Now it is possible to plot the absolute magnitude of the current (Figure 63), of which only the correct samples are kept.



*Figure 63: Absolute Current Velocity Magnitude*

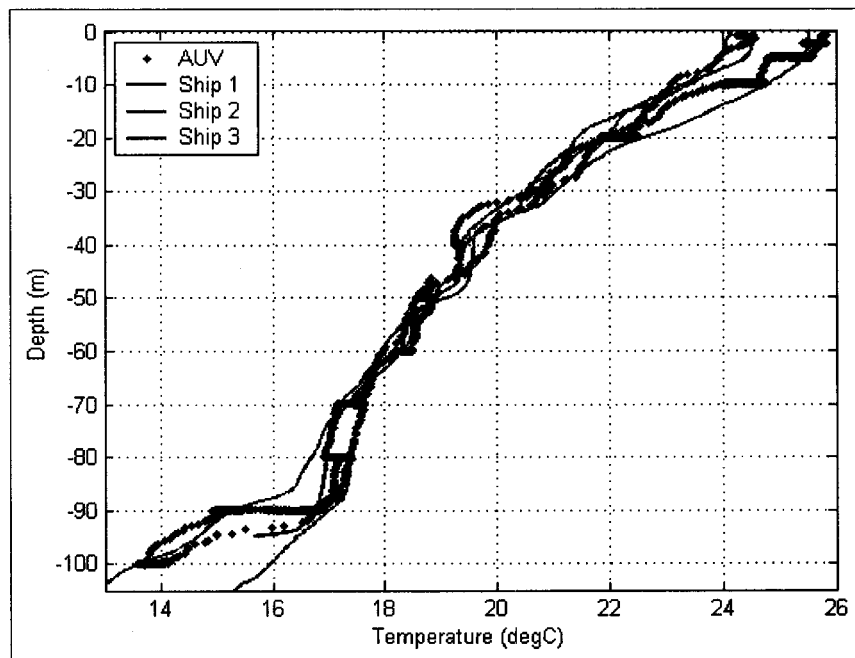
This figure shows a non-homogeneous current. The pattern showing a regular alternation between velocities around 0.3 and 0.5 m/s indicates that the platform motion has not correctly been removed. To achieve a better compensation of the vehicle motion, it would be necessary to have accurate information about the vehicle position. The correction performed here is only based on data from dead-reckoning, corrected for a drift assumed regular during the three hours of the mission. This method obviously does not provide an estimate of the vehicle velocity accurate within 20cm/s at every instant. This explains the 0.2m/s discrepancy between each segment of the above profile.

Nevertheless, this still gives an estimate of a mean current around 0.5 m/s, which

corresponds to what has been estimated from the vehicle drift in IV.4.3.2, and what was observed from the shipboard ADCP. Moreover, this does not show any significant current magnitude variation with the depth. But anyway, because as explained before there was no strong localized thermocline in that water column, no significant current variation possibly related to a thermocline was likely to be observed.

#### **IV.4.4. Multiple Datasets Comparison**

Once every piece of data had been analyzed separately, it was interesting to compare some of them together. First of all, the water slice temperature profile measured by the AUV was compared to that obtained from shipboard CTD cast (Figure 64), to verify the comments and interpretations discussed in IV.4.3.3.



*Figure 64: Temperature Profiles Comparison*

*(Profiles Measured by the AUV and the Shipboard CTD)*

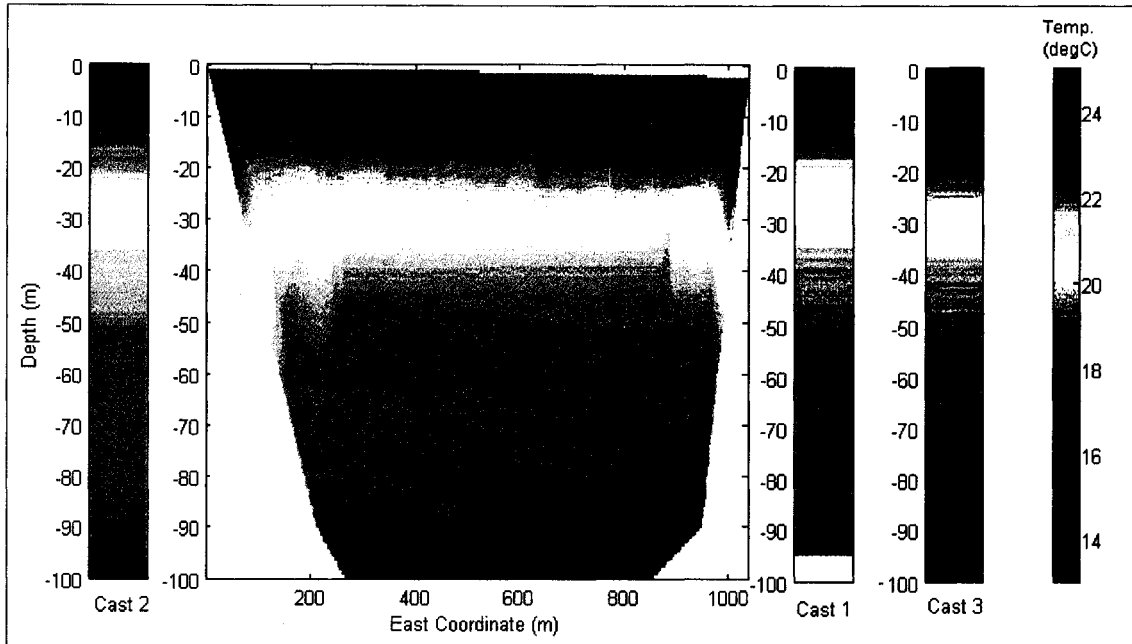
*(1): 26°4.4'N, 80°3.0'W, 16:30 UTC, (2): 26°3.6'N, 80°4.0'W, 17:00 UTC,*

*(3): 26°7.9'N, 80°2.1'W, 22:00 UTC*

Overall, all the temperature profiles have the same shape. Particularly, between 80m and 70m for instance, the above plot confirms that the AUV CTD has captured the temperature variation between the edges of the water slice, as discussed in IV.4.3.3. The profile measured by the vehicle when it dives at the western edge of the water slice matches that from the western CTD cast, whereas, when the vehicle climbs, at the eastern edge, the profile matches that from the eastern cast. In a same way, the time-dependent variation discussed in IV.4.3.3 is confirmed here because, when the vehicle dives, the profile approximately corresponds to the cast performed just before the vehicle was launched, and when it surfaces, the profile matches that from the cast performed when the vehicle was recovered.

Nevertheless, some discrepancies appear, which may be explained by offsets in the temperature and pressure sensors of both CTDs. To be able to achieve a correct comparison between the data from both CTD, the sensors would probably have to be calibrated, particularly the AUV CTD, that has not been calibrated for years.

We can also try to compare the reconstructed temperature map (Figure 65) with the profiles measured from shipboard CTD casts.



*Figure 65: Temperature Maps Comparison*

*(from AUV and Shipboard CTD Measurements)*

*(1): 26°4.4'N, 80°3.0'W, 16:30 UTC, (2): 26°3.6'N, 80°4.0'W, 17:00 UTC,*

*(3): 26°7.9'N, 80°2.1'W, 22:00 UTC*

Although it is not easy to make a formal comparison, some similarities are nevertheless noticeable. Particularly, the left hand side of the map, that mainly depends on the temperature sensed by the vehicle when it dives at the beginning of the mission, is really close to the profile shown by the second cast, performed at the launch point. In a same way, the top of the map, rather influenced by the temperature at the end of the mission is similar to the profile shown by the third cast, performed once the vehicle was recovered. In between, it is more difficult to compare the plots because both the spatial variations shown by the two first casts, and the temporal variations shown by the third cast appear on the map.

#### **IV.4.5. Conclusions**

The conclusions that can be drawn from the March, 2003 mission are:

- As during the previous missions, the vehicle performed the pattern it had been programmed for well. Moreover, because this time that pattern was defined differently, the vehicle didn't try to correct its drift, and then really remained in the same water column throughout the whole mission.
- This time, all the required variables were correctly acquired by the AUV. The only piece of data that is missing is that from the tracking system. This information would have been useful to more accurately obtain the absolute path of the vehicle and compute its velocity.
- The different temperature datasets all show a water column with an almost constant gradient profile throughout the first 100m. There is no localized strong thermocline.
- It has been possible to reconstruct a temperature profile and a temperature map from the data acquired from the AUV, and both are relatively similar to what was measured by shipboard CTD casts.
- All temperature datasets show small horizontal variations of the temperature profile, but these variations are not as important as expected.
- The water current profile didn't show any significant vertical variation of the current. Anyway, since no strong thermocline was present, important current variations related to the thermocline were unlikely to be found.
- Overall, this mission proved that the method used for the experiment is valid, and allows the reconstruction of a temperature map, which was the primary goal.

To improve data acquisition and processing, it would be necessary to recalibrate the CTDs together, to allow a more accurate comparison. Again, it would also be desirable to have tracking system data to process.

## ***IV.5. Conclusions***

After these three missions, the overall method for the experiment and data analysis has now been validated. Different problems have been discovered and fixed so that now the experiment can be performed satisfactorily.

These experiments mainly showed that:

- In case there existed a strong localized thermocline, the temperature gradient within that layer was of the order of a few tenths of a degree per meter, with maximum gradient as large as 1 °C/m. The mixed layer showed some temperature gradients approximately 50 times smaller. Sometimes, a shallow thermocline with gradients around 0.1°C was present, but this layer was very thin.
- During the third experiment, the temperature variation in the upper 100m of the water column was unfortunately homogeneous. There wasn't any strong localized thermocline. Thus, the temperature map obtained from that experiment cannot really be used as an input for a thermocline tracking controller simulation. Indeed, this kind of map could be used to test a tracking controller to analyze its behavior in the absence of thermocline to track, but cannot be used for the development of the controller.
- As far as the observation of the conductivity profile of a water column is concerned, it was noticed that a layer of rapid variation of conductivity with the depth was often present below the top of a thermocline. Nevertheless, this was not always the case, and

the magnitude of these conductivity gradients vary a lot from one CTD cast to another. Therefore, no relation systematic enough to be used to help locate the thermocline has been found.

- Finally, as far as the water current profile is concerned, we were interested in knowing if some important current variations related to a thermocline could be observed. Unfortunately, during the only mission when the current profile was acquired, no localized thermocline was present.

From the observations performed during these experiments, it is possible to say that, to obtain significant information:

- A mission plan similar to that of the third mission should be used,
- The experiment should probably be run in Fall or early Winter, so that there exists a strong localized thermocline. Nevertheless, during Winter, the main thermocline is likely to be found deeper, which may be a drawback.

It may be of interest to investigate the possibility of more significant horizontal variations of the thermocline at a different location. Particularly, running the mission throughout a water slice oriented towards the North-South direction may be considered.



## **V. Thermocline Tracking Simulation**

This chapter describes the simulation of the tracking of a thermocline with an AUV. The main requirement, in order to have an AUV track a thermocline, is a controller that maintains the vehicle within the thermocline layer. To facilitate the design and thorough testing of such a controller, a simulation platform is developed, on which various controllers can be tested. The test platform is then used to consider several controllers. Finally, based on the observations of various simulations, some guidelines for further improvements and subsequent implementation of an effective thermocline tracking controller on an AUV are discussed.

### ***V.1. Description of the Problem***

The following section discusses the motivations for the simulation of thermocline tracking with an AUV, and more particularly for the design of a complete simulation tool. A typical thermocline tracking mission to be simulated is then described, from which the requirements for the simulation are derived.

#### **V.1.1. Motivations**

Once again, the goal is to design a controller that would make an AUV follow a thermocline layer in the ocean. For a controller to be sea-worthy, exhaustive tests are required. In order to facilitate both the design and testing of such a controller, significant

time was devoted to various simulations.

Using a simulator, simple situations of thermocline tracking can be performed for the early stage of the design of the controller. Then, once a successful tracking algorithm has been developed, it can be tested on more complex thermocline patterns, so as to improve the efficiency of the controller. Finally, critical situations like the absence or the disappearance of the thermocline can be simulated to check the way the controller reacts in such cases.

To allow an easy design, test and modification of several controllers, it has been decided to create a general test platform. That platform provides a tool that includes all the components required so that a standalone controller can be tested. Although there already exists a complete Hardware In the Loop (HIL) simulator for the Ocean Explorer AUV, it is too complex for the early stage of the development of a controller. Moreover, to allow testing using the HIL simulator, the controller is required to be operational and integrated to the AUV software. Therefore, to satisfy the needs of the early stage of the controller design, it is considered sufficient to have a more basic simulator. Once an efficient control method is worked out, the controller can be integrated to the vehicle software. It will then be possible to use the HIL simulator to thoroughly test and fine-tune the controller before it can be validated for a first at-sea test.

### **V.1.2. Typical Mission to Simulate**

The typical Thermocline Tracking Mission we consider would take place as follows: first, a region of interest as well as an horizontal direction for the tracking are decided. The vehicle is then programmed to run a straight line in the chosen direction. The thermocline

tracking controller only handles the vertical control of the AUV, acting either as a depth controller or a high-level fins controller, depending on the chosen method.

Then the vehicle is launched. It first dives to the maximum predefined depth, while measuring the temperature profile of the water column. Once it reaches the maximum depth, the AUV decides at which depth it has crossed the part of the thermocline to track, based on the temperature profile it just acquired. Once that target depth is defined, the vehicle climbs to reach it.

Then, when the target depth is reached and the vehicle is in the thermocline, the AUV has to travel straight ahead, while tracking the thermocline layer. To do so, given that a thermocline is defined considering the temperature above and below it, the vehicle has to somehow sense the temperature above and below the assumed depth of the thermocline.

One way to enable the AUV to compare the temperature at different depths is to have it follow an oscillating pattern centered on the assumed depth of the thermocline to track. The mean depth of the oscillations then varies with the thermocline.

From the above discussion, the whole thermocline tracking mission is broken down into three phases as shown in Figure 66:

- First phase: The AUV dives to the maximum depth while measuring the temperature, and computes the target depth at which the thermocline is likely to be,
- Second phase: The AUV climbs up and reaches that target depth,
- Third phase: The AUV actually tracks the thermocline, oscillating around its mean depth.

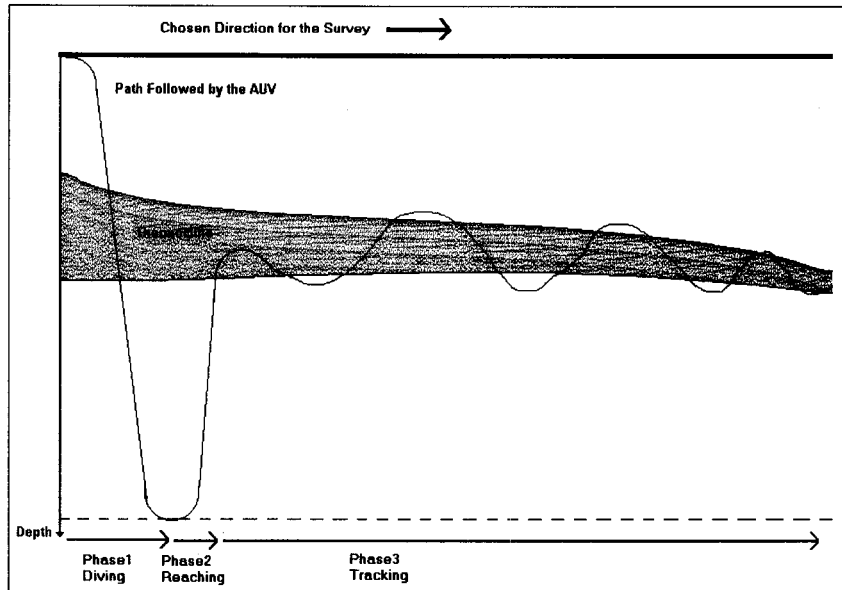


Figure 66: Sketch of a Typical Thermocline Tracking Mission

### V.1.3. Simulation Needs

Based on the considerations described above, the simulation needs can be defined:

- It is only necessary to consider a problem in two dimensions, forward and down. The third direction is assumed to be handled by a heading or steering controller.
- To be able to simulate the tracking of a thermocline with an AUV, it is necessary to simulate a water column showing a thermocline, the AUV cruising within, and a CTD that makes the AUV sense the temperature. It is also necessary to simulate the thermocline tracking controller, which is considered separately.
- To maximize the amount of information obtained from the simulation, it is necessary to store the time history of a number of variables, such as the trajectory of the AUV.
- Finally, it is necessary to choose a platform and environment for the simulation. Here, a simulation is run with Matlab on a PC because it not only allows an implementation

as easy as with many other languages, but also it is a convenient environment for the pre-processing of the required input data, as well as the post-processing and display of the results.

Figure 67 presents a basic overview of the structure of the simulator.

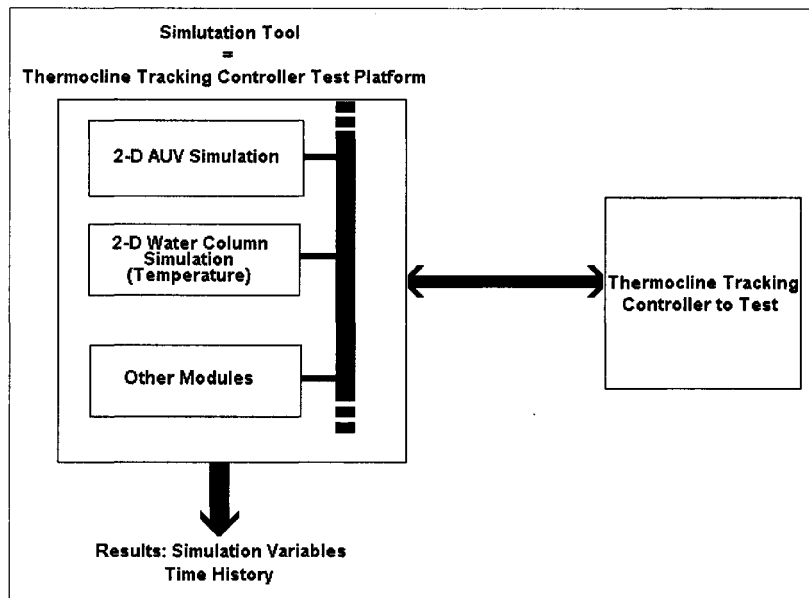


Figure 67: Simulator Structure Overview

## V.2. Simulator Design and Implementation

To allow for an easy design, implementation and modification, the simulation tool is broken down into several modules, each handling a single independent task. The two main requirements are the simulation of the vehicle and that of a water column. The design of the modules that handle these two tasks is discussed hereafter. The other required modules are then listed, followed by a description of their interactions and a summary of their implementation.

## **V.2.1. Method**

The simulation to be designed is based on the simulation of AUV motion throughout a water column based on its temperature characteristics. Therefore, the core of the simulation is composed of an AUV motion simulation, and a water slice simulation.

### **V.2.1.1. AUV Motion Simulation**

For the purpose of the work we intend to do here, it is not required to have an accurate motion simulation. A coarse, yet realistic model is sufficient for the early design of a controller that, once sufficiently efficient, can then be more accurately tested using the HIL simulator. For this reason, the AUV motion simulation is based on a simplified version of the classical Six Degrees Of Freedom (6-DOF) equations of motion summarized hereafter. Before discussing the equations of motion, a reminder on the frames of reference used is given. Then the 6-DOF equations of motion are discussed, followed by the method used to compute the AUV trajectory based on these equations.

#### **V.2.1.1.1. Frames of Reference**

Consider two different frames:

- The geographical or local-level frame, fixed to the Earth, is usually oriented along the North, East and Down directions, as shown in Figure 68. The North-East plane is parallel to the surface of the Earth, and the Down direction is normal to that surface. The frame can be centered anywhere, and the center is commonly chosen at the starting point of a mission. For application of the equations of motion to underwater vehicles, the effect of earth rotation can be neglected, and the earth can be assumed to

be flat [53]. This frame is therefore approximated as a Cartesian inertial frame.

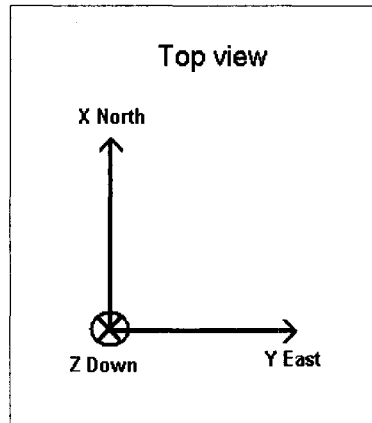


Figure 68: 3-D Geographical Frame

- The body-fixed frame, shown in Figure 69, oriented along the forward, starboard and down axis of the vehicle, can be centered anywhere, but is usually placed either at the center of gravity or at the center of geometry of the vehicle. Although it is easier to write the equations in a frame whose origin is at the center of gravity, it is necessary to account for the fact that this center of gravity is displaced any time the vehicle is modified, for instance by addition of some weight or foam for trimming purpose.

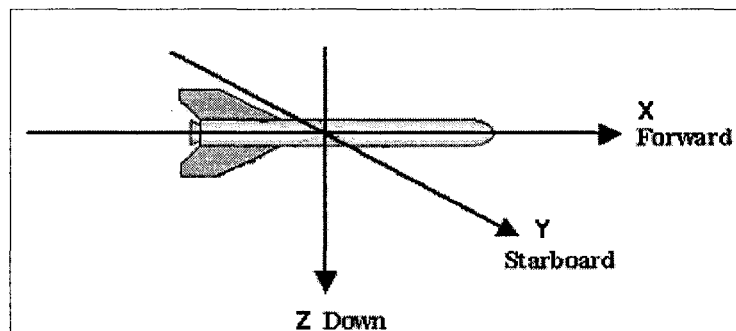


Figure 69: Body-Fixed Frame

In the case discussed here, the problem is 2-Dimensional. We can decide to perform a yaw rotation to use a local-level frame with the x and y-axis oriented towards the

alongtrack and crosstrack direction instead of the usual North and East. Doing so, the  $x$  and  $z$  axis of the body-fixed frame remain in the plane defined by the  $x$  and  $z$  axis of the local-level frame, and we can get rid of the  $y$  axis of both frames, as appears in Figure 70. The center of that local-level frame remains at the starting point.

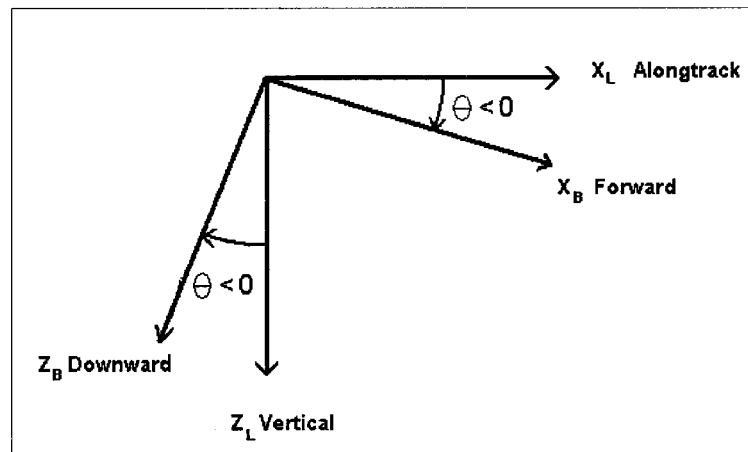


Figure 70: Local-Level ( $x_L, z_L$ ) and Body-Fixed ( $x_B, z_B$ ) Frames for Simulation

*This configuration shows a negative pitch  $\theta$*

The body-fixed frame is obviously not inertial, since it is in translation and rotation with respect to the local-level frame defined above, which is assumed inertial for the needs of that study.

#### **V.2.1.1.2. Equations of Motion**

The vehicle motion simulation is based on solving the simplified, linearized and uncoupled Six Degrees Of Freedom (6-DOF) equations of motion, of which only the longitudinal part is considered. The whole mathematical derivation of the equations [53] is not detailed here. Rather, the method and assumptions used to derive and simplify these equations are summarized.



The derivation of the 6-DOF equations of motion is based on the use of Newton's second law of dynamics for translation and rotation in an inertial frame of reference.

$$\sum f_a = m \cdot \frac{d^2 X}{dt^2} \quad \text{and} \quad \sum \tau_a = \frac{dH}{dt} \quad (\text{Eq. 5.1})$$

Where  $f_a$  and  $\tau_a$  are the applied forces and torques, and  $X$  is the position and  $H$  the angular momentum in an inertial frame.

Considering each particle of the vehicle, and writing the time derivatives of their position and angular momentum, the acceleration and rate of change of angular momentum are expressed as functions of the linear and angular velocities of the body-fixed frame. The expressions are then simplified assuming that the vehicle is a rigid body. Therefore, each particle is fixed in the body-fixed frame. Although this is true for the body of the vehicle itself, in case of a flooded vehicle such as the OEX, the water is moving inside the body. It is nevertheless possible to assume that the motion of the water inside the shell of the vehicle can be neglected. Further assuming no change in the entrained water, the mass of each elementary particle can be considered invariant with time. Therefore, the mass and mass distribution of the vehicle is constant, which allows the simplification of the summation of the equations over the whole rigid body.

These 6-DOF equations for a rigid body can be further simplified by choosing the center of the body-fixed frame at the center of gravity of the vehicle, and considering that the vehicle is symmetric with respect to its major axis. This requires that the body-fixed axes correspond with the axes of inertia. Although this is not always the case when the body-fixed axes are centered at the center of gravity, this is usually considered a reasonable assumption, mainly because of the symmetric shape of the vehicle. These two

assumptions allow to write that the position of the center of gravity in the body-fixed frame is zero, and that the inertia tensor is diagonal. This moreover allows to uncouple the equations for longitudinal and lateral motions, considering only the forward and downward motion, the pitch and the action of the sternplanes for the longitudinal motion. This gives the simplified, uncoupled 6-DOF equations of motion, of which only the 3-DOF part describing the longitudinal motion is further considered.

These equations contain products of dependent variables, therefore they are generally non-linear. Two more assumptions are required to obtain linear equations: firstly, the motion of the vehicle is decomposed in a mean equilibrium motion, or operating point, and a perturbation that accounts for the small dynamic motion about the equilibrium. In the equilibrium condition, the mean velocities, angular rates and mean roll angle are assumed to be zero, except the forward velocity. Secondly, assuming that the velocities, angular rates and attitude angles perturbations are small enough so that their products, squares and higher order terms can be neglected compared to the perturbations themselves yields to the simplified uncoupled linearized equations of motion.

Then it is necessary to estimate the forces and moments acting on the vehicle. These come from three origins: the hydrodynamic forces and moments, and the gravity and buoyancy.

Hydrodynamic forces and moments are due to the action of the surrounding fluid on the vehicle body. In steady flight, they result from relative motion between the vehicle and the fluid mass, or from accelerated flow produced by the deflection of the control surfaces. The forces and moments are therefore functions of the relative velocity, acceleration, position and control surface deflection. Assuming that these forces and

moments are continuous functions of these different variables, each force and moment can be expanded in Taylor series about the equilibrium point. Because of the small perturbation assumptions, second and higher order terms can be neglected. Moreover, because of the symmetry assumptions that yielded an uncoupled model, the forces and moments influencing the longitudinal motion only depend on the longitudinal motion. This allows to simplify a lot the expressions for the hydrodynamic forces and moments. Then the gravity and buoyancy terms have to be considered. These are known in the local-level frame, it is just necessary to obtain their expression in the body-fixed frame using the appropriate transformation matrix, which here only depends on the vehicle pitch.

Now that on one side the forces and moments, and on the other side the accelerations and rate of change of momentum have been derived and simplified, it is possible to equate both expressions according to equation 5.1. The equilibrium forces and moments cancel one another, and only those due to the small perturbations remain.

This gives the complete simplified uncoupled linearized equations of motion for the small perturbations of longitudinal (also called symmetrical) motion. Finally, these equations can be rewritten in a single differential equation in a matrix form :

$$M \cdot \dot{X} + D \cdot X = C \cdot \delta_s \quad (\text{Eq. 5.2})$$

where  $M$  is the mass matrix,  $D$  is the drag matrix,  $C$  is the control matrix,  $\delta_s$  is the input sternplane angle, and  $X$  is the state (or unknown) vector. The expressions for  $M$ ,  $D$  and  $C$  are not detailed here.

The state vector  $X$  is:

$$X = \begin{bmatrix} u & \dot{u} & w & \dot{w} & q & \dot{q} & \theta \end{bmatrix}^T \quad (\text{Eq. 5.3})$$

where :

$u$  is the perturbation of the forward velocity in the body-fixed frame,

$w$  is the perturbation of the downward velocity in the body-fixed frame,

$q$  is the perturbation of the vehicle angular velocity about its y axis,

$\theta$  is the perturbation of the vehicle pitch.

The expressions of  $M$ ,  $D$  and  $C$  depend on the characteristics of the vehicle, particularly its geometry and a set of non-dimensional hydrodynamic coefficients [54]. To use these coefficients in the equation, it is necessary to dimension them, using both the geometry of the vehicle and the equilibrium conditions, particularly the mean forward velocity  $U_0$ .

### **V.2.1.1.3. Solving the Equations of Motion**

The above equation can be solved using different methods. In our case, we transformed this differential equation into a difference equation as follows. Using the definition of the time derivative:

$$\dot{X}(t) = \lim_{\delta t \rightarrow 0} \frac{[X(t + \delta t) - X(t)]}{\delta t} \quad (\text{Eq. 5.4})$$

and defining a timestep  $T$  sufficiently small, the above can be approximated as:

$$\dot{X}(nT) \approx \frac{[X((n+1)T) - X(nT)]}{T} \quad (\text{Eq. 5.5a})$$

or

$$X(nT) \approx X((n-1)T) + T \dot{X}((n-1)T) \quad (\text{Eq. 5.5b})$$

rewriting equation 5.2, assuming that the inverse of  $M$  exists, and using equation 5.5 yields to the following two difference equations:

$$\dot{X}((n-1)T) = -M^{-1} \cdot [D \cdot X((n-1)T) + C \cdot \delta_s((n-1)T)] \quad (\text{Eq. 5.6a})$$

$$X(nT) \approx X((n-1)T) + T \dot{X}((n-1)T) \quad (\text{Eq. 5.6b})$$

Then at each iteration, the previous derivative of the state vector is computed based on the previous state and input using equation 5.6a, then the actual state is computed based on the previous state and state derivative using equation 5.6b. We then have to define the initial conditions. Here, the initial state vector is set to 0.

This allows the solving of the time history of the state vector  $X$ , which includes the forward and downward velocities we need. But these are the values of the perturbation about the equilibrium, in the body-fixed frame. Two more steps are then required to obtain the actual velocities in the local-level frame:

$$\begin{aligned} U_{body}(nT) &= U0 + u(nT) \\ W_{body}(nT) &= w(nT) \end{aligned} \quad (\text{Eq. 5.7})$$

gives the total velocity in the body fixed frame, and

$$\begin{bmatrix} U \\ W \end{bmatrix}(nT) = C_b^l(nT) \cdot \begin{bmatrix} U_{body} \\ W_{body} \end{bmatrix}(nT) \quad (\text{Eq. 5.8})$$

gives the velocity in the local-level frame.  $C_b^l(nT)$  is the transformation matrix from body-fixed to local-level frame at time  $nT$ :

$$C_b^l(nT) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}_{\theta=\theta(nT)} \quad (\text{Eq. 5.9})$$

Then, using the same approximation as between equations 5.4 and 5.5, we write

$$\begin{bmatrix} x \\ z \end{bmatrix} (nT) \approx \begin{bmatrix} x \\ z \end{bmatrix} ((n-1)T) + T \begin{bmatrix} U \\ W \end{bmatrix} ((n-1)T) \quad (\text{Eq. 5.10})$$

and  $\begin{bmatrix} x \\ z \end{bmatrix} (0) = \begin{bmatrix} x_0 \\ z_0 \end{bmatrix}$  (Eq. 5.11)

to compute the trajectory  $(x,z)$  of the vehicle in the local frame.

Here,  $x_0$  is set to 0 since we decided to place the local level frame at the origin of the trajectory, but  $z_0$  is chosen to be 10m so that, setting the surface at  $z=0$ , we don't have to consider the diving phase which may be critical, since in such a case the vehicle is at the interface between air and seawater.

Providing the characteristics of the vehicle to simulate [54] and the initial conditions, we are then able to obtain the whole trajectory of the vehicle.

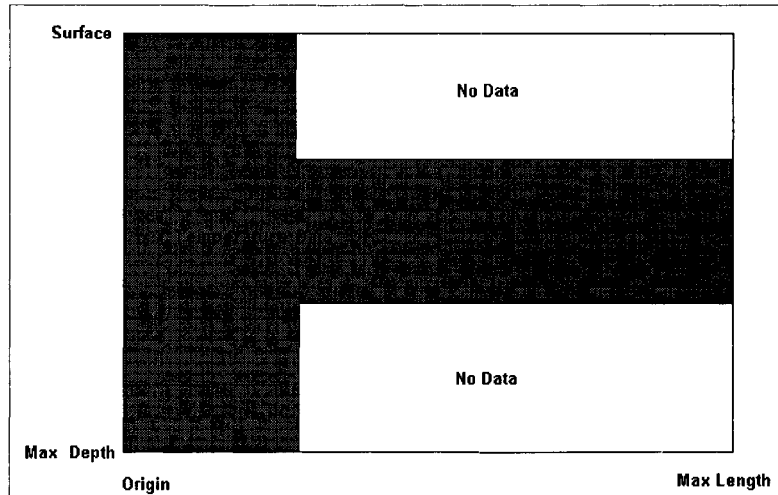
### **V.2.1.2. Water Slice Simulation**

The second main requirement for the simulation is a temperature map of a water slice. This map can either be totally synthetic, and generated as desired for a specific simulation need, or be based on the interpolation of real data acquired at sea, for instance during a Thermocline Survey Experiment. Synthetic data are useful for the early stage of the design of a controller, or its coarse tuning, while real data are required for finer tuning and validation. As explained before, the simulation discussed here only considers a 2-Dimensional problem. Therefore, a 2-D map of a water slice is necessary. The goal is to be able to obtain the temperature at a point identified by its horizontal and vertical coordinates. Two different things are then to be considered: the generation and storage of a map, and its use by a CTD simulator.

#### **V.2.1.2.1. Generation of a Temperature Map**

A 2-D map of temperature is basically a plane scalar field. The easiest way to store such a field is to use a 2-D matrix of floating point numbers. A simple lookup in the matrix, using the coordinates of a point as the row and column indices gives the temperature. But, using such storage method, no continuous map can be created. Rather than interpolating nearby measurements, it was decided to increase the resolution of the map, and round off the coordinates of a point to the nearby point for which a measurement is available. Particularly with real measurements, it is important to preserve the measurement noise in order to realistically simulate the water column. This requires to use a map resolution as high as possible, and no smoothing by interpolation or approximation. The drawback of this method is that using high resolution may create a huge map. A large map takes some time to be loaded, and uses a lot of memory, and the time spent switching memory pages reduces the processing speed.

When large maps are used, the memory requirements can be reduced by not storing unnecessary data: the simulated vehicle is supposed to dive from the surface to the maximum depth and then follow the thermocline around the middle of the water column. If the simulation is successful, the vehicle then doesn't need to measure the temperature in the upper and lower part of the water column except at the beginning when it is diving. The required map therefore has a shape as shown in Figure 71.



*Figure 71: Sparse Temperature Map*

A map with such a shape can be stored, considering that the temperature outside the relevant area is zero, in a sparse matrix. This can really reduce the memory requirements.

The map is then implemented as a file containing a structure made of:

- A 2-D matrix, possibly sparse, that represents the temperature map itself,
- Two floating point numbers representing the resolution in both x and z direction so that the CTD simulator knows how to read the map.

A particular case is defined when the resolution along the x direction is set to zero. This means that only a vertical profile is stored and that the map is homogeneous horizontally. This is useful to easily create small synthetic data sets for basic simulations in a water slice with no horizontal temperature variation.

#### **V.2.1.2.2. Use of the Map by the CTD Simulator**

For the CTD simulator to provide the temperature, it is then necessary to look up the temperature in the map based on the position of the AUV. Given the resolution  $dx$  and  $dz$  of the map, and the position  $x$  and  $z$  of the AUV, the temperature  $T$  at that location on the



map is then given by:

$$T = \text{Map}(\text{round}(x/dx), \text{round}(z/dz)) \quad (\text{Eq. 5.12})$$

Where  $\text{round}(x)$  is  $x$  rounded to the closest integer value.

The CTD Simulator also provides the measured depth  $z_{\text{measured}}$  which is first set to the real depth  $z$ :

$$z_{\text{measured}} = z \quad (\text{Eq. 5.13})$$

Then, to accurately emulate a CTD, it may be desirable to add some errors in the measurements. For the needs of the simulation, it is considered that errors modeled by static offsets and random noise are sufficient. However, because the thermocline tracking problem is really based on the measurements of relative variations of depth and temperature, possible offsets do not affect the behavior of the system. Therefore, no offset is considered. The random noise is simply chosen to have a normal distribution, with zero mean, and a standard deviation defined by the operator as the noise level for temperature ( $\text{noiseLevel}_T$ ) and depth ( $\text{noiseLevel}_d$ ) measurements. The noise on each sample of the temperature ( $\text{noise}_T$ ) and depth ( $\text{noise}_d$ ) are then computed as

$$\text{noise}_T = \text{randn}() * \text{noiseLevel}_T \quad (\text{Eq. 5.14})$$

$$\text{noise}_d = \text{randn}() * \text{noiseLevel}_d \quad (\text{Eq. 5.15})$$

where  $\text{randn}()$  returns a number drawn in a random sequence with normal distribution, zero mean and unity standard deviation.

Then the measurements computed in equations 5.12 and 5.13 are modified as:

$$T = T + \text{noise}_T \quad (\text{Eq. 5.16})$$

$$z_{\text{measured}} = z_{\text{measured}} + \text{noise}_d \quad (\text{Eq. 5.17})$$

Finally, taking into account the finite resolutions of the CTD, set to 1cm and 10<sup>-3</sup>°C, the final values returned for temperature and measured depth are computed as:

$$T = \text{round}(1000 \times T) / 1000 \quad (\text{Eq. 5.18})$$

$$z_{\text{measured}} = \text{round}(100 \times z_{\text{measured}}) / 100 \quad (\text{Eq. 5.19})$$

The final equations used to simulate the CTD are therefore:

$$T = \frac{\text{round}\{1000 \times [\text{Map}(\text{round}(x/dx), \text{round}(z/dz)) + \text{randn}() \times \text{noiseLevel}_T]\}}{1000} \quad (\text{Eq. 5.20a})$$

$$z_{\text{measured}} = \frac{\text{round}\{100 \times [z + \text{randn}() \times \text{noiseLevel}_d]\}}{100} \quad (\text{Eq. 5.20b})$$

### V.2.1.3. Other Main Requirements

The other important tasks are described hereafter.

#### V.2.1.3.1. Logger

For the simulation to be really useful, it is necessary to retain and output the time history of a number of variables. Given the number of variables, the duration and timestep of the simulation, storing this data in the memory would make the size of the used memory grow rapidly. Having the memory loaded that much reduces the processing speed a lot. Since there is no need for the simulator itself to retain the whole variable history, there is no need to have these variables saved in the directly accessible memory. Rather, it is a more efficient way to proceed to save the data in a file. To perform that task, a logger, that regularly writes the content of a certain number of variables to a file is required. The logger output is decided to be a text file, which although is larger than a binary file,

presents the advantage that it can be read by any application without specific processing. Particularly, the file can be opened with a text editor if we just want to have a quick look at the data.

### **V.2.1.3.2. Fins Controller**

The thermocline tracking controllers to be tested can either act as depth controllers or directly as fins controllers. In case it acts as a depth controller, the desired depth setpoint it outputs has to be translated into a sternplane position command. This task is handled by a fins controller simulator. For the needs of the simulation considered here, it is not really necessary to have a very fast, accurate and efficient controller. For that reason, a basic bang-bang controller is chosen. It is based on the following equation:

$$\delta_s = \delta_{sMax} \cdot \text{sign}(d_{desired} - d_{measured}) \quad (\text{Eq. 5.21})$$

where  $\delta_s$  is the sternplane angle,  $\delta_{sMax}$  is a predefined maximum value for that angle, which is here set to  $20^\circ$ ,  $d_{desired}$  is the desired depth and  $d_{measured}$  is the measured depth, which may differ from the actual depth. Indeed, this value comes from the CTD measurements, and thus has a smaller resolution than that kept for the motion simulation, and may moreover take into account a simulated measurement noise.

In order to limit the overshoot, the controller equation (5.21) was modified as:

$$\delta_s = \frac{\delta_{sMax}}{5} \cdot \text{sign}(d_{desired} - d_{measured}) \quad \text{if } 2m > |d_{desired} - d_{measured}| > 0.2m \quad (\text{Eq. 5.22a})$$

$$\delta_s = \frac{\delta_{sMax}}{50} \cdot \text{sign}(d_{desired} - d_{measured}) \quad \text{if } 0.2m > |d_{desired} - d_{measured}| \quad (\text{Eq. 5.22b})$$

$$\delta_s = \delta_{sMax} \cdot \text{sign}(d_{desired} - d_{measured}) \quad \text{otherwise} \quad (\text{Eq. 5.22c})$$

Although such a controller is very simple and obviously not the most efficient, it is sufficient for the needs of the simulation, and is indeed able to control the vehicle to the desired depth in a relatively small amount of time, without significant overshoot or steady-state error. Later, if a more efficient controller is desired for the simulation, the fins controller can be modified so as to emulate that used in the OEX.

#### **V.2.1.3.3. Scheduler**

Because the simulation considers a discrete time, the execution of each task has to be regularly scheduled. Moreover, the timestep required for each task is not necessarily the same. For instance, the AUV motion simulation requires a relatively small timestep, so that the differential equation 5.4 can be correctly approximated by the difference equation 5.5, whereas it is not necessarily interesting to log the data at the same high rate. For that reason, each task has to be performed regularly at its own pace. Therefore, it is necessary to correctly schedule the execution of all tasks. Moreover, most of the tasks do not really consider the existence of an absolute time, but rather some iterations, separated by timesteps. Thus, the scheduler also has to maintain an equivalence between iteration number and simulated time. The role of that scheduler is to first initiate the simulation, then count the simulated time while supervising the execution of each task at the required pace, and finally terminate the simulation. Throughout the execution, it also monitors the indication of errors. In case an error occurs, it properly aborts and terminates the simulation.

## **V.2.2. Implementation**

Based on the theory and general considerations discussed above, the simulation tool is implemented as a set of Matlab functions and scripts as described hereafter.

### **V.2.2.1. Required Modules**

Table 13 lists the modules implemented in the simulation tool, along with a summary of the purpose, input and output of each module.

<i>Module</i>	<i>Task</i>	<i>Needs</i>	<i>Produces</i>
Simulation initialization	Initializes the simulation	All user-defined parameters	Indication of errors
Simulation termination	Terminates the simulation	Identification of each open file	Indication of errors
Scheduler	Schedules the execution of other modules, and stores the common variables	Scheduling parameters	Indication of errors
Motion simulator	Simulates the motion of the AUV	Previous motion history, and position of the sternplane	Motion of the AUV at the next instant
CTD simulator	Simulates the AUV CTD	The position of the AUV and a temperature map	Temperature and depth readings
Logger	Logs selected variables in a file	The variables to log, and an output file	Output file
Safety simulator	Simulates the safety procedures of the vehicle	Variables to monitor, particularly the depth	Indication of errors
Fins controller	Controls the sternplane, based on a depth command	Desired and actual depth	Sternplane position
Thermocline tracking controller <sup>(1)</sup>	Controls the AUV to track the thermocline	AUV depth, CTD temperature and other variables to define	Either a desired depth or a desired sternplane angle
Summary generation	Creates a summary of the simulation	Parameters, conclusions, and an output file	Summary file
Display results	Displays the results of the simulation	Logger and CTD map files	Summary plot

*Table 13: Simulation Modules Description*

*(1): Again, the thermocline tracking controller is not really considered a part of the simulation tool itself. What is rather considered here is an interface for that module.*

### V.2.2.2. Module Interactions

Figure 72 shows the different modules and their interactions.

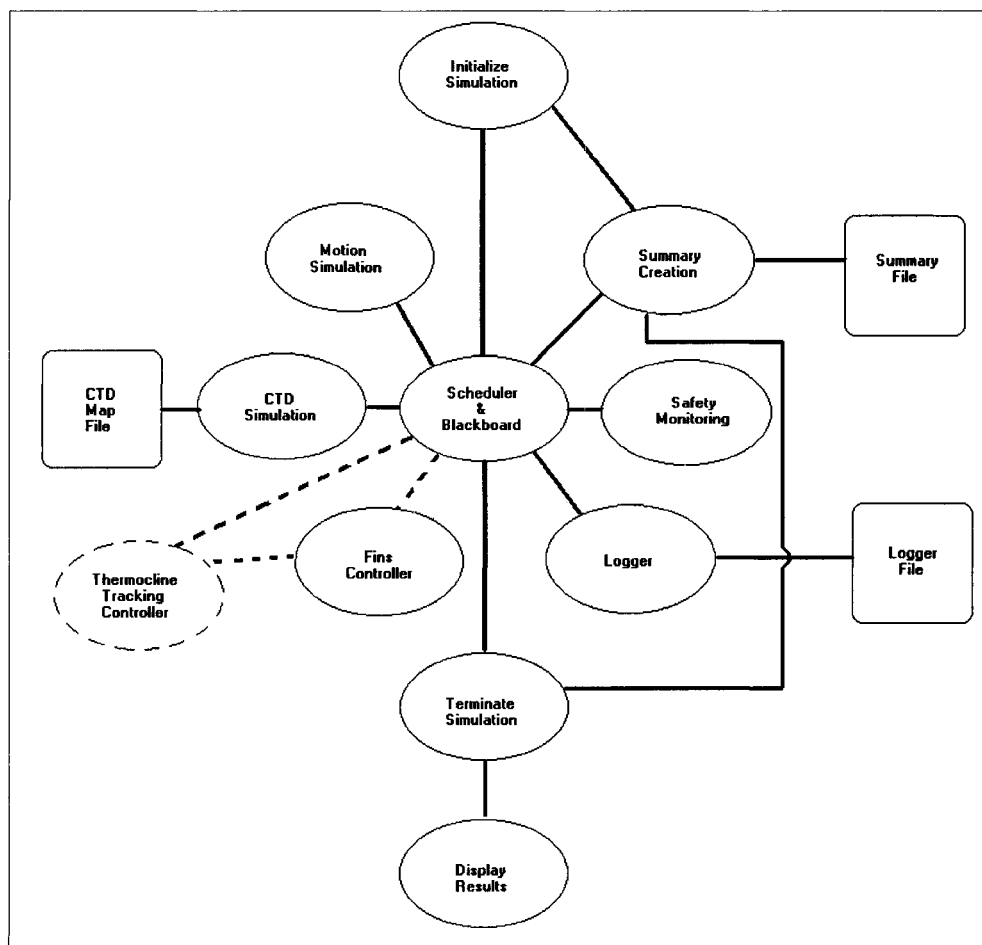


Figure 72: Simulation Modules Interactions

In Figure 72, the dashed links indicate an alternative choice: the fins controller can be optionally used, depending on the type of thermocline tracking controller implemented.

### V.2.2.3. Simulation Interface

The interface of the simulator, from the operator point of view, includes the simulation parameters and the simulation output.

### **V.2.2.3.1. Simulation Parameters**

The parameter of the simulation that are to be defined by the operator are listed in

Table 14.

<i>Parameter</i>	<i>Name</i>	<i>Comments</i>
Scheduler timestep	<i>Scheduler_dt</i>	Must be smaller than any other timestep
Motion simulation timestep	<i>Motion_dt</i>	
CTD simulation timestep	<i>CTD_dt</i>	
Fins controller timestep	<i>Fins_dt</i>	
Thermocline tracking controller timestep	<i>Tracking_dt</i>	
Logger timestep	<i>Logger_dt</i>	
Safety monitor timestep	<i>Safety_dt</i>	
Simulation duration	<i>Sim_duration</i>	
CTD map file	<i>CTD_File</i>	Includes the map and indications of its resolution
Noise level on temperature measurements	<i>NoiseLevelT</i>	Represents the standard variation of a normal distribution
Noise level on depth measurements	<i>NoiseLevelD</i>	Represents the standard variation of a normal distribution
Maximum depth for thermocline tracking	<i>Max_d_track</i>	
Minimum depth for thermocline tracking	<i>Min_d_track</i>	
Maximum safety depth	<i>Max_d</i>	Should be deeper than Max_d_track
AUV initial forward velocity	<i>U0</i>	
Logger file name	<i>Logger_file</i>	
Summary file name	<i>Summary_file</i>	

Table 14: Simulation Parameters



### **V.2.2.3.2. Simulation Output**

The simulation output consists of four parts:

- Messages displayed on the console that provide critical information and summarize the progression of the simulation once in a while,
- A window showing a plot of the vehicle trajectory over the temperature map,
- A summary file that lists a reminder of the parameters and a brief summary of the simulation. See Appendix for an example of summary file as generated by the simulation tool.
- A logger file that contains the time history of as many variables as desired.

### **V.2.2.4. Thermocline Tracking Interface**

Here we are considering the interface between the thermocline tracking controller and the whole simulation tool. The thermocline tracking controller is a separate module that is interfaced with the simulation tool. It can get access to any desired variable stored in the blackboard. In turn, the thermocline tracking controller outputs either a desired depth or a sternplanes setpoint, depending on the type of controller considered. Moreover, it is possible for the controller to output additional variables to be recorded by the logger.

### **V.2.2.5. Implementation of Each Module**

In this part, the implementation of each module is summarized.

#### **V.2.2.5.1. Simulation Initialization**

The simulation initialization module checks all the parameters, creates the required output files, and loads the CTD map. In case of error, the simulation is aborted. Once

every parameter has been checked, a header is written in the summary file, listing all the parameters. Then, the execution returns to the scheduler to begin the computation. This module is implemented as the Matlab function summarized in Table 15.

<i>Function: SimInit</i>	<i>Variables</i>	<i>Comments</i>
<i>Input Variables</i>		The inputs of this function consist of all the user-defined input parameters of the simulation. These parameters are listed in Table 14
<i>Output Variables</i>	<i>Error</i>	Simulation error
	<i>f_Summary</i>	Handler to the summary file
	<i>f_logger</i>	Handler to the logger file
	<i>CTD_Map</i>	CTD map and resolution

Table 15: Simulation Initialization Implementation Summary

#### **V.2.2.5.2. Simulation Termination**

This module writes a message indicating the end of the simulation in the summary file, closes all open files, and frees up some memory. It is implemented as the Matlab function summarized in Table 16.

<i>Function: SimFinish</i>	<i>Variables</i>	<i>Comments</i>
<i>Input Variables</i>	<i>Error</i>	Simulation error
	<i>f_Summary</i>	Handler to the summary file
	<i>f_logger</i>	Handler to the logger file
<i>Output Variable</i>	<i>Error</i>	Simulation error

Table 16: Simulation Termination Implementation Summary

#### **V.2.2.5.3. Scheduler**

This module is the core of the simulation tool. It initializes the simulation, then schedules the execution of each other module, and finally terminates the simulation. It moreover

creates the correspondence between iteration index and simulated time. Finally, throughout the simulation, the scheduler monitors the indication of errors reported by the other modules. In case of errors, it properly aborts and terminates the simulation.

The scheduler is implemented as the Matlab function described in Table 17.

<b><i>Function: SimSchedule</i></b>	<b><i>Variables</i></b>	<b><i>Comments</i></b>
<b><i>Input Variables</i></b>		The inputs of this function consist of all the user-defined input parameters of the simulation. These parameters are listed in Table 14
<b><i>Output Variable</i></b>	<b><i>Error</i></b>	Simulation error

*Table 17: Scheduler Implementation Summary*

#### **V.2.2.5.4. Vehicle Motion Simulation**

This module generates the instantaneous position, velocity and attitude of the vehicle, depending on its previous motion history, its dynamics, and the command applied to its control surfaces. It is based on the implementation of the equations 5.6 to 5.11 discussed in V.2.1.1.2. This module is implemented as the Matlab function summarized in Table 18.

<i>Function: SimVehicleMotion</i>	<i>Variables</i>	<i>Comments</i>
<i>Input Variables</i>	<b>ui_prev</b>	Previous alongtrack velocity
	<b>wi_prev</b>	Previous vertical velocity
	<b>ub_prev</b>	Previous forward velocity
	<b>ubd_prev</b>	Previous forward acceleration
	<b>wb_prev</b>	Previous downward velocity
	<b>wbd_prev</b>	Previous downward acceleration
	<b>q_prev</b>	Previous pitch rate
	<b>qd_prev</b>	Previous pitch rate derivative
	<b>pitch_prev</b>	Previous pitch
	<b>x_prev</b>	Previous alongtrack position
	<b>z_prev</b>	Previous depth
	<b>SternAngle</b>	Sternplanes position
	<b>U0</b>	Equilibrium forward velocity
	<b>Motion_dt</b>	Motion simulation timestep
<i>Output Variables</i>	<b>x</b>	New alongtrack position
	<b>z</b>	New depth
	<b>ui</b>	New alongtrack velocity
	<b>wi</b>	New vertical velocity
	<b>ub</b>	New forward velocity
	<b>wb</b>	New downward velocity
	<b>ubd</b>	New forward acceleration
	<b>wbd</b>	New downward Acceleration
	<b>q</b>	New pitch rate
	<b>qd</b>	New pitch rate derivative
	<b>pitch</b>	New pitch
	<b>error</b>	Simulation error

*Table 18: Motion Simulation Implementation Summary*

Most of the variables used for each iteration, such as the velocities in the body-fixed frame, are implemented as inputs and outputs so that they are visible from outside the function in order to be recorded by the logger. This was used for testing of the motion

simulation. It also allows to monitor these variables during the development of a thermocline tracking controller. For instance, it is interesting to monitor the vehicle pitch, since it has been observed that the accuracy of the CTD, which is the main sensor on which the tracking algorithm relies, can be degraded by turbulences as soon as the vehicle pitches significantly. A simple modification would make possible to statically retain most of these variables within the motion simulation function so as to reduce the number of parameters exchanged between modules if it is considered that these variables do not need to be logged anymore.

#### **V.2.2.5.5. CTD Simulation**

The purpose of this module is to simulate the CTD, providing the simulated vehicle with measurements of the depth and temperature, according to equations 5.20a and 5.20b. A CTD capable of averaging several samples is implemented. If the CTD samples at a frequency  $f_{CTD}$ , and averages  $N_{avg}$  samples, its output varies at a frequency  $f_{Output}$ .

$$f_{Output} = f_{CTD} / N_{avg} \quad (\text{Eq. 5.23})$$

To implement this averaging, the module is called every  $CTD\_dt$ . At each execution, it computes a new sample, and averages it with the previous ones. After  $N_{avg}$  calls, the output takes the value of the average, and the averaged data is reseted.

It is also interesting to compute the depth rate and the vertical temperature gradient since these values are likely to be necessary for a thermocline controller. It is reasonable to implement these computations in the CTD simulator, provided that the values are computed based on the simulated CTD measurements, possibly noisy. The measured depth rate  $z_{rate}$  and temperature gradient  $T_{gradient}$  are computed using:

$$z_{rate} = \frac{z_{measured} - z_{measured, previous}}{dt \times N_{avg}} \quad (\text{Eq. 5.24})$$

$$T_{gradient} = \frac{T - T_{previous}}{z_{measured} - z_{measured, previous}} \quad (\text{Eq. 5.25})$$

The algorithm described by equations 5.20, 5.24 and 5.25 is implemented in the Matlab function summarized in Table 19.

<b>Function: SimCTD</b>	<b>Variables</b>	<b>Comments</b>
<b>Input Variables</b>	<b>x</b>	Horizontal position of the AUV
	<b>z_real</b>	Real vertical position of the AUV
	<b>CTD_dt</b>	CTD simulation timestep
	<b>CTD_Map</b>	Temperature map and map resolution
	<b>CTD_Navg</b>	Number of CTD samples to average
	<b>NoiseLevelD</b>	Noise level on depth measurements
	<b>NoiseLevelT</b>	Noise level on temperature measurements
<b>Output Variables</b>	<b>z_measured</b>	Measured depth, rounded and possibly noisy
	<b>Temperature</b>	Temperature, rounded and possibly noisy
	<b>z_rate</b>	Depth rate as measured by the CTD
	<b>Temp_grad</b>	Temperature gradient as measured by the CTD

Table 19: CTD Simulation Implementation Summary

#### **V.2.2.5.6. Fins Controller Simulation**

The purpose of this module is to determine the position of the sternplane based on a desired depth setpoint and the actual measured depth, using equation 5.22, as discussed in V.2.1.3.2. The fins controller is implemented in the Matlab function summarized in Table 20.

<i>Function: SimFins</i>	<i>Variables</i>	<i>Comments</i>
<i>Input Variables</i>	<b>z_desired</b>	Desired depth to reach
	<b>z_measured</b>	Actual depth, as measured by the CTD
<i>Output Variable</i>	<b>SternAngle</b>	Sternplanes angle

Table 20: Fins Controller Implementation Summary

### V.2.2.5.7. Logger Simulation

This module stores the history of selected variables in a file. The logger is a function called at a regular, user-defined, simulation timestep. All the data to be logged is passed to the function. The data is first written in a temporary string which is then written to the file, so that, for each iteration of the logger function, there is only one file access, instead of one for each variable, which may take more time. For the same reason, to increase the speed of that process, the logger file is not opened and closed each time, which would require the disk to be accessed very often. Rather, the file is written and let open. Doing so, we let the Operating System (OS) manage the disk buffer, instead of forcing a buffer flush each time we close the file. Therefore, a disk access is only necessary once the file buffer has reached the size defined by the OS. This algorithm is implemented in the Matlab function described in Table 21.

<i>Function: Logger.m</i>	<i>Variables</i>	<i>Comments</i>
<i>Input Variables</i>	<b>time</b>	Simulation time for timestamp
		Any variable to log
<i>Output Variable</i>	<b>error</b>	Indication of file error

Table 21: Logger Implementation Summary

Every time the logger is called, the following variables are logged with a timestamp:

- AUV position, velocity, attitude,

- Desired depth,
- Temperature and depth measured by the CTD,
- Sternplane angle.

Other variables can be logged with little rewriting: one just has to send the variable to the logger function, and specify how the variable should be written in the output file.

#### **V.2.2.5.8. Summary Generator**

The purpose of that module is to write a small summary file that is a reminder of the simulation that has been run. The summary consists of:

- The date and time at which the simulation began,
- A reminder of all the user-defined parameters used for that simulation,
- A timestamped summary of the tasks performed during the simulation,
- The conclusion of the simulation: success or abortion due to an error. In case of an error, a description of the error is given whenever possible.

The summary is written in a text file so that it is easily open with any text editor, and the operator can add a few comments at the end of a file as a reminder of his conclusions on the simulation. As for the logger, the summary file is kept open during the simulation, so as not to force a buffer flush and not to have to reopen and close the file every time something is written. The goal is to minimize the time lost in hard drive access.

The implementations is done in three parts:

- At the beginning of the simulation, the function SimInit described in V.2.2.5.1 writes the current date and time, and lists the value of all user-defined parameters used for the simulation, and writes a few messages during the simulation initialization phase.



- Once these tasks have been performed, the summary file remains open, and a function summarized in Table 22 can be used to write a few timestamped messages. This function receives the message to be written as parameter, gets the current time, and writes everything in a temporary text string, appended to the summary file.
- Finally, once the simulation is terminated, a last message is written and the file is closed by the function SimFinish described in V.2.2.5.2.

<i>Function: WriteSummary</i>	<i>Variables</i>	<i>Comments</i>
<i>Input Variables</i>	<b>time</b>	For timestamp
	<b>error</b>	Does that message indicate an error ?
	<b>msg</b>	Message to write in the summary file
<i>Output Variable</i>		None

Table 22: Message Writing Implementation Summary

### **V.2.3. Graphical User Interface**

To simplify the use of the simulation tool, a user friendly front-end layer in the form of a Graphical User Interface (GUI) was developed, as summarized hereafter.

#### **V.2.3.1. Overview**

The purpose of this GUI is to provide the operator with a user-friendly way to configure and run the simulation, and look at some results. It should moreover implement a few systematic tests that check the parameters entered by the operator, to avoid basic mistakes. Finally, it should be designed in such a way that the operator is not required to have much knowledge about the implementation of the simulation tool itself. Doing so, the operator only needs to implement the thermocline tracking controller and configure the simulation tool.

### **V.2.3.2. Implementation**

The GUI is implemented as a main Matlab graphical window containing a number of text labels, text edition fields, buttons, and option checkboxes. Two other windows are temporarily used to display particular graphical data when necessary, as discussed after. The implementation of each part of the GUI is discussed in the following sections.

#### **V.2.3.2.1. Main Window**

Launching the simulation creates the interface as a main graphical window, which waits for the operator to type in some text, click a checkbox or depress a button. When the window is opened, some of its properties, such as “resizable” and “numbering” are modified, the title is set to “Thermocline Tracking Simulation”, and the window is brought to the front. Most of the parameters fields are filled in with some default values. A few other operations, not detailed here, are performed.

#### **V.2.3.2.2. Timing Parameters Edition**

A part of the GUI is dedicated to the input of the timing parameters for the simulation. These parameters consist of the timestep for each module – or task – as well as the simulation duration. As explained above, most of these parameters are filled in with some default values that the operator can change. A particular case is implemented for the field “Fins dt” that corresponds to the timestep for the execution of the fins controller module. The use of this module is optional, since it is required only for generating the fins position in case the thermocline tracking controller only outputs a desired depth command. For that reason, the “Fins dt” field is originally not active (not modifiable).

Checking a box labeled “Need fins controller” not only sets the corresponding variable to TRUE, but also activates the “Fins dt” parameter field for the operator to set it.

For any of the timing parameters, anytime one of the fields is changed, the corresponding callback function checks that the value entered in the field is a numerical value, as expected. Figure 73 presents a detailed view of that section of the GUI.

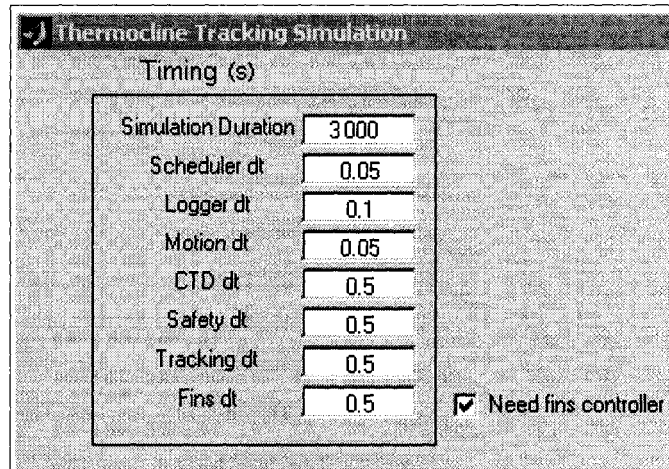


Figure 73: Timing Parameters Section of the GUI

### **V.2.3.2.3. Input and Output Files Specification**

A second part of the GUI is dedicated to the specification of the I/O files, namely the summary, logger and the CTD map files. For each file, a text field allows the operator to directly type in the path and name of the file. A more convenient “Browse” button is added to avoid typing and prevent mistakes.

For each of the output files (summary and logger), an “Overwrite” checkbox is added to prevent undesired overwriting of previous results. If that box is not checked, the overwriting is not allowed, and the simulation will not run in case the file already exists. For the input file (CTD map), a “Preview” button is made active once a file is specified in the filename field. Pressing that button loads the specified file, and creates a second

window entitled “CTD Map Preview” that displays the temperature map. Meanwhile, the “Preview” button is changed to “Close Preview”, and the browse button is made inactive. When either the close button of the figure or the “Close Preview” button of the GUI is depressed, the figure is closed, the GUI brought to the front, the “Close Preview” button changed back to “Preview”, and the “Browse” button made active again. Figure 74 presents a detailed view of that portion of the GUI, in one of its two possible states.

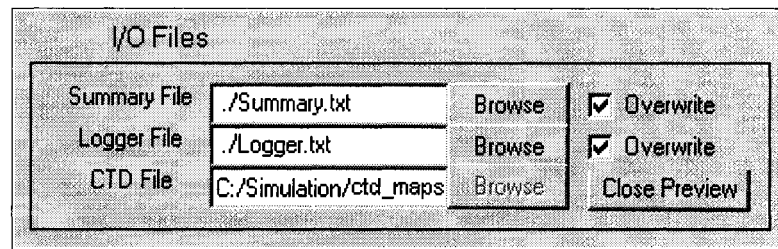


Figure 74: I/O Parameters Section of the GUI

#### **V.2.3.2.4. Tracking Controller Specification**

A third part of the GUI is dedicated to specifying the name of the file containing the core of the thermocline tracking controller. In case the controller uses several files, only the name of the topmost function has to be specified, provided that the other files are in the same directory. This section is very similar to that related to the I/O files described above. It includes both an edition field and a “Browse” button. Figure 75 presents a detailed view of that portion of the GUI.

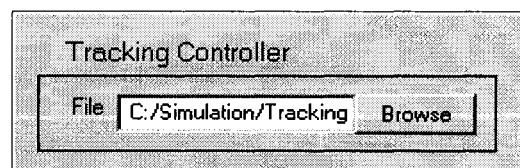


Figure 75: Tracking Controller Section of the GUI

### **V.2.3.2.5. CTD Simulation Configuration**

A fourth part of the GUI is dedicated to the configuration of the CTD simulator. Three checkboxes are implemented: “Depth Noise”, “Temperature Noise” and “Averaging”. Checking one of the boxes enables the corresponding field where the value for the noise level or the number of samples to average is specified. Figure 76 presents a detailed view of that portion of the GUI.

CTD		
<input checked="" type="radio"/> Depth Noise	0.02	m
<input checked="" type="radio"/> Temperature Noise	0.002	degC
<input type="radio"/> Averaging	2	Samples

*Figure 76: CTD Configuration Section of the GUI*

### **V.2.3.2.6. Mission Parameters**

A last parameter section, similar to that where the timing parameters are entered, is dedicated to the specification of other parameters related to the AUV mission simulation. These parameters are: the mean (equilibrium) vehicle forward velocity, the maximum and minimum depth for the tracking, and the maximum safety depth for the AUV. This section consists of four text fields, with default values that the operator can modify. Figure 77 presents a detailed view of that portion of the GUI.

AUV		
Initial Forward Velocity	1.5	m/s
Minimum Tracking Depth	10	m
Maximum Tracking Depth	80	m
Maximum Safety Depth	100	m

*Figure 77: Mission Configuration Section of the GUI*

#### **V.2.3.2.7. Running the Simulation**

Whenever the “Go” button is depressed, the corresponding callback function retrieves all the required parameters from the other fields of the GUI, performs a few verifications on these parameters, and calls the main simulation module (scheduler). When the button is depressed, it freezes to indicate that the computation is running, until the computation is completed. At that point, the “Go” button is set back to its normal active state, and a message box is created in the forefront to indicate the end of the simulation. Depending on the conclusion of the simulation, this message box is either an error or success message box.

#### **V.2.3.2.8. Displaying Results**

When simulation results are available, a “View Results” button is activated. When this button is depressed, a new window, entitled “Simulation Results” is opened. The AUV trajectory, as recorded in the logger file, is plotted over the temperature map retrieved from the CTD map file. As long as simulation results are not available, the “View Results” button is inactive.

#### **V.2.3.2.9. Terminating the Simulation**

When the “Close Simulation” button is depressed, the corresponding callback function terminates all activities, closes the secondary windows (CTD Map preview and results display) if any, frees all the allocated resources, closes the simulation tool and the GUI itself, and finally returns to Matlab.

### V.2.3.3. Graphical Layout

An general overview of the whole GUI layout is presented in figure 78.

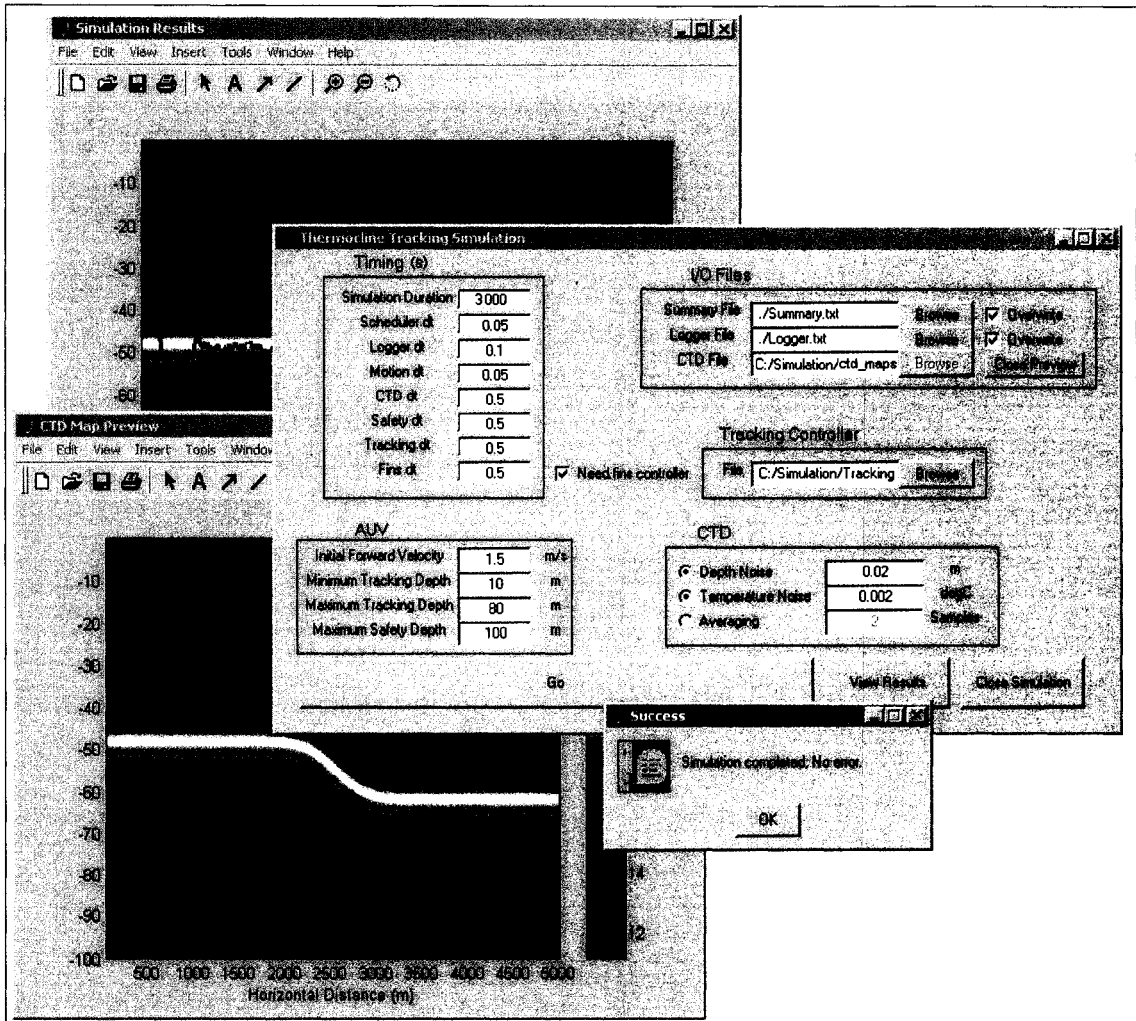


Figure 78: Graphical Layout of the whole GUI

### V.2.4. Test of the Simulation Platform

Once the whole simulation tool was implemented, a general test was performed to ensure that everything was working satisfactorily so that the problem of designing a tracking controller could be considered. This section describes the test performed and the results.

### **V.2.4.1. Description of the Test Simulation**

The goal of this simulation was to test all the functionalities of the simulation tool. Because only the simulator was considered here, and not the thermocline tracking controller, it was necessary to design a module that can command the AUV, in place of the tracking controller. A simple module successively generating various depth setpoints was written. The fins controller was then used to translate these depth commands into sternplane angle setpoints. In order to thoroughly test the CTD simulator, all its functionalities were used, particularly the averaging and noise generation. Because the simple depth command generator we used did not consider the temperature, a very simple map of the water column was used, just to ensure that it was correctly read. Finally, the last depth command was set beyond the maximum defined safety depth, in order to ensure that the safety module correctly aborts the simulation.

### **V.2.4.2. Simulation Configuration**

The simulation parameters used for this test are listed in Table 23.



<i>Parameter</i>	<i>Name</i>	<i>Value</i>
Scheduler timestep	<i>Scheduler_dt</i>	20ms
Motion simulation timestep	<i>Motion_dt</i>	20ms
CTD simulation timestep	<i>CTD_dt</i>	0.1s (5 samples averaging, output frequency 2Hz)
Fins controller timestep	<i>Fins_dt</i>	0.2s
Thermocline tracking controller timestep	<i>Tracking_dt</i>	0.2s
Logger timestep	<i>Logger_dt</i>	0.1s
Safety monitor timestep	<i>Safety_dt</i>	0.5s
Simulation duration	<i>Sim_duration</i>	700s
CTD map file	<i>CTD_File</i>	../ctd/simple_test.mat
Noise level on temperature measurements	<i>NoiseLevelT</i>	$2.10^{-3}$ °C
Noise level on depth measurements	<i>NoiseLevelD</i>	2cm
Maximum depth for thermocline tracking	<i>Max_d_track</i>	40m
Minimum depth for thermocline tracking	<i>Min_d_track</i>	10m
Maximum safety depth	<i>Max_d</i>	60m
AUV initial forward velocity	<i>U0</i>	1.5m/s
Logger file path and filename	<i>Logger_file</i>	../Logger.txt
Summary file path and filename	<i>Summary_file</i>	../Summary.txt

*Table 23: Input Parameters for Test Simulation*

The CTD map used was a simple profile, with only a vertical variation, which is shown in Figure 79. The horizontal map resolution was therefore set to 0. The vertical one was 15cm. The depth command was the pattern shown in Figure 79.

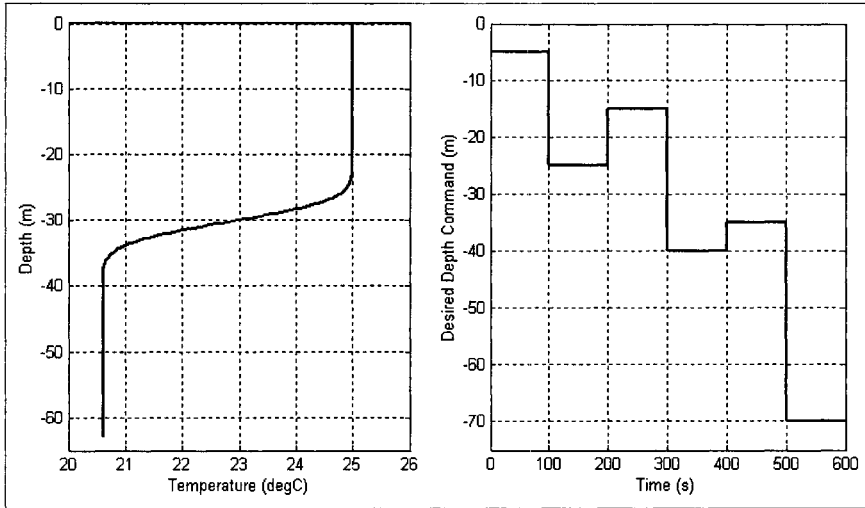


Figure 79: Test Simulation Inputs: Temperature Profile and Depth Command Pattern

### V.2.4.3. Results

The simulation was run, and as expected, aborted near the end because the safety simulation module fired a “Too Deep” error. A look at the generated summary showed, after a reminder of the parameters used, that after 89s of simulation (simulated time 589s), the safety module reported a “Too Deep” error, and that in turn, the scheduler also reported the error and aborted the simulation. Then, the data recorded in the logger output file was post-processed. First the path of the vehicle is presented in Figure 80.

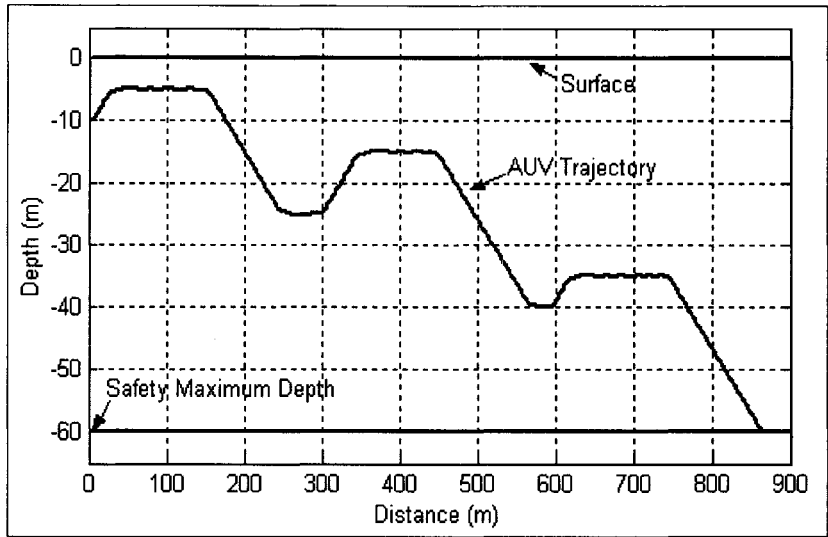


Figure 80: Trajectory of the Vehicle

Then, it is interesting to compare the actual, measured and desired depth, in Figure 81.

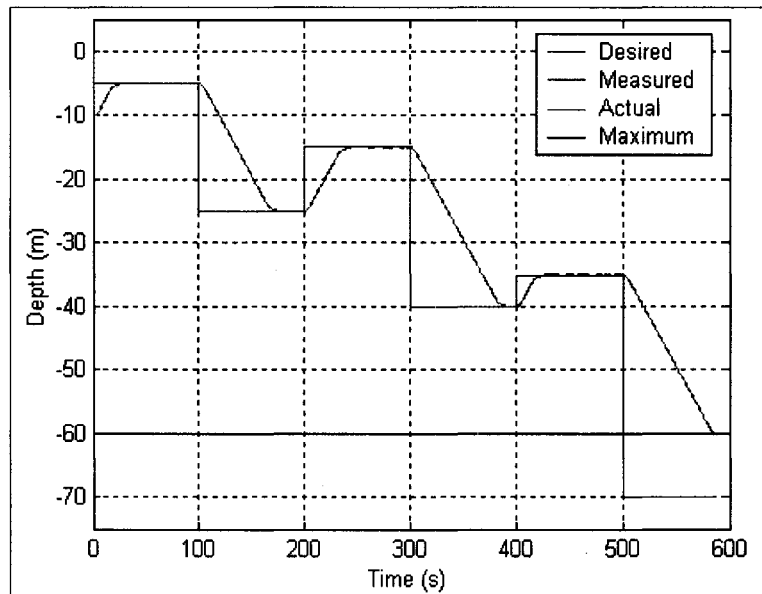


Figure 81: Depth Comparison (Desired, Measured, Actual and Maximum)

Figure 82, showing the difference between real and measured depth, emphasizes the simulated noise on the CTD depth measurements.

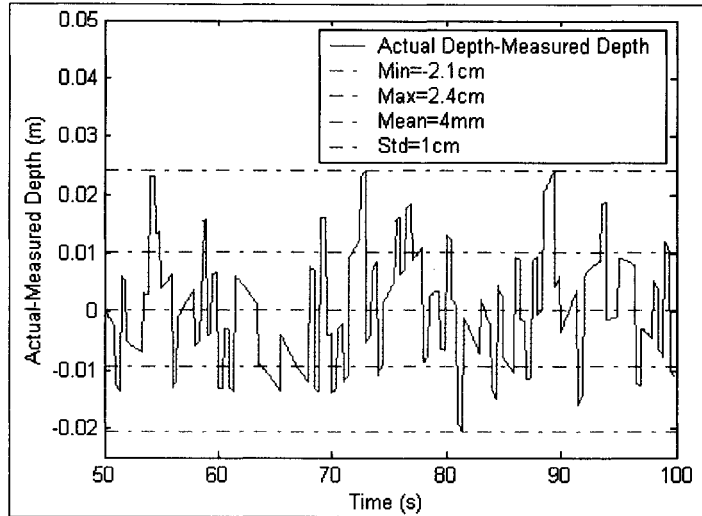


Figure 82: Simulated Noise on Depth Measurements

Figure 83, comparing the temperature profile sensed by the vehicle to the temperature profile used as an input, shows the simulated noise on the temperature measurements.

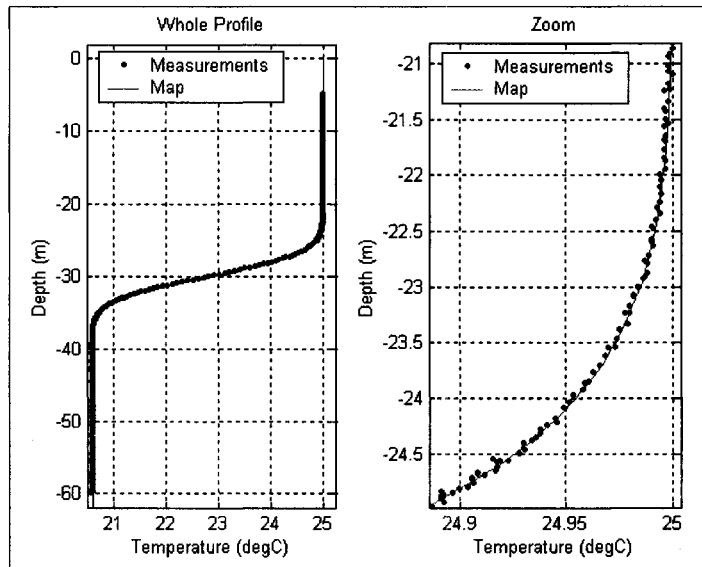


Figure 83: Simulated Noise on Temperature Measurements

Finally, Figure 84 compares the sternplane angle and the depth error.

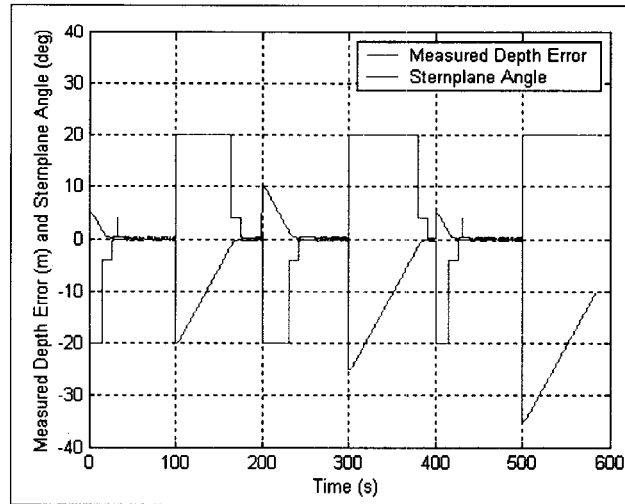


Figure 84: Measured Depth Error and Sternplane Angle

#### V.2.4.4. Conclusion

The test summarized above, as well as others not reported here, show that overall, the simulation tool is working, and is now ready to be used to test some thermocline tracking controllers.

### V.3. Thermocline Tracking Controller Simulation

Now that a complete simulation platform is available, the design of a thermocline tracking controller is considered.

#### V.3.1. Method

The goal is to design a module that controls the vertical motion of an AUV based on temperature measurements. Because of the definition of a thermocline, which is a layer of more intensive vertical temperature gradient than that found in the layers above and below it, the control algorithm cannot be simply based on the temperature. Rather, it

should be based on a local comparison of the temperature variation found at different depths. To determine the vertical temperature gradient, two different methods can be considered. The AUV can either travel horizontally, while two sensors, one above and the other below the vehicle, measure the temperature, acting as a differential sensor. In such a case, the goal is to control the AUV in order to maximize the measured difference. The other method relies on the use of a single temperature sensor. It consists of having the AUV change its depth to measure the temperature at different depths. From the measurements at different depths, the temperature gradient can be computed, and the AUV motion is then controlled to maximize this gradient.

The advantage of the first method is that the control algorithm is probably simpler. The drawback is that it requires an accurate differential sensor. This requires the use of two similar sensors, calibrated one against the other to eliminate differential mode errors (common mode errors are not critical). Moreover, because the order of magnitude of the vertical temperature gradient is around  $0.1\text{ }^{\circ}\text{C}/\text{m}$ , the sensors have to be mounted at least one meter apart, which may not be convenient. This approach is not considered further here. The second method is simpler in terms of sensing. Only one temperature and depth sensor is required, and the calibration is less critical. In turn, the control algorithm is more complicated. Because it is not feasible to have an AUV measure vertical temperature profiles, the AUV is required to change depth while traveling. The required trajectory is therefore in the shape of an oscillating pattern. Assuming that the depth variation is on the same order of magnitude as the horizontal displacement, and moreover considering that the horizontal variation of temperature is much less than the vertical one (Figure 79), the AUV can measure a local vertical gradient along an oscillation.

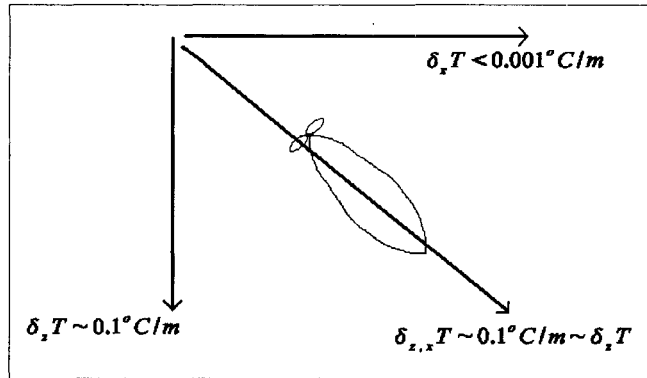


Figure 85: Local Temperature Gradient Measurement

Therefore, the problem is to control the vehicle to perform an oscillating pattern and measure the vertical temperature gradients, while the mean depth of the oscillation remains around the depth of maximal gradient. This constant oscillation requires the AUV to be somehow unstable. If the vehicle depth converges towards a steady value, we cannot ensure any longer that the thermocline is tracked, because no comparison with the temperature above or below is performed.

Two methods can be used to have the AUV oscillate around the depth of the thermocline. The trajectory can be partially modeled: the depth or sternplane command can be generated according to a mathematical model describing an oscillating pattern designed to suit the dynamics of the AUV. The mean depth of the oscillations is then computed based on the analysis of the acquired data. The other method is to bypass the phase of reasoning and modeling, using a more reactive controller. As discussed in II.2, modeling and reasoning are usually complex, time-consuming and of limited robustness. It is believed that more basic reflex-like behaviors are likely to provide better results, although they are more difficult to globally analyze, and it is more difficult to assess their theoretical performance and robustness.

Finally, two more distinctions can be made between control methods. The thermocline tracking controller can either control only the depth of the vehicle, generating a desired depth command, or control the sternplane of the vehicle, directly generating the fin position command. More complex algorithms can also involve pitch or vertical velocity control.

The work summarized here considers two different cases of control of an oscillating AUV based on measurements from a single CTD sensor: first, a depth controller based on the modeling of an oscillating pattern, then a reactive sternplane controller based on the implementations of simple reflexes are considered.

### **V.3.2. Thermocline Tracking Depth Controller**

This section describes the design, implementation and testing of a thermocline tracking depth controller that makes the AUV oscillate around the mean depth of the thermocline.

#### **V.3.2.1. Approach**

The idea used here is to have the vehicle perform some oscillations, the mean depth of which is updated based on temperature and depth measurements so as to remain around the thermocline to track. Several approaches are possible to ensure that the AUV is oscillating, one of them being to generate an oscillating command without taking into account the trajectory really followed by the vehicle. The advantage of this method is that it is much simpler to implement. The drawback is that the efficiency is not certain.

Two ways are possible to generate an oscillating trajectory, considering a depth command that is an oscillating function of either horizontal position or time. This doesn't really



make much difference if we consider a constant mean forward velocity. Here, we chose to generate a time-dependent oscillation along a sinusoidal pattern. The depth command is then computed adding this oscillation to a mean depth obtained from processing of the depth and temperature measurements that computes the depth of the maximum vertical temperature gradient.

### **V.3.2.2. Design**

Since the tracking mission can be divided into three phases as described in V.1.2, each phase is considered separately for the design of the controller. The generation of the depth command during the first phase is the simplest one. The desired depth is set to the maximum tracking depth. Meanwhile, the target depth for the second phase has to be computed. To do so, each new temperature and depth measurement obtained from the CTD is stored in a stack. Then the content of the whole stack is filtered using a mobile average filter, to reduce the measurement noise. Based on the filtered depth and temperature, the vertical temperature gradient is computed, and the depth of the maximum gradient is kept as the target depth. Doing so, when the vehicle reaches the maximum depth, an estimate of the depth of maximum temperature gradient is available as a goal for the second phase.

During the second phase, the target depth previously computed is simply set as the desired depth. While the AUV is climbing up, the target depth keeps on being updated, so as to possibly improve the estimation of the depth of maximum gradient.

Once the target depth is reached, the third phase begins. The mean depth is first set to the previous target depth, and the desired depth is computed by adding the mean depth to the

oscillating command computed as a sine function of the time. During a whole oscillation, the temperature and depth measurements are logged, filtered and derived as previously to obtain the mean depth of the next oscillation. Once the oscillation is completed, the mean depth is updated as the previously computed target depth, and so on. In order for the old measurements not to influence the computation of the new target depth, which would prevent the AUV from reacting fast enough to variations of the depth of the thermocline, the measurements history has to be somehow reseted. Indeed, if for instance a maximum gradient of 1 °C/m is found at 60m during one hour of mission, and then the maximum gradient drops to 0.6 °C/m and dives to 80m, the determination of the depth of maximum gradient based on the whole measurement history would yield a target depth of 60m, and the AUV would miss the thermocline.

The parameters of that control algorithm are:

- The maximum depth for the first phase (diving),
- The characteristics of the filter used to remove the noise on the measurements,
- The size and management/reset method for the measurements buffer stack,
- The magnitude and frequency of the oscillations.

The maximum depth is a user-defined parameter for the simulation, and doesn't really change the behavior of the vehicle. The filter used to remove the measurement noise is decided to be a simple moving average filter in the shape of a Blackman window. Its width is a parameter that can be modified. As far as the buffer stack for the storage of the measurement is concerned, two approaches are possible. The size of the stack can be either limited and the stack used as First In First Out (FIFO) buffer, the oldest measurement being removed every time a new measurement is added, or the size can be

variable, but with a reset forced once in a while, for instance after a certain number of oscillations. Both methods have been implemented, and yielded similar results, except that a finite length buffer slows a little bit the reactions of the vehicle, while the reset method reduces the smoothness of the depth command. Finally, the magnitude and frequency of the oscillation have to be decided. To get an idea of the order of magnitude of the frequency that can be used, we first considered the dynamics of the vehicle. Based on the equations of motion discussed in V.2.1.1.2, we used the diving model to obtain the pitching frequency. Considering a zero downward velocity and velocity derivative, and assuming that the hydrodynamic forces and moments are dominated by the linear terms, the equation of the longitudinal motion can be rewritten to obtain the diving model in the form of a transfer function between pitch (output) and sternplane angle (input). This transfer function corresponds to the general form of a second order low-pass filter of which the natural frequency can be computed. In the case of the OEX, the numerical computation yields a frequency around 0.11Hz, corresponding to a period of 9s. This is the natural pitching frequency of the vehicle. This can be interpreted as that any sternplane command consisting of frequencies higher than 0.1Hz will have little effect on the vehicle pitch. The natural period also gives an idea the order of magnitude of the response time between a sternplane command and the corresponding pitch response. This leads us to consider only oscillations frequencies much smaller than the natural pitching frequency. It was decided that oscillations periods under 30 seconds were not realistic especially if the pitch is to remain limited. Moreover the choice of an oscillation period is related to the desired magnitude of the oscillations: larger oscillations require a longer period. Otherwise the AUV would be unable to achieve the correct oscillation magnitude.

### V.3.2.3. Implementation

Several controllers were implemented as Matlab functions and simulated. The differences among them were mainly related to the way the measurements buffer is managed. The controllers were implemented as three Matlab functions:

- `SimTracking` is the core of the controller. It distinguishes the three parts of the mission, and accordingly performs the required tasks. It returns the desired depth, set as the maximum tracking depth for phase 1, as the computed target depth of maximum gradient for phase 2, and as the value returned by the function “Oscillate” described hereafter for phase 3.
- `Get_Target_Depth` adds new CTD measurements to a buffer, filters its content, then, based on the filtered content, computes the depth of maximum gradient, and returns it as the target depth.
- `Oscillate` is the function generating a depth command based on the addition of an oscillation to the target depth computed by the previous function.

The functions `SimTracking` and `Oscillate` remained approximately the same for each controller implemented. Only the period and amplitude of the oscillations, that are some parameters of the function `Oscillate`, were modified to get an insight into their influence on the tracking efficiency. On the other hand, different `Get_Target_Depth` functions were implemented. The two main versions are summarized hereafter:

- One version included a large buffer, the content of which was stacked so that every new sample added makes the oldest disappear.
- One version included a smaller buffer, which can be reseted to only the last received measurements, based on the value of a “Reset” parameter.

In both cases, the length of the buffer as well as that of the Blackman window used as a mobile averaging filter to process the content of the buffer were some parameters of the function.

The implemented functions are summarized in Tables 24, 25 and 26:

<i>Function: SimTracking</i>	<i>Variable</i>	<i>Comments</i>
<i>Input Variables</i>	<b>Time</b>	Simulation time, used to generate an oscillation
	<b>z_measured</b>	CTD measurements, sent to Get_Target_Depth for processing aimed at estimating the depth of maximum temperature gradient
	<b>Temperature</b>	
<i>Output Variables</i>	<b>z_desired</b>	Desired depth for the AUV
	<b>mean_depth</b>	Mean depth of each oscillation, to be logged
	<b>error</b>	Simulation error

Table 24: Tracking Controller Main Module Implementation Summary

<i>Function: Get_Target_Depth</i>	<i>Variable</i>	<i>Comments</i>
<i>Input Variables</i>	<b>z_measured</b>	CTD measurements, for processing aimed at estimating the depth of maximum temperature gradient
	<b>Temperature</b>	
	<b>Phase</b>	Information used for the management of the measurement storage buffer, depending on the version implemented
	<b>Reset</b>	
<i>Output Variables</i>	<b>Target_Depth</b>	Depth of maximum temperature gradient
	<b>error</b>	Simulation error

Table 25: Measurements Processing Implementation Summary

<i>Function: Oscillate</i>	<i>Variable</i>	<i>Comments</i>
<i>Input Variables</i>	<b>z_measured</b>	CTD measurements, sent to Get_Target_Depth for processing aimed at estimating the depth of maximum temperature gradient
	<b>Temperature</b>	
	<b>Time</b>	Simulation time, used to generate an oscillation
<i>Output Variables</i>	<b>Desired_Depth</b>	Depth of maximum temperature gradient
	<b>error</b>	Simulation error

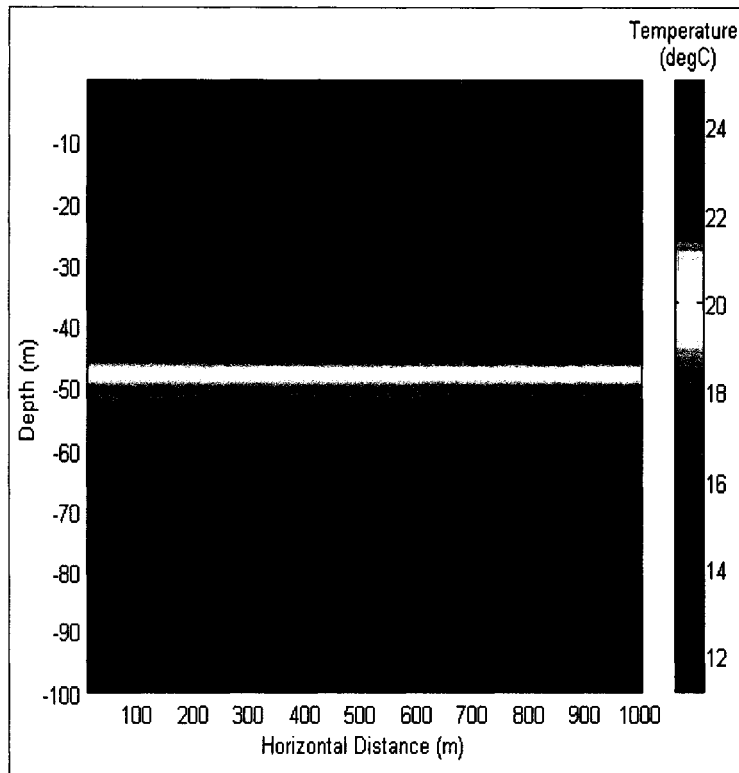
Table 26: Oscillations Generation Implementation Summary

### V.3.2.4. Testing

The controllers implemented as described previously were tested by simulation, using different parameters, so as to:

- Ensure that the control algorithm was valid,
- Get an insight into the influence of each parameter,
- Derive some conclusions about the efficiency of the controller.

First, several tests were performed using a simple temperature profile, without horizontal variation, as presented in Figure 86.



*Figure 86: Simple Temperature Map*

Using this map, any reasonable combination of parameters showed a vehicle able to locate the thermocline and oscillate around its mean depth. That was particularly efficient with the “Get\_Target\_Depth” module using a long buffer. Obviously, because the thermocline is not varying, a long measurement history is more likely to provide the same target depth. But again, that may be a drawback in the case of a varying thermocline. More thorough tests were then performed using a temperature map showing a thermocline whose depth is not constant, as shown in Figure 87.

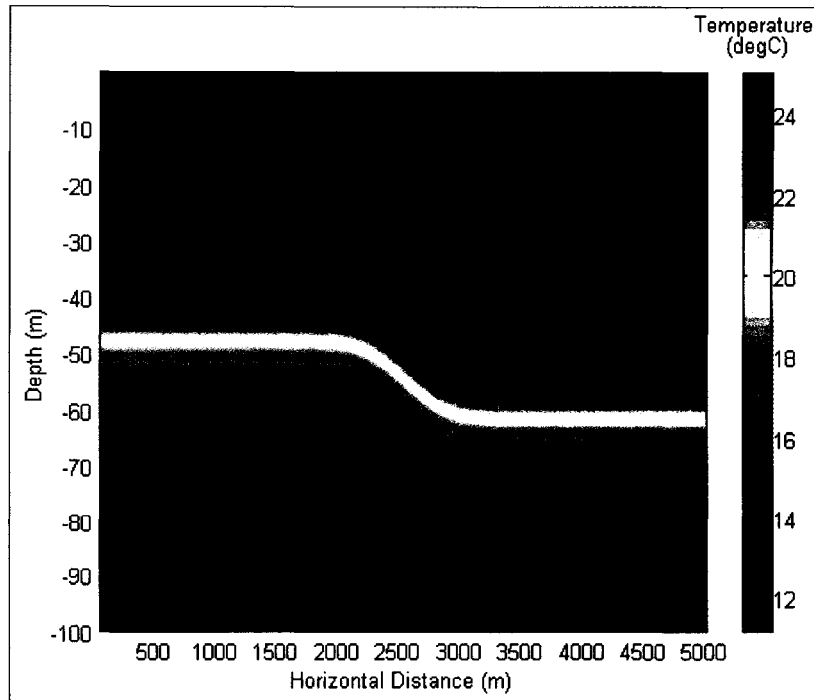


Figure 87: Complex Temperature Map showing a Diving Thermocline

One particular example of the results obtained is detailed hereafter. The parameters used for the tracking controller are summarized in Table 27.

<i>Parameter</i>	<i>Value</i>	<i>Comment</i>
Update target depth estimation during phase 2 (climbing)	NO	The target depth to begin the tracking is computed based only on measurements acquired during phase 1, when the vehicle is diving.
Oscillations period	60s	
Oscillations magnitude	8m	Peak to peak, $\pm 4m$ .
Change mean depth	Every period	The target depth is updated constantly, but the mean depth for the oscillation is set to that target depth for each period.
Reset target depth estimation buffer	Every period	

Table 27: Controller Parameters for Test Simulation

The parameters used for the simulation tool are summarized in Table 28.



<i>Parameter</i>	<i>Name</i>	<i>Value</i>
Scheduler timestep	<i>Scheduler_dt</i>	20ms
Motion simulation timestep	<i>Motion_dt</i>	20ms
CTD simulation timestep	<i>CTD_dt</i>	0.5s, No averaging
Fins controller timestep	<i>Fins_dt</i>	0.2s
Thermocline tracking controller timestep	<i>Tracking_dt</i>	0.5s
Logger timestep	<i>Logger_dt</i>	0.5s
Safety monitor timestep	<i>Safety_dt</i>	0.5s
Simulation duration	<i>Sim_duration</i>	3200s
CTD map file	<i>CTD_File</i>	../ctd/diving_ctd_50_01.mat (temperature map shown in Figure 87)
Noise level on temperature measurements	<i>NoiseLevelT</i>	0
Noise level on depth measurements	<i>NoiseLevelD</i>	0
Maximum depth for thermocline tracking	<i>Max_d_track</i>	70m
Minimum depth for thermocline tracking	<i>Min_d_track</i>	10m
Maximum safety depth	<i>Max_d</i>	100m
AUV initial forward velocity	<i>U0</i>	1.5m/s
Logger file path and filename	<i>Logger_file</i>	./Logger.txt
Summary file path and filename	<i>Summary_file</i>	./Summary.txt

*Table 28: Simulation Parameters for Controller Test*

The result of the simulation is presented in Figure 88, showing the vehicle trajectory over the temperature map.

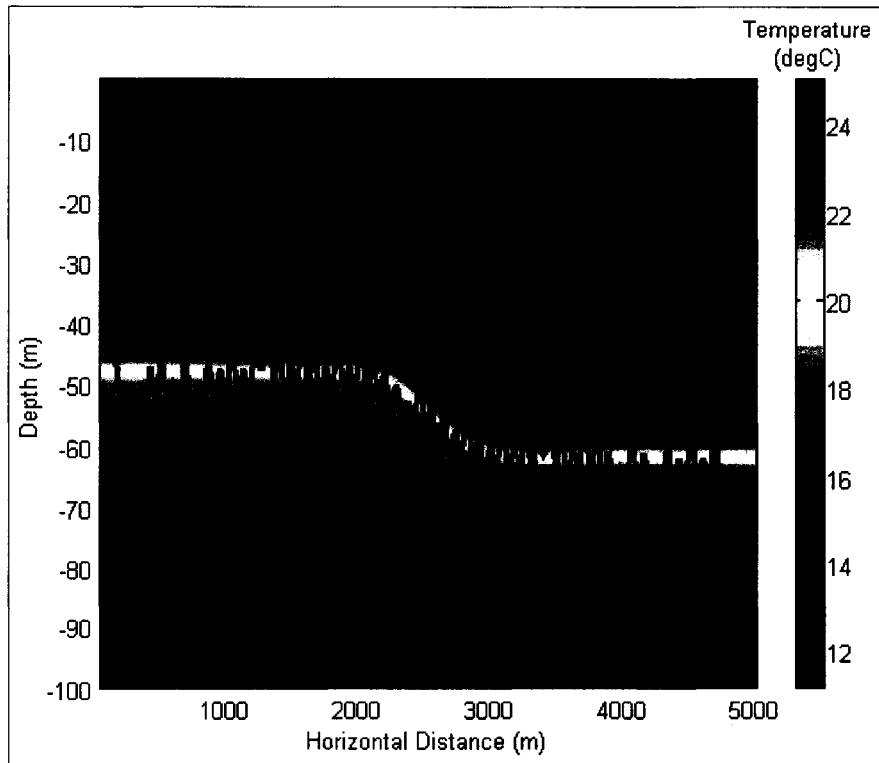


Figure 88: Simulation Results: Vehicle Trajectory over the Temperature Map

Indeed, the vehicle is able to locate the thermocline, and remain within the layer of maximum vertical temperature gradient. Even when the depth of the thermocline is changing, the vehicle remains within it, and tracks the variation with almost no delay. Although the oscillations seem to be very sharp because of the scale used for the plot, a closer look would confirm that the magnitude of each oscillation is much smaller than its wavelength. Figure 89 confirms that with such choice of oscillation magnitude and period, the vehicle is able to keep up with the depth command, with only a slight delay. Occasionally, when the difference between the mean depth of two successive oscillations is significant, the vehicle is unable to reach the magnitude of the first half of the following oscillation, but the second half is fine, providing correct measurements for the control algorithm.

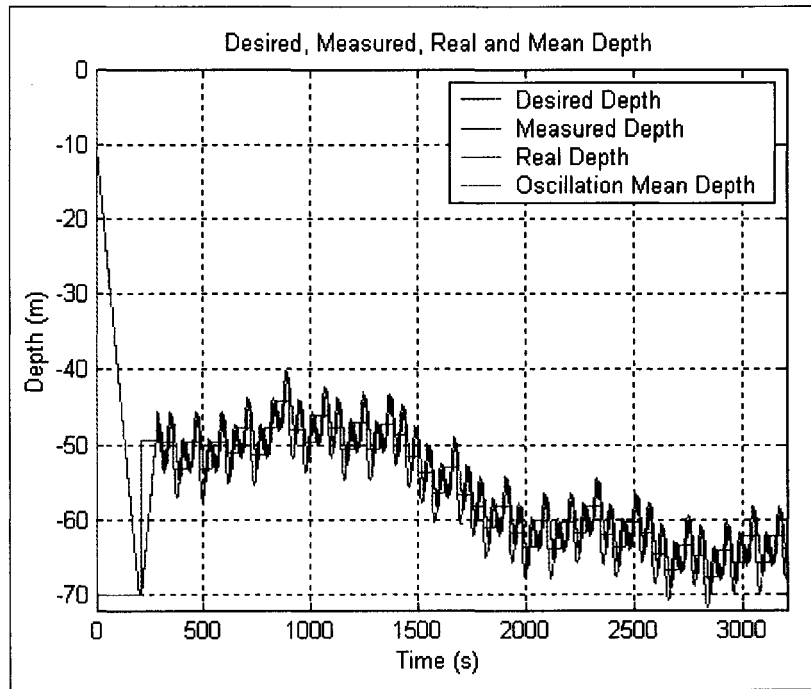


Figure 89: Depth Variables (Desired, Measured, Real and Mean)

Finally, the temperature profile measured by the vehicle is shown in Figure 90, where the different phases are evident. The profile measured by the vehicle when it dives during the first phase, and when it climbs up to the target depth during the second phase appears clearly. Then the upper oblique cluster corresponds to the vehicle tracking the thermocline during the first half of phase 3, and the lower cluster corresponds to the second half. The sparse cluster between the two previous corresponds to what the AUV senses when it dives to follow the thermocline variation.

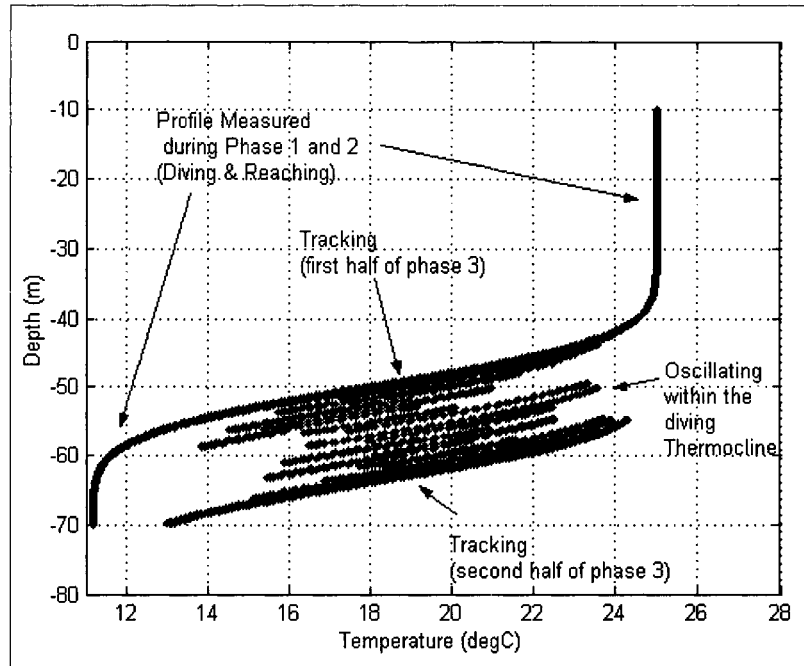


Figure 90: Temperature Profile Measured by the Vehicle

The same controller was then tested in the case of noisy measurements, through two simulations using the parameters summarized in Table 29 and 30.

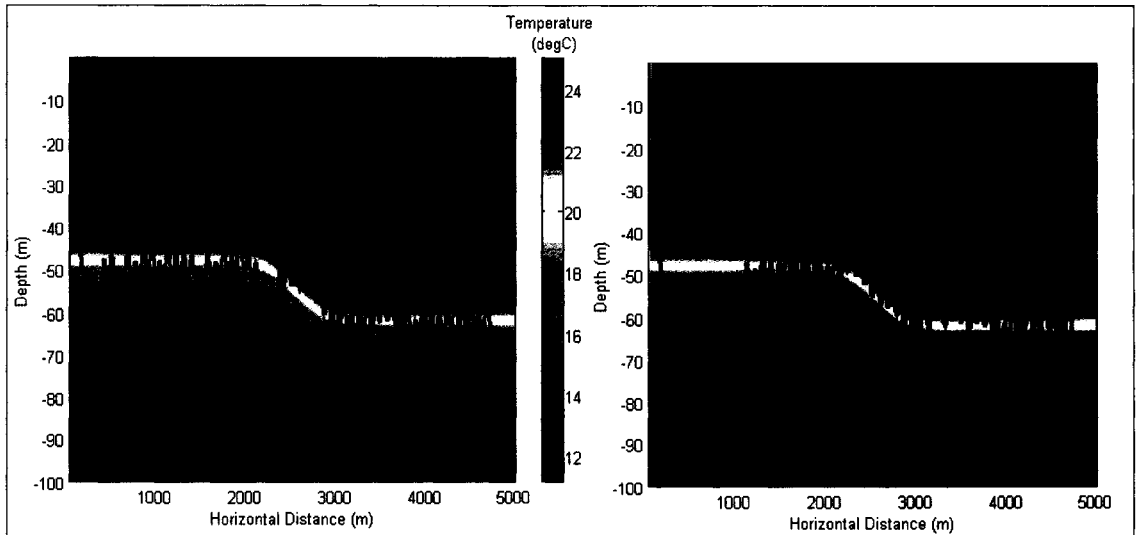
<i>Parameter</i>	<i>Name</i>	<i>Value</i>
CTD simulation timestep	<i>CTD_dt</i>	0.1s, 10 samples averaged, output rate: 1Hz
Noise level on temperature measurements	<i>NoiseLevelT</i>	$1 \cdot 10^{-3} \text{°C}$
Noise level on depth measurements	<i>NoiseLevelD</i>	1cm
All other parameters		Same as the previous simulation (see Table 28)

Table 29: Simulation Parameters for Controller Test in Low Noise Environment

<i>Parameter</i>	<i>Name</i>	<i>Value</i>
Noise level on temperature measurements	<i>NoiseLevelT</i>	$5 \cdot 10^{-3} \text{°C}$
Noise level on depth measurements	<i>NoiseLevelD</i>	5cm
All other parameters		Same as the previous simulation (see Table 29)

Table 30: Simulation Parameters for Controller Test in High Noise Environment

The result are presented in Figure 91 showing the trajectory of the vehicle over the temperature map for both simulations.



*Figure 91: Simulations Results for Noisy CTD Measurements (Low and High Noise)*

Figure 91 shows that the tracking remains fine in case of noisy measurements. On the left plot, corresponding to a small noise, the result is not significantly different from what was obtained without noise. The trajectory is only a little less smooth, and the vehicle takes a little more time to sense the diving of the thermocline. The right hand plot however shows an interesting thing: because of the noise that has probably (randomly) made the vehicle sense a smaller gradient when it crossed the thermocline, and a larger one when it was below, the target depth for tracking was initialized around 68m. The interesting thing is that the tracking algorithm sensed that the vehicle was below the thermocline, and eventually made the depth command converge towards the thermocline.

Finally, Figure 92 shows the temperature profile measured by the vehicle during the last simulation, with high-noise. The vertical thickness of the clusters shows the influence of the measurement noise on the temperature profile sensed (compare to Figure 90).

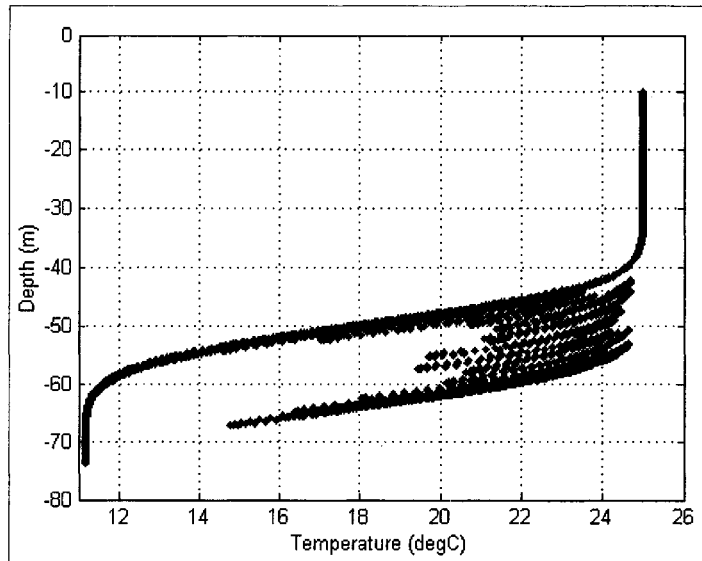


Figure 92: Temperature Profile Measured in Case of High Noise

The thermocline tracking controller was then tested using real data instead of a synthetic temperature map. The only real temperature map available was that obtained from the Thermocline Survey Experiment conducted on March 19<sup>th</sup>, 2003. Unfortunately, as explained in IV.4.3, the temperature profile measured during this experiment doesn't show any strong localized thermocline. For that reason, it was necessary to reconstruct a temperature map using other temperature data gathered at sea. As discussed IV.3.3.2, the shipboard CTD casts performed during the experiment conducted on December 18<sup>th</sup>, 2002, showed a relatively strong thermocline, with a vertical temperature gradient as large as  $0.6^{\circ}\text{C}/\text{m}$ , about 20m thick, at around 100m depth. Even if no temperature map is available for that day, it is possible to build one using the data from one of the CTD casts. Simply assuming no horizontal variation of the temperature profile, a single temperature profile measured by the CTD cast can be used as a map with zero resolution along the horizontal direction, as discussed in V.2.1.2. The temperature profile used for that test is

presented, along with the corresponding temperature gradient profile, in Figure 93.

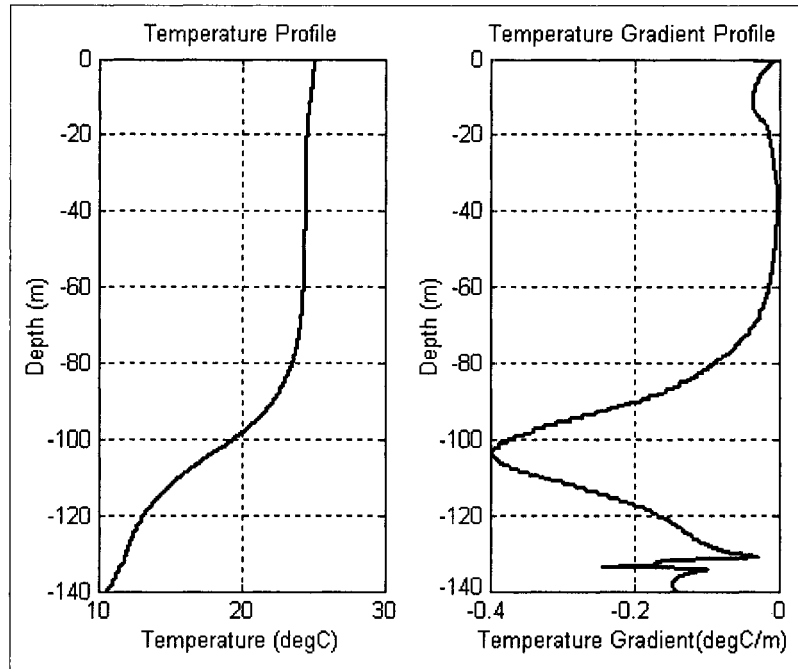


Figure 93: Real Temperature and Temperature Gradient Profiles used for Simulation

A simulation was run using this temperature map and the simulation and controller parameters listed in table 31 and 32 respectively.

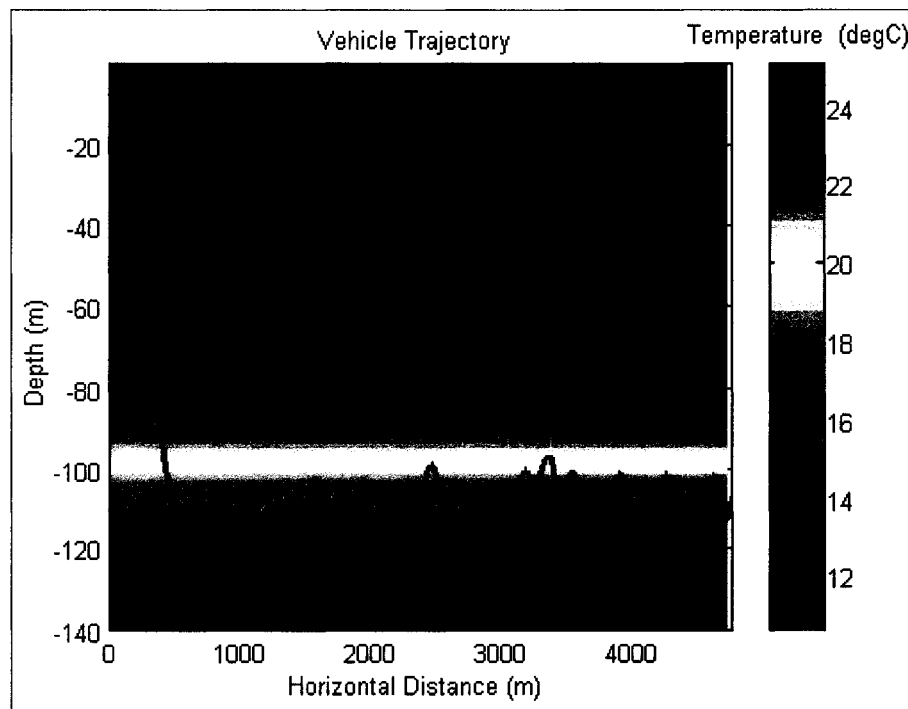
<i>Parameter</i>	<i>Name</i>	<i>Value</i>
CTD simulation timestep	<b>CTD_dt</b>	10ms, 10 samples averaging, output rate 1Hz
CTD map file	<b>CTD_File</b>	../ctd/profile_20021218.mat
Noise level on temperature measurements	<b>NoiseLevelT</b>	0
Noise level on depth measurements	<b>NoiseLevelD</b>	0
Maximum depth for tracking	<b>Max_d_track</b>	120m
Minimum depth for tracking	<b>Min_d_track</b>	10m
Maximum depth for tracking	<b>Max_d</b>	135m
Other parameters		As in table 28

Table 31: Simulation Parameters for Controller Test with Real Temperature Data

<i>Parameter</i>	<i>Value</i>	<i>Comment</i>
Oscillation period	120s	
Oscillation magnitude	8m	Peak to peak, $\pm 4m$
Other parameters		As in table 27

*Table 32: Controller Parameters for Controller Test with Real Temperature Data*

The result of the simulation are summarized in Figure 94 showing the AUV trajectory over the temperature map.



*Figure 94: Simulation Results: Vehicle Trajectory over the Temperature Map*

Figure 94 shows that the controller is able to locate the thermocline and make the vehicle oscillate around its mean depth. Once again, because of the noise, the vehicle fails to locate the thermocline during the first (diving) phase, but successfully reaches and tracks it during the third (tracking) phase. This is a conclusive test since, this time, it is not run with a synthetic temperature profile that suits our needs, but with a real temperature profile as was indeed measured during one of our at-sea experiments.



### **V.3.2.5. Conclusions**

The tests described above as well as some others not detailed here show that the control algorithm is valid. A controller issuing a depth command oscillating around the depth of maximum measured temperature gradient is able to make the AUV acquire and track the thermocline. Among the other conclusions drawn from several tests, it was evidenced that this method is efficient even with noisy measurements. The selection of parameters, mainly oscillation magnitude and measurements buffer management method mainly depends on the kind of efficiency desired. A compromise has to be made between smoothness of the trajectory and response time to a change of depth of the thermocline. The smoother the trajectory, the longer it takes for the vehicle to reacquire the thermocline after its depth has changed. In case of very noisy measurements, it may be desirable to increase the magnitude of the oscillations as well as the size of the measurements buffer so that the algorithm can use a larger amount of data to estimate the mean depth of the thermocline during each oscillation.

### **V.3.3. Thermocline Tracking Sternplane Controller**

This section describes the design, implementation and testing of a thermocline tracking sternplane controller that makes the AUV react to sensor measurements in order to remain around the mean depth of the thermocline.

#### **V.3.3.1. Approach**

The approach considered for the design of that controller was to bypass the phase of modeling of an oscillating trajectory and the reasoning aimed at determining the mean

depth of the thermocline based on measurements acquired from a local profile. Instead, we tried to implement a reflex for the vehicle. The idea was to command the vehicle to the middle of the thermocline, and have it react by inverting its motion as soon as it senses that it is leaving the region of interest. The emphasis was put on a link, as direct as possible, between sensing and action. To that end, the vehicle doesn't really consider the concepts of depth and thermal structure. Rather, the acquired measurements are used to derive an indication of how appropriate the current motion is. Based on this quantity, a simple decision is made to either continue with the same motion, slow down the motion to confirm what has been sensed, or completely reverse the motion.

The decision algorithm, in a simplified approach, can be thought as:

- When the vertical temperature gradient increases, continue in the same (vertical) direction,
- When the vertical temperature gradient decreases, reverse the (vertical) direction.

### **V.3.3.2. Design**

One of the easiest ways to design a controller based on the above decision algorithm was thought to use a Fuzzy Inference System (FIS). Indeed, it is, as already discussed in II.2, one of the most powerful tools to handle imprecise information and goals, as well as human reasoning mimicry. A controller with a fuzzy-logic core was then designed. It was decided to base the decision-making on the variation of the temperature gradient and the depth rate of the vehicle. Based on the consideration of these inputs, a sternplane angle output is generated. As a reminder, a positive sternplane angle makes the vehicle pitch negatively and go down, and vice versa.

The underlying reasoning is based on four different cases (Figure 95):

- (1) If the vehicle is going up and the temperature gradient decreases, it is leaving the top of the region to track. It should therefore go down, and the sternplane angle command is positive,
- (2) If the vehicle is going up and the temperature gradient increases, it is approaching the region to track from below. It should therefore keep going up, and the sternplane angle command is negative,
- (3) If the vehicle is going down and the temperature gradient decreases, it is leaving the bottom of the region to track. It should therefore go up, and the sternplane angle command is negative,
- (4) If the vehicle is going down and the temperature gradient increases, it is approaching the region to track from above. It should therefore keep going down, and the sternplane angle command is negative.

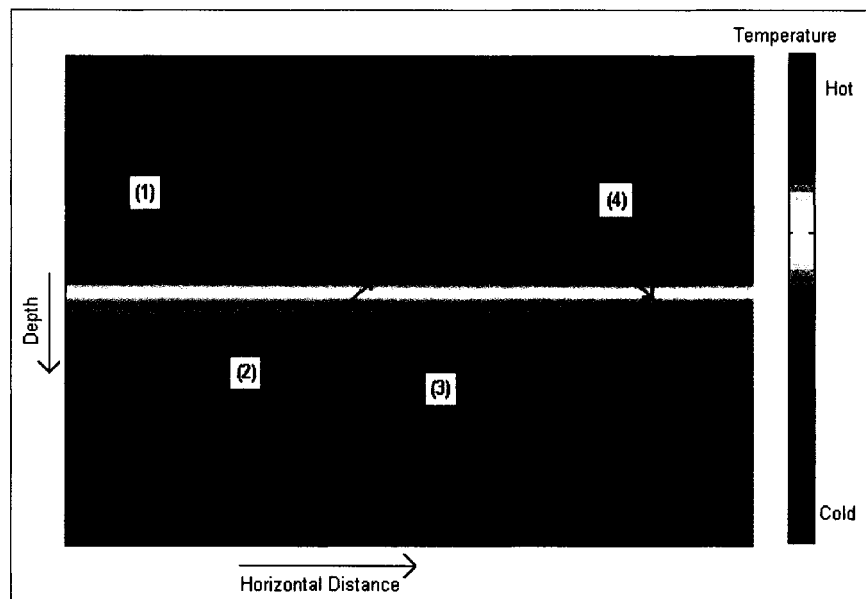


Figure 95: Simplified Reasoning: Four Cases

The fuzzy part handles how much the temperature gradient variation, depth rate and sternplane angle are positive or negative. Several fuzzy engines were considered. Only the common characteristics are discussed here. The characteristics specific to the implementation of a particular engine are reported in V.3.3.3.

Let us first consider the inputs fuzzification and output defuzzification. The inputs are the gradient variation between the two last measurements, and the depth rate, as measured by the CTD. The depth rate mainly depends on the characteristics of the vehicle and the command applied to its control surface. Therefore, it always remains within the same range. This is not the case for the temperature gradient variation, which depends on the vehicle motion and the characteristics of the water column. To eliminate that dependence, the gradient variation input is normalized. As far as the output range is concerned, it is limited by the maximum authorized sternplane angle, which is set to  $20^\circ$ . Then we defined the range of each input and output as follows:

- Input normalized temperature gradient variation:  $[-1.5;1.5]$  (  $^\circ\text{C}/\text{m}/(\text{iteration})$  ).
- Input depth rate:  $[-1;1]$  (m/s). Possible values outside this range are considered  $\pm 1\text{m/s}$ .
- Output sternplane angle:  $[-20;20]$  ( $^\circ$ ).

All membership functions for the fuzzifier and defuzzifier were chosen trapezoidal or triangular. The chosen methods for fuzzification and defuzzification are listed in Table 33.

<i>Operator</i>	<i>Method</i>
Intersection/Logical AND	Min
Union/Logical OR	Max
Implication <sup>(1)</sup>	Min
Defuzzification	Centroid (center of gravity)

*Table 33: Characteristics of the Fuzzy Engine*

*(1) The implication method is listed here although it is not strictly a property of the fuzzifier or defuzzifier, but rather of the fuzzy inference engine.*

For the work considered here, a Mamdani-type FIS was used in a first time, mainly because, although less computationally efficient, its design is more intuitive than Takagi-Sugeno-Kang fuzzy inference systems[55].

This FIS is the core of the controller used for tracking. It only has to be surrounded by a module that creates the formatted input. Successive CTD measurements are logged in a two-measurement buffer, from which instantaneous temperature gradient and depth rate are computed. Using the temperature gradient of the previous iteration, the temperature gradient variation is computed and normalized. Then the input are fed to the FIS which, after evaluation, outputs a desired sternplane angle which is returned as a command to the simulation tool.

This fuzzy controller is only used for the tracking phase. Indeed, to efficiently react to the thermocline, the vehicle has to already be within that layer. There is no need to use a fuzzy controller to get the vehicle to within the thermocline. Rather, simple controllers are used during the two first phases of the mission. For these phases, the control algorithm is very similar to what was described in V.3.2. A desired depth command is set to the maximum tracking depth during the first phase, so that the vehicle can sample the whole water column. This desired depth command is translated to a sternplane position

using the sternplane controller available from the simulation tool. During that phase, the CTD measurements are logged and processed as described in V.3.2, in order to compute the target depth for the second phase. In a same way, during the second phase, the desired depth is set equal to the target depth, and the sternplane controller is used to translate that desired depth into a sternplane position.

### **V.3.3.3. Implementations**

Several similar controllers were implemented, all of them using the three following functions:

- SimTracking is the core of the controller. It distinguishes the three phases of the mission, and accordingly performs the required tasks.
- Get\_Target\_Depth adds new CTD measurements to a buffer, filters its content, then, based on the filtered content, computes and returns the depth of maximum gradient. This function is very similar to that described in V.3.2.3.
- React is the function that implements the reflex based on the use of the FIS described above for the third phase (tracking).

During the first phase, the function SimTracking sends each new CTD measurement to the function Get\_Target\_Depth so that it can compute the depth of maximum gradient where the tracking phase is to begin. Meanwhile, it uses the sternplane controller available from the simulation tool to control the vehicle to its maximum search depth. Once the vehicle has switched to phase 2, SimTrack keeps on sending CTD updates to Get\_Target\_Depth, and again, uses the available sternplane controller to command the vehicle to the target depth. Then, the vehicle switches to phase 3, and SimTracking calls

the React module, providing it with every CTD measurement and getting from it the desired sternplane angle command that is returned to the simulator. Meanwhile, the React function processes the inputs, evaluates the FIS and returns its output as the desired sternplane angle, as described in V.3.3.2.

Only a particular case of controller implemented is described further here, because, although simple, this controller provides promising results. It is based on a very simplified fuzzy engine. The fuzzy sets have been simplified to contain only two sets for each variable, either positive (P) or negative (N). The corresponding membership functions are all trapezoidal or triangular, as illustrated in Figure 96.

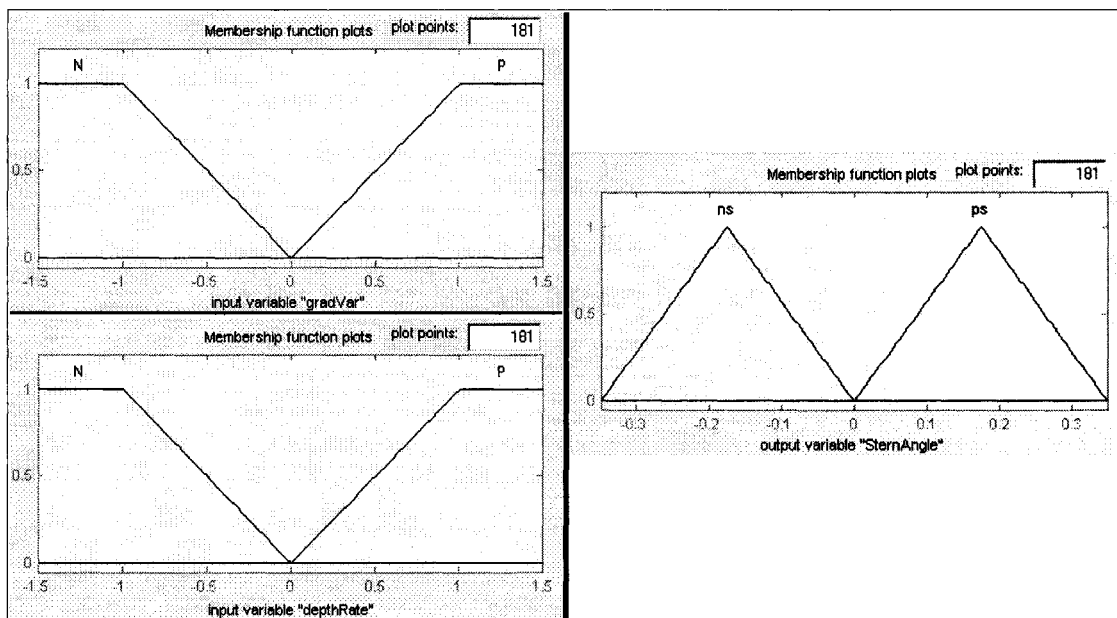


Figure 96: Simple Membership Functions

Then the decision-making is based on the fuzzy rules described in Table 34.

<b>Temp Gradient Variation</b>	<b>Depth Rate</b>	<b>N</b>	<b>P</b>
<b>N</b>		<b>P</b>	<b>N</b>
<b>P</b>		<b>N</b>	<b>P</b>

Table 34: Inference Rules

Figure 96 and Table 34 show that in this particular simplified case, the controller is no longer fuzzy. Indeed, looking only at the rules in Table 34, writing T the temperature gradient variation, D the depth rate and S the sternplane angle, and considering P as true and N as false, the control logic can be rewritten as:

$$S = P \text{ if } T = D \quad (\text{Eq. 5.26})$$

$$\text{or } S = (T = D) \quad (\text{Eq. 5.27})$$

which can then be expanded as:

$$S = (\overline{T.D}) + (T.D) \quad (\text{Eq. 5.28})$$

Where . is the logical AND, + is the logical (inclusive) OR, and  $\overline{\phantom{x}}$  is the logical NOT.

Using twice De Morgan's law, the above equation 5.28 can be rewritten as:

$$S = \overline{(\overline{T.D} + D.T)} = T \text{ NXOR } D \quad (\text{Eq. 5.29})$$

where NXOR is the Not eXclusive OR logical operator.

Then, because of the chosen membership functions and defuzzification method, the output is always,  $\pm 0.5 \cdot \delta_{s, Max}$ , half the maximum sternplane angle, and the sign only



depends on whether S is P or N, and is obtained from equation 5.29.

The control law can then be rewritten as:

$$\delta_s = \frac{\delta_{s, Max}}{2} \times [\text{sign}(T) \times \text{sign}(D)] \quad (\text{Eq. 5.30})$$

This is verified with the plot of the control surface shown in Figure 97.

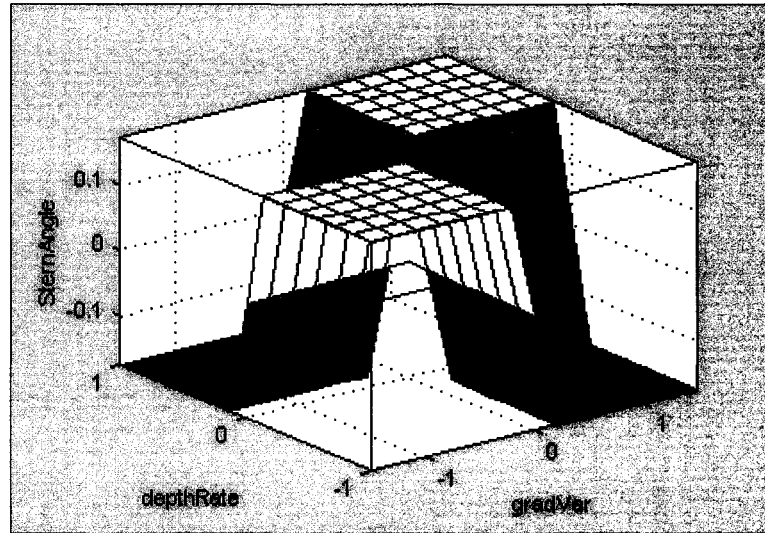


Figure 97: Control Surface

#### V.3.3.4. Testing

The reactive controller described above has been tested by simulation using the controller parameters summarized in Table 35.

<i>Parameter</i>	<i>Value</i>	<i>Comment</i>
Buffer length for Get_Target_Depth module	Infinite	Because that module is only used during the first two phases, there is no need to reset the buffer or implement it as a finite length stack.
Filter for Get_Target_Depth module buffer	10 points Blackman window	
Fuzzy engine	As described above	

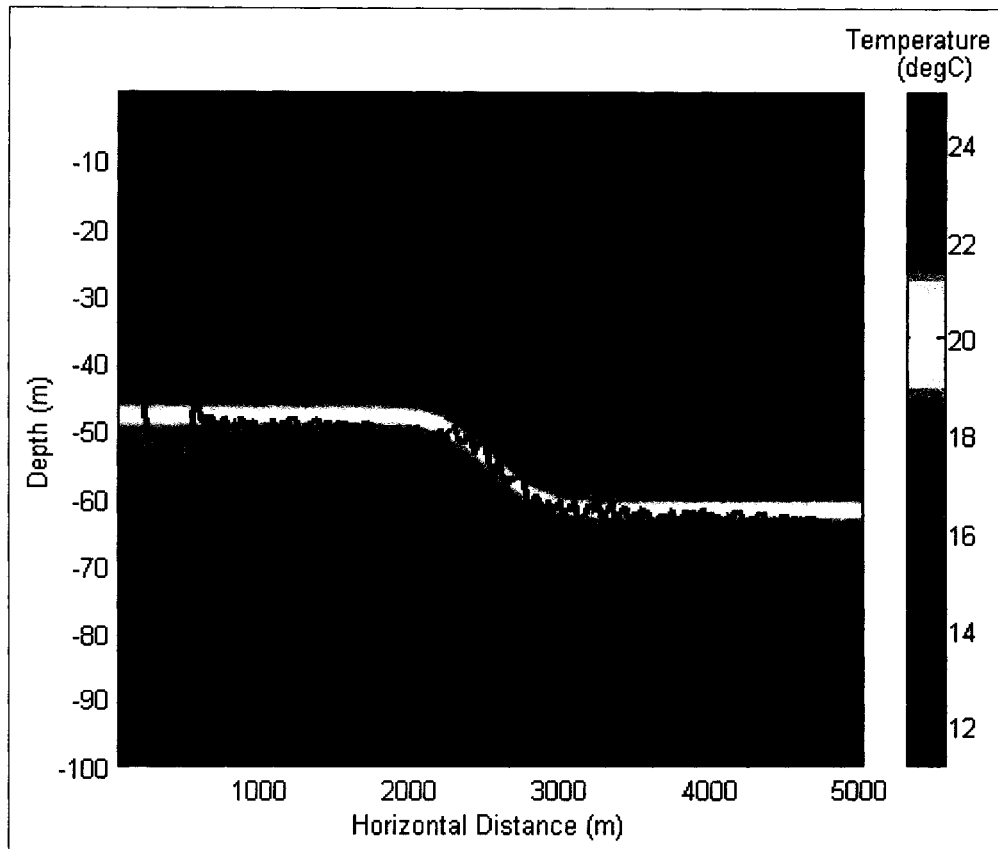
Table 35: Controller Parameter for Test Simulation

Then the simulation was set up with the parameters summarized in Table 36 and run.

<i>Parameter</i>	<i>Name</i>	<i>Value</i>
Scheduler timestep	<i>Scheduler_dt</i>	20ms
Motion simulation timestep	<i>Motion_dt</i>	20ms
CTD simulation timestep	<i>CTD_dt</i>	0.1s, 20 samples averaging, output rate 0.5Hz
Fins controller timestep	<i>Fins_dt</i>	0.5s
Thermocline tracking controller timestep	<i>Tracking_dt</i>	0.5s
Logger timestep	<i>Logger_dt</i>	0.2s
Safety monitor timestep	<i>Safety_dt</i>	0.5s
Simulation duration	<i>Sim_duration</i>	3200s
CTD map file	<i>CTD_File</i>	../ctd/diving_ctd_50_01.mat (temperature map shown in Figure 87)
Noise level on temperature measurements	<i>NoiseLevelT</i>	0
Noise level on depth measurements	<i>NoiseLevelD</i>	0
Maximum depth for thermocline tracking	<i>Max_d_track</i>	80m
Minimum depth for thermocline tracking	<i>Min_d_track</i>	10m
Maximum safety depth	<i>Max_d</i>	100m
AUV initial forward velocity	<i>U0</i>	1.5m/s
Logger file path and filename	<i>Logger_file</i>	./Logger.txt
Summary file path and filename	<i>Summary_file</i>	./Summary.txt

Table 36: Simulation Parameters for Controller Test

The results are summarized in Figure 98, that shows the vehicle trajectory over the temperature map.



*Figure 98: Simulation Results: Vehicle Trajectory over the Temperature Map*

Figure 98 shows that, in the case simulated here, this controller is really efficient. The vehicle is able to accurately locate the thermocline and remain less than 2m around the depth of maximum gradient. Moreover, the reaction of the vehicle when the thermocline dives is very fast. At worst, in the middle of the thermocline diving phase, the vehicle is only 4m too shallow, and it immediately dives to return in the middle of the thermocline. The oscillations, although less regular than that obtained with the controller described in V.3.2, are much smaller, due to a faster reaction, as expected. Again, although the oscillations seem to be sharp because of the scale used for the plot, a closer look would

confirm that the magnitude of each oscillation is much smaller than its wavelength.

Figure 99 shows the time history of several simulation variables.

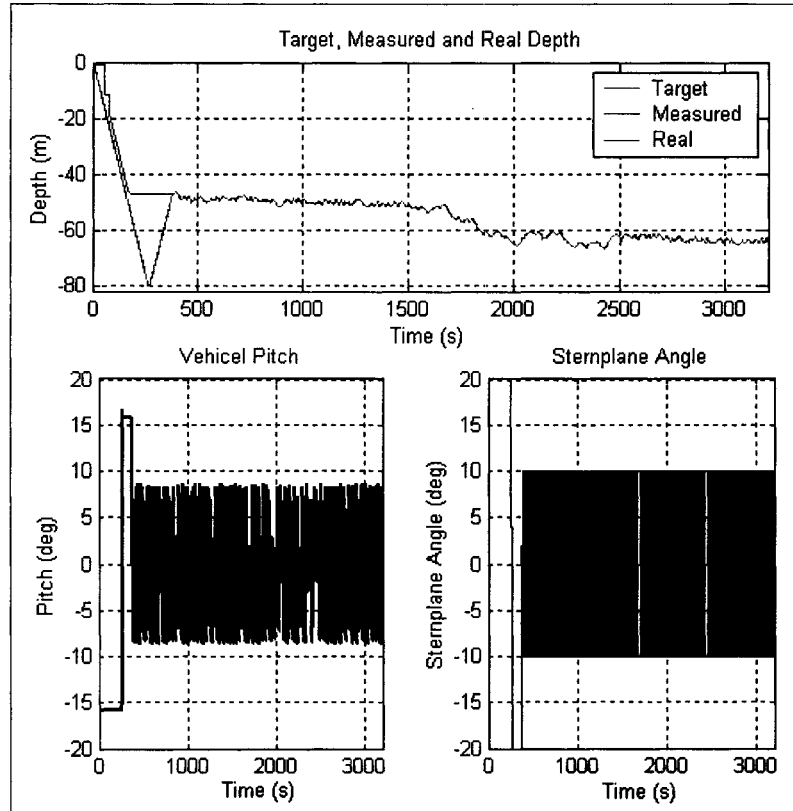


Figure 99: Depth, Pitch and Sternplane Angle vs. Time

The upper plot in Figure 99 shows the setting of the target depth during phase 1, just above the middle of the thermocline, and the depth of the vehicle reaching that target and tracking the depth of the thermocline. The lower left plot shows that the vehicle pitch remains limited, usually a few degrees, and always less than  $\pm 8^\circ$ , during the tracking phase. This pitch is approximately half of what was obtained with the controller described in V.3.2.4. This is an advantage since the AUV CTD is expected to obtain more accurate measurements when the vehicle pitch is limited. Finally, the lower right plot shows that, as expected, the simplified fuzzy engine described in V.3.3.2 and used for the third phase

always outputs a sternplane angle command of  $\pm 0.5 \cdot \delta_{s, Max} = \pm 10^\circ$ .

Finally, as for the previous simulation, it is interesting to look at the temperature profile measured by the vehicle, in Figure 100.

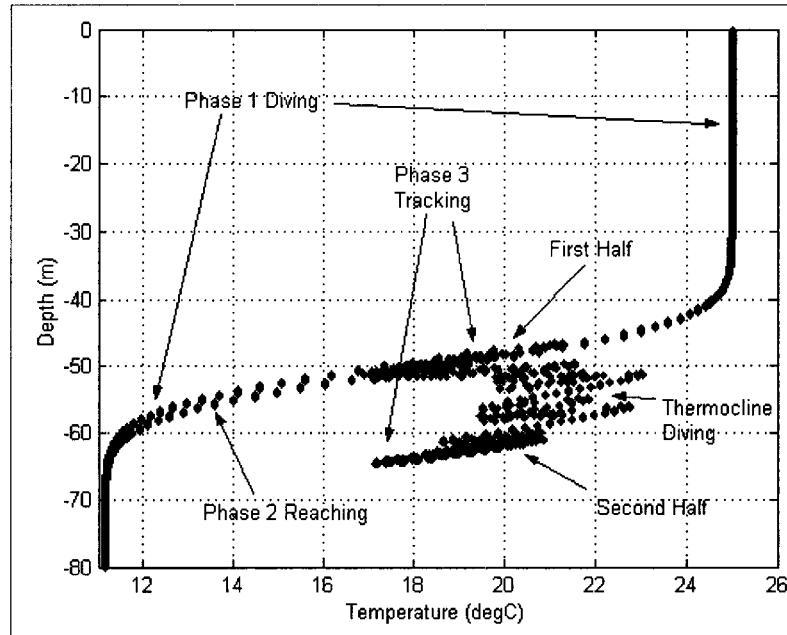


Figure 100: Temperature Profile Measured by the Vehicle

Again, in Figure 100 above, the different phases of the mission are evident. The profile measured by the vehicle when it dives during the first phase, and when it climbs up to the target depth during the second phase appears clearly. The upper oblique cluster corresponds to the vehicle tracking the thermocline during the first half of the third phase, and the lower cluster corresponds to the second half. The sparse cluster between the two previous corresponds to what the AUV senses when it dives to follow the thermocline variation. The comparison of Figure 90 (or 92) and 100 shows that the reactive controller discussed here is more efficient as far as keeping the vehicle in the middle of the thermocline is concerned, as evidenced by the length of the oblique clusters. A shorter

cluster indicates that the vehicle more often remained in the same region of the profile. Moreover, a closer look at Figure 100 confirms that the part of the profile the AUV sampled is indeed that where the temperature gradient is maximum.

### **V.3.3.5. Conclusions**

The above test, as well as some other not described here, confirm that this control method is an interesting alternative to that described in V.3.2. Among the advantages are an intuitive design and tuning, and a better efficiency in terms of tracking accuracy, response time to a depth variation in the thermocline and limited vehicle motion. Moreover this method also has the advantage to be more model-free than the previous one. No assumption regarding the trajectory that the vehicle is able to achieve is necessary. On the contrary, the vehicle may perform any arbitrary motion, determine the appropriateness of that motion, and then decide either to continue with or reverse the motion. Unfortunately, the simple controller detailed above is not as efficient in case of significant measurement noise. More complex fuzzy engines are currently under investigation to improve the noise immunity. Nevertheless the main drawback of that method is that once the AUV has completely lost the thermocline, it is unlikely to be able to reacquire it by simply evaluating the appropriateness of its motion. Indeed, once the vehicle arrives in the mixed layer or in the isothermal layer, the measured temperature gradients are not significant enough for the vehicle to determine a preferred direction of travel. Some tests performed with a controller poorly tuned showed that the mean depth of the vehicle doesn't necessary converge towards that of the thermocline. In such cases, we are not facing poor performances but really a failure to achieve the mission goal.

## ***V.4. Comparison of both Methods and Conclusions***

Overall, two control methods have been considered, and corresponding controllers have been successfully implemented and tested by simulation. The results confirmed that both methods are valid, and provided some insight into the advantages and limitations of each method, as well as some guidelines for further improvements.

A comparison of both methods was performed, which led to the following conclusions:

The first method consisting of a trajectory in the form of some oscillations whose mean depth is derived from an analysis of the measurements provides a systematic tracking method. Nevertheless, the performances are limited by the choice of the predefined oscillation magnitude and period, as well as the measurements buffer management method. In cases the parameters are not optimally set, the response to a varying thermocline is slowed down, which reduces the overall tracking efficiency. Moreover, the accuracy of the tracking, measured by how close the vehicle remains to the middle of the thermocline is directly influenced by the magnitude of the oscillations. The main advantage is that even with non-optimal parameters, the tracking remains efficient. Moreover, in case of high noise, the tracking is still made possible by the choice of larger oscillations and a longer measurements buffer. As far as the reactive controller is concerned, its efficiency is much better than that of the previous controller, when the vehicle is in the thermocline. Indeed, the AUV remains closer to the middle of the thermocline and is able to track depth variations in the thermocline faster than the previous controller. Nevertheless, the disadvantage of that method is that an incorrect choice of parameters, particularly update rate, either for the controller itself or for the

CTD measurements, usually leads to a vehicle simply oscillating around a constant depth, unable to track any change in the depth of the thermocline. These parameters are somehow tricky to fine-tune. Moreover, the simple controller implemented as detailed above has a limited efficiency in presence of significant noise. Finally, with that controller, high noise does not slow down the tracking by increasing the time it takes for the vehicle to reacquire the thermocline once it has lost it, but rather makes the vehicle totally unable to reacquire the thermocline. In such cases, the vehicle either continues around a slowly varying depth, or the mission is aborted by the safety monitor when the vehicle surfaces or hits the maximum safety depth.

The main conclusion is that a controller based on the algorithm generating an oscillating trajectory can be used on the AUV to provide some preliminary results without too much work. It is only necessary to derive a more systematic way to fine-tune the various parameters. On the other hand, a controller based on the implementation of a reflex is not yet robust enough for use on an AUV. More work is required to improve the robustness and noise immunity.

Finally, the idea of designing a controller using a differential CTD sensor, as discussed in V.3.1, is currently under investigation. It is believed that such a method can provide interesting results in case a differential sensor can be mounted on the vehicle. Simulations will be used to assess the efficiency of such a control method, and evaluate the performances required for the differential sensor.



## **VI. Conclusions and Future Work**

In this chapter, we summarize the thesis work and present the main conclusions and guidelines for future work.

### ***VI.1. Summary of the Thesis Work***

As part of the work described in this thesis, three separate tasks have been achieved:

- One of FAU's Ocean Explorer AUV has been successfully upgraded to a new OEX-D version. A new main computer has been installed, and its operating system and software, consistent with the other AUVs of the department of Ocean Engineering, have been integrated. The OEX-D now offers a convenient programming and operating software interface. Several tests and at-sea missions have confirmed that the vehicle is now fully operational. From these tests, a few weaknesses or possible ways of improvements have been identified, and significant problems encountered have been fixed.
- Once operational, this new vehicle was used in three Thermocline Survey Experiments, intended to gather certain data relevant to the characterization of the thermocline. After encountering several problems, which were identified and resolved, these missions provided plenty of data to work with. Particularly, a temperature map of a vertical water slice has been computed from the temperature and position data of the

AUV. This validates the mission planning design and data processing method that can be used to gather further data during similar Thermocline Survey Experiments. Moreover, the data acquired during these experiments, and particularly the reconstructed temperature map, provides interesting background input to simulations aimed at designing some feature-relative navigation controllers based on temperature sensing.

- Finally, a simulation tool has been designed, whose primary purpose was to investigate the design of a thermocline tracking controller for the AUV. This tool provides a simple, yet useful, 3 degrees of freedom longitudinal vehicle motion simulation, as well as all the functionalities required to emulate a CTD sensor measuring the AUV depth and reading the temperature. The temperature readings come from a lookup map either based on real or synthetic data. This tool has then be used to evaluate the performances of several Thermocline Tracking Controllers. Mainly, two kind of controllers have been designed: one is a depth controller based on a model of an oscillating trajectory, whose mean depth is computed from the analysis of temperature and depth measurement history that provides the depth of maximum temperature gradient. The other is a more reactive sternplane controller that makes the AUV evaluate the appropriateness of its motion as far as the tracking of a thermocline is concerned, and, based on that evaluation, decides either to continue with the same motion or reverse it. Both methods have proven to be valid, and several simulations allowed the identifications of the strengths and weaknesses of each method. Particularly, the control method based on a model of an oscillating trajectory has proven to be robust, although its efficiency, in terms of accuracy of the tracking, is

limited. On the other hand, the reactive controller has shown some promising results as far as the accuracy and reactivity are concerned, but its robustness and noise immunity need to be further improved. Some possible ways of improvement are currently under investigation. Finally, a third control method, based on the use of a differential temperature sensor – or gradiometer – is currently considered.

## ***VI.2. Main Conclusions and Future Work***

Overall, as far as the goal of the thesis – addressing the problem of thermocline tracking with an autonomous underwater vehicle – is concerned, an AUV is now operational, and different methods to control it for thermocline tracking purposes have been developed. The main conclusions drawn from the thesis work can be discussed separately for each of the three tasks considered: upgrade of one of the Ocean Explorer AUVs, its use in experiments aimed at gathering oceanographic data relevant to the thermocline, and design of a control algorithm for the tracking of a thermocline.

Although a few improvements remain to be implemented on the Ocean Explorer D, it is now ready to be used in oceanographic missions. Particularly, the background requirements for the consideration of a thermocline tracking have been met, and the implementation of a tracking controller is now possible. Meanwhile, an efficient method for programming Thermocline Survey Experiments and processing the acquired data has been validated, and their use successfully provided a temperature map of a water column. Finally, two control methods are now available for tracking a thermocline, one requiring very little work to be ready to be implemented in the AUV software. The other method is promising, but not yet ready for implementation. Particularly, the most significant

requirements for a successful thermocline tracking, aside from an efficient controller, have been identified, of which the most important are probably related to the CTD data. It has been shown that not only accuracy but also high throughput of the CTD measurements were the main requirements for the enhancement of the ability of the control algorithm to successfully locate and follow the thermocline.

Finally, some guidelines for future work have been drawn. First a few problems remain to be fixed with the OEX-D, including the speed controller failure discussed above. Moreover, one of the problems not previously mentioned, but considered important, is that of logger failure. It has been seen from time to time that the AUV, although able to correctly perform a mission, was unfortunately unable to record data because no logger output file was created. Since more often than not the main purpose of a mission is to record data, the absence of a logger leads to a failure to meet the goal of an experiment. For that reason, an important improvement is required, and can be brought by the addition of a software procedure that simply checks that the logger output file is correctly created, and aborts the mission if not. Obviously, a better solution would be to find the reason for which the logger creation sometimes fails. Then, a few other Thermocline Survey Experiments should be run to provide a variety of temperature maps with which thorough tests of thermocline tracking controllers could be performed. Indeed, as of now the temperature structure of the water column for which a map has been acquired does not really show a profile useful for thermocline tracking controller tests. It is believed that running the same experiments later in the year, probably in fall or early winter, will provide more usable results. Finally, the thermocline tracking algorithms have to be improved, as already discussed. Then, once satisfactory results are achieved with the

simulation tool designed as part of the work described in this thesis, the controller should be implemented as part of the AUV software, so that it could undergo extensive tests with the Hardware In The Loop simulator. Once fully tested, and particularly once the failure behaviors have been verified to ensure that the vehicle safety is preserved, real at-sea tests could be planned. At-sea missions will probably provide some more information for the improvement of the controller.

Finally, when the controller is fully operational, the vehicle can be used in thermocline tracking mission, that will enable an accurate localization and characterization of the thermocline, as well as the measurement of various parameters relevant to oceanographic studies along that layer interface. This would undoubtedly improve the knowledge and understanding of several oceanographic phenomenon, as well as acoustic and biological properties of layered oceans, to name a few.

# Appendix

## ***Summary File Generated by the Simulation Tool***

Hereafter is an example of a summary file generated by the thermocline tracking simulation tool for a simple test run.

\_\_\_\_\_ File: Summary.txt \_\_\_\_\_

Simulation started 19-May-2003 16:39:24.

Parameters list follows (all units are SI):

Scheduler\_dt: 0.050000

Motion\_dt: 0.050000

CTD\_dt: 0.100000

Fins\_dt: 0.200000

Tracking\_dt: 0.500000

Logger\_dt: 0.500000

Safety\_dt: 0.500000

Sim\_duration: 3200.000000

CTD\_File: ../ctd/diving\_ctd\_dx50\_dz01.mat

NoiseLevelT: 0.050000

NoiseLevelD: 0.005000

Summary\_file: ./Summary.txt

Logger\_file: ./Logger.txt

Initial forward velocity (U0): 1.500000

Min depth for tracking (Min\_d\_track): 20.000000

Max depth for tracking (Max\_d\_track): 80.000000

Max safety depth (Max\_d): 100.000000

No other parameters.

- 16:39:24.38 : Checking parameters...
- 16:39:24.38 : Parameters checked successfully.
- 16:39:24.38 : Logger output file Logger.txt created successfully
- 16:39:24.41 : CTD map file ../ctd/diving\_ctd\_dx50\_dz01.mat loaded successfully
- 16:39:24.43 : Simulation begins now.
- 16:39:24.51 : SimTracking reports: Switching to phase 1.
- 16:39:37.78 : SimTracking reports: Switching to phase 2. Target depth: 70.0119
- 16:39:37.81 : SimTracking reports: Switching to phase 3.
- 16:42:35.39 : Simulation finished. No error.

## Bibliography

- [1] Rhodes W. Fairbridge, "The Encyclopedia of Oceanography", a new Reinhold Encyclopedia of Earth Sciences, Reinhold Publishing Corporation, 1966.
- [2] Georges L. Pickard and William J. Emery, "Descriptive Physical Oceanography, an Introduction", Butterworth-Heinemann Publishers Ltd. , 5<sup>th</sup> edition, 1990.
- [3] Robert J. Urick, "Principles of Underwater Sound", McGraw-Hill Professional Publishing, 3<sup>rd</sup> edition, April 1983.
- [4] Robert. J. Urick, "Sound Propagation in the Sea", Peninsula Publishing, December 1982.
- [5] Littoral Ocean Observing and Predictive System (LOOPS) Home Page :  
[www.opl.ucsb.edu/loops.html](http://www.opl.ucsb.edu/loops.html)
- [6] Stephen Pond and George L. Pickard, "Introductory Dynamical Oceanography", Butterworth-Heinemann Publishers Ltd. , 2<sup>nd</sup> edition, 1983.
- [7] Gerhard Neuman and Willard J. Pierson Jr., "Principles of Physical Oceanography", Prentice-Hall Inc. , 1966.
- [8] Alyn C. Duxbury and Alyson B. Duxbury, "An Introduction to the World's Ocean", McGraw-Hill Professional Publishing, 4<sup>th</sup> edition, 1993.
- [9] Geoffrey K. Vallis, "Thermocline Theories and WOCE: A Mutual Challenge", International WOCE Newsletter, no. 39, August 2000.



- [10] A. A. Bennett and J. J. Leonard, "A Behavior-Based Approach to Adaptive Feature Detection and Following with Autonomous Underwater Vehicles", *IEEE Journal of Ocean Engineering*, Vol. 25, no. 2, pages 213-226, April 2000.
- [11] E. Burian, D. Yoerger, A. Bradley and H. Singh, "Gradient Search with Autonomous Underwater Vehicles using Scalar Measurements", *Proceedings of the 1996 Symposium on Autonomous Underwater Vehicle Technology AUV'96*, pages 86-98, 1996.
- [12] Y. Le Page, K. Holappa, M. Dhanak, "Active Turbulence Following using an AUV: Direct In-Situ Measurement of Ambient Turbulence and Self-Propelled Vehicle Wake with Intelligent Control of the Measurement Platform", *Department of Ocean Engineering, Florida Atlantic University*, September 2000.
- [13] Robert A. Regan, "Autonomous Minehunting and Mapping Technologies Program – Autonomous Maneuvering Capabilities", *Proceedings of the 1996 MTS/IEEE OCEANS'96 Conference, Prospects for the 21<sup>st</sup> Century*, vol. 2, pages 807-812, 1996.
- [14] P. R. Bonasso, D. R. Yoerger and K. W. Stewart, "Semi-Autonomous Vehicles for Shallow Water Mine-Clearing", *Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology AUV'92*, pages 22-28, June 1992.
- [15] A. Balasuriya and T. Ura, "Autonomous Target Tracking by Twin-Burger 2", *Proceedings of the 2000 IEEE/RJS International Conference on Intelligent Robots and Systems IROS'2000*, vol. 2, pages 849-854, 2000.
- [16] T. Curtin, J.G. Bellingham, J. Catipovic and D. Webb, "Autonomous Ocean Sampling Networks", *Oceanography*, vol. 6, no. 3, pages 86-94, 1993.
- [17] P. Lagstad and P. G. Auran, "Real Time Sensor Fusion for Autonomous Underwater Imaging in 3D", *Proceedings of the 1996 MTS/IEEE OCEANS'96 Conference, Prospects for the 21<sup>st</sup> Century*, vol. 3, pages 1330-1335, September 1996.
- [18] J. R. Deschamps, P. Charles and A. Kusterbeck, "Chemical Sensing in the Marine

Environment: Practical Aspects of Sensor Performance and Plume Dynamics”.

- [19] J. S. Willcox, Y. Zhang, J. G. Bellingham and J. Marshall, “ AUV Survey Designed Applied to Oceanic Deep Convection”, Proceedings of the 1996 MTS/IEEE OCEANS'96 Conference, Prospects for the 21<sup>st</sup> Century, vol.2, pages 959-954, 1996.
- [20] J.G. Bellingham and J. S. Willcox, “Optimizing AUV Oceanographic Surveys”, Proceedings of the 1996 Symposium on Underwater Vehicle Technology AUV'96, pages 391-398, 1996.
- [21] H. Kezhong, S. Hainhang, G. Muhe and W. Hong, “Research of Intelligent Mobile Robot Key Techniques”, Proceedings of the 1996 IEEE International Conference on Industrial Technology ICIT'96, pages 503-507, 1996.
- [22] T. S. Chen and R. C. Luo, “Multilevel Multi-Agent Based Team Decision Fusion for Mobile Robots Behavior Control”, Proceedings of the 3<sup>rd</sup> World Congress on Intelligent Control and Automation, vol. 1, pages 489-494, June 2000.
- [23] R. A. Brooks, “Real-Time Vision through Sensor Fission”, Preprints of Japan Artificial Intelligence AI'87, pages 443-446, October 1987.
- [24] A. Zilouchian and M. Jamshidi, “Intelligent Control Systems using Soft Computing Methodologies”, CRC Press, March 2001.
- [25] K. Ganesan, S. M. Smith, K. White and T. Flanigan, “A Pragmatic Software Architecture for UUVs”, Proceedings of the 1996 Symposium on Autonomous Underwater Vehicle Technology AUV'96, pages 209-215, 1996.
- [26] M. N. Desai, M. B. Milton and J. Irza, “Information Augmentation of Passive Tracking for Autonomous Underwater Vehicles”, Proceedings of the 1990 Symposium on Autonomous Underwater Vehicle Technology AUV'90, pages 238-247, June 1990.

- [27] R. C. Luo and T. M. Chen, "Target Tracking by Grey Prediction Theory and Look-Ahead Fuzzy Logic Control", Proceedings of the 1999 IEEE International Conference on Robotics and Automation, vol. 2, pages 1176-1181, 1999.
- [28] D. Iijima, W. Yu, H. Yokoi and Y. Kakazu, "Obstacle Avoidance for a Multi-Agent Linked Robot in the Real World", Proceedings of the 2001 IEEE International Conference on Robotics and Automation, vol. 1, pages 523-528, May 2001.
- [29] G. Grenon, "Enhancement of the Inertial Navigation System for the Florida Atlantic University Autonomous Underwater Vehicles", College of Engineering, Florida Atlantic University, August 2000.
- [30] Advanced Marine Systems (AMS) Laboratory Home Page:  
[www.oe.fau.edu/research/ams.html](http://www.oe.fau.edu/research/ams.html).
- [31] Advanced Marine Systems (AMS) Laboratory Autonomous Underwater Vehicles (AUV) Home Page: [www.oe.fau.edu/AMS/auv.html](http://www.oe.fau.edu/AMS/auv.html).
- [32] S. M. Smith, S. E. Dunn, T. L. Hopkins, K. Heeb and T. Pantelakis, "The application of a Modular AUV to Coastal Oceanography: Case Study on the Ocean Explorer", Proceedings of the 1995 MTS/IEEE OCEANS'95 Conference, Challenges of Our Changing Global Environment, vol. 3, pages 1423-1432, 1995.
- [33] M. Dhanak, E. An, K. Holappa and S. Smith, "Using Small AUV for Oceanographic Measurements", OCEANS'99 MTS/IEEE, Riding the Crest into the 21<sup>st</sup> Century, vol. 3, pages 1410-1417, 1999.
- [34] S. M. Smith, P. E. An, K. Holappa, J. Whitney, A. Burns, K. Nelson, E. Hatzig, O. Kempfe, D. Kronen, T. Pantelakis, E. Henderson, G. Font, R. Dunn, S. E. Dunn, "The Morpheus Ultramodular Autonomous Underwater Vehicle", IEEE Journal of Ocean Engineering, vol. 26, no. 4, October 2001.
- [35] Ocean Explorer Payload Interface Specification,  
[www.oe.fau.edu/AMS/auv\\_payload\\_interface\\_spec.html](http://www.oe.fau.edu/AMS/auv_payload_interface_spec.html).

- [36] An Introduction to VME, [www.lecroy.com/lrs/appnotes/introvme/introvme.htm](http://www.lecroy.com/lrs/appnotes/introvme/introvme.htm).
- [37] Ocean Explorer Electronics Schematics, Electronics Laboratory, Department of Ocean Engineering, Florida Atlantic University.
- [38] Windriver Operating Systems Home Page:  
[www.windriver.com/products/family/os.html](http://www.windriver.com/products/family/os.html).
- [39] PC104 History, [www.pc104.org/history.html](http://www.pc104.org/history.html).
- [40] QNX Operating System – System Architecture, For QNX 4.24, QNX Software Systems Ltd. , 2<sup>nd</sup> edition, October 1997.
- [41] OEX-C Software Manual v2.6, Department of Ocean Engineering, Florida Atlantic University.
- [42] LonWorks Technology Overview - A LonWorks Overview, Echelon Corporation, 1995.
- [43] Introduction to the Lonworks System, v1.0, Echelon Corporation, 1992.
- [44] S. M. Smith, “An Approach to Intelligent Distributed Control for Autonomous Underwater Vehicles”, Proceedings of the 1994 Symposium on Autonomous Underwater Vehicle Technology AUV'94, pages 105-111, July 1994.
- [45] NodeBuilder User's Guide , Revision 3, Echelon Corporation, 1995.
- [46] Neuron C Programmer's Guide, Revision 2, Echelon Corporation, 1992.
- [47] Echelon's LonWorks Products Catalog, Fall 2001 – Spring 2002, Version A, Echelon Corporation.
- [48] LonWorks Microprocessor Interface Program (MIP) User's Guide, Echelon Corporation, Revision 3, 1995.
- [49] IEC Intelligent Technologies Products: PC/104 Adapter,  
[www.ieclon.com/Products/PC104.html](http://www.ieclon.com/Products/PC104.html).

- [50] LonMaker for Windows User's Guide, Echelon Corporation, Version 1.0, 1998.
- [51] G. Grenon, P. E. An, S. M. Smith and A. J. Healey, "Enhancement of the Inertial Navigation System for the Morpheus Autonomous Underwater Vehicle", IEEE Journal of Oceanic Engineering, vol. 26, no. 4, pages 548-560, October 2001.
- [52] "SBE 49 FastCAT Configuration and Calibration Manual", Sea-Bird Electronics Inc. , October 2002.
- [53] D. E. Humphreys, "Development of the Equations of Motion and Transfer Functions for Underwater Vehicles", Naval Coastal Systems Laboratory, July 1976.
- [54] F. Song, "AUV Simulation Manual", Advanced Marine Systems Laboratory, Department of Ocean Engineering, Florida Atlantic University, July 2001.
- [55] Matlab "Fuzzy Logic Toolbox User's Guide", The MathWorks Inc. September 2002.



