

# OPTIMISATION OF DOCKING LOCATIONS FOR REMOTELY OPERATED VEHICLES

by

Trevor John Larkum B.Sc. M.Sc.

A Thesis Submitted for the Degree of

Doctor of Philosophy

in the

Faculty of Engineering

UNIVERSITY OF LONDON

September 2002

Department of Engineering

University College London

Torrington Place

LONDON WC1E 7JE

UMI Number: U592974

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI U592974

Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.  
Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against  
unauthorized copying under Title 17, United States Code.



ProQuest LLC  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

---

## ABSTRACT

---

This thesis describes work aimed at developing practical methods for determining the best docking locations for an underwater remotely operated vehicle (ROV) when inspecting an offshore platform. ROVs are used extensively in the offshore oil and gas industry to conduct a large variety of intervention tasks such as visual inspection, operational monitoring, equipment installation and operation, debris recovery, and so on. However, they have found only limited use in the more difficult tasks such as the detailed inspection of complex weld geometries. These complex welds are, however, found extensively in the construction of the majority of offshore structures and platforms ('oil rigs'). Furthermore, there is a safety requirement to have them inspected regularly since failure of these welds can potentially lead to catastrophic failure of the structures, the majority of which are manned.

A number of specialist ROV systems have been developed that are able to attach onto platform structures and use their manipulators to conduct inspection. However, due to the short reach of the manipulators and the complex geometry of the welds (often encumbered with protruding pipes and other fittings) the success of any inspection is crucially dependent on a good initial choice of ROV docking position. This thesis will describe the problems and current manual planning methods, and then detail the development of two new methods for automated optimisation of docking positions – firstly using neural networks, and secondly using more conventional numerical processing.

This thesis will also review related work in the field, such as the development of neural networks and their applications in the general offshore environment and in the control of ROVs and robot manipulator arms, and other approaches to ROV docking. It will further describe the use of the system developed here for planning docking positions on example commercial ROV inspection work programmes.

---

## **DEDICATION**

---

**In memory of David Roy Broome (1944-1998), late Professor of Automatic Control at  
University College London and Managing Director of General Robotics Limited;  
employer, supervisor, mentor, friend**



---

## **ACKNOWLEDGEMENTS**

---

My thanks go to Dr Alistair Greig, my current supervisor at University College London, and to Dr Martin Hall, of General Robotics Limited, who have both given me invaluable advice and guidance. Thanks are due, too, to my colleagues at GRL and TSC for help with illustrations, and to Dan Clark and his colleagues in Australia for providing underwater photographs of the Covus NICS inspection system in action.

---

## CONTENTS

---

ABSTRACT	2
DEDICATION	3
ACKNOWLEDGEMENTS	4
CONTENTS	5
LIST OF FIGURES	10
LIST OF TABLES	16
LIST OF ACRONYMS	17
FOREWORD	20
Introduction	20
Background	21
Summary of Chapters in the Thesis	21
CHAPTER 1: INTERVENTION BY ROV	23
1.1. Introduction	23
1.2. History and Background	23
1.3. Description	30
1.4. Subsea Inspection	32
1.5. Non-ROV Methods	35
1.6. AROWS	40
1.7. Force Controlled Manipulators	44
1.8. Conclusions	44
CHAPTER 2: ROBOTIC ROV MANIPULATOR SYSTEMS	45
2.1. Introduction	45
2.2. Research and Development	46

2.3. Supervisory Control Systems	50
2.4. REMO	51
2.5. ATEs	54
2.6. ARM	57
2.7. Summary	61
<b>CHAPTER 3: DEVELOPMENT OF NEURAL NETWORK SOFTWARE</b>	<b>62</b>
3.1. Introduction	62
3.2. Neural Network History and Terminology	62
3.3. Implementation	65
3.4. Related Applications	66
3.5. Development of New Software	70
3.6. Neural Networks for Windows (NNW)	72
3.7. NNW in Use	73
<b>CHAPTER 4: TESTING AND VERIFICATION OF NEURAL NETWORK SOFTWARE</b>	<b>75</b>
4.1. Background	75
4.2. Interactive Activation and Competition Network	75
4.3. Constraint Satisfaction Network, Schema Model	85
4.4. Constraint Satisfaction Network, Boltzmann Machine	93
4.5. Constraint Satisfaction Network, Harmony Model	95
4.6. Pattern Associator Network, Hebb Learning Rule	99
4.7. Pattern Associator Network, Delta Learning Rule	104
4.8. Back Propagation Network	106
4.9. NNW Features	112
4.10. Discussion of Deviations	112
4.11. Early Experimentation	116
4.12. Thesis CD-ROM	117
<b>CHAPTER 5: MANUAL PLANNING OF ROV DOCKING</b>	<b>118</b>
5.1. Introduction	118
5.2. Background to Access Simulation	125
5.3. ARM Access Simulation	126
5.4. Development of ARM Docking Planning	127
5.5. Simulation Process	134

<b>CHAPTER 6: DOCKING PLANNING USING NEURAL NETWORKS</b>	<b>137</b>
6.1. Introduction	137
6.2. First Manual Scenario (Coincident Attachment Legs), Schema Model	137
6.3. Second Manual Scenario (Offset Port Attachment Leg), Schema Model	147
6.4. Second Manual Scenario (Offset Port Attachment Leg), Boltzmann Machine	158
6.5. Conclusion	163
<b>CHAPTER 7: DOCKING PLANNING USING NUMERICAL PROCESSING</b>	<b>164</b>
7.1. Introduction	164
7.2. First Automated Scenario (Coincident Attachment Legs)	165
7.3. Second Automated Scenario (Offset Port Attachment Leg)	168
7.4. Third Automated Scenario (ARM Collision Detection and Attachment Leg Features)	170
7.5. Fourth Automated Scenario (Weld Access Check and Deployment System)	173
7.6. Final Development	177
7.7. Time Analysis	180
7.8. Conclusions	183
<b>CHAPTER 8: FIRST USES OF THE AUTOMATED DOCKING PLANNER</b>	<b>185</b>
8.1. Introduction	185
8.2. Docking Planning for Woodside	185
8.3. Docking Planning for Elf	193
8.4. Competing Systems	194
8.5. Conclusions	199
<b>CHAPTER 9: OPERATIONAL USE OF THE AUTOMATED DOCKING PLANNER</b>	<b>200</b>
9.1. Introduction	200
9.2. ROV Support Vessel	201
9.3. Operations	204
9.4. Results	207
9.5. Conclusions	208
<b>CHAPTER 10: CONCLUSIONS</b>	<b>210</b>
10.1. Summary of Results	210
10.2. Neural Network Software	211
10.3. Automated Docking Software	212

10.4. Future Development	214
10.5. Summary	215
REFERENCES	217
APPENDIX A: PUBLICATIONS BY THE AUTHOR	244
APPENDIX B: DETAILED SOFTWARE DEVELOPMENT HISTORY	247
APPENDIX C: NEURAL NETWORK SOFTWARE FEATURES	251
C.1. File Menu	251
C.2. Edit Menu	252
C.3. View Menu	253
C.4. Settings Menu	254
C.5. Patterns Menu	257
C.6. Run Menu	258
C.7. Options Menu	258
C.8. Window Menu	259
C.9. Help Menu	260
C.10. Toolbar	260
C.11. PDP Network Definition Files	261
APPENDIX D: NEURAL NETWORK TEST DEFINITION FILES	262
<u>ROV.tem File</u>	262
<u>ROV.loo File</u>	262
<u>ROV.net File</u>	263
<u>ROV2.str File (Schema Model)</u>	263
<u>ROV2.tem File</u>	264
<u>ROV2.loo File</u>	264
<u>ROV2.net File</u>	265
<u>ROV2.wts File (Boltzmann Machine)</u>	265
APPENDIX E: DOCKING LIBRARY MAIN SOURCE CODE	266
<u>Docking.h</u>	266
<u>Docking.cpp</u>	268
<u>Candidate.h</u>	282
<u>Candidate.cpp</u>	283

<b>APPENDIX F: OFFSHORE OPERATIONS</b>	<b>285</b>
<b>F.1. Mobilisation</b>	<b>285</b>
<b>F.2. Node 4E2, Weld 8</b>	<b>286</b>
<b>F.3. Node 4G2, Weld 5</b>	<b>287</b>
<b>F.4. Node 3C2, Weld 1</b>	<b>288</b>
 <b>APPENDIX G: THESIS CD-ROM</b>	 <b>292</b>
<b>Contents</b>	<b>292</b>

---

## LIST OF FIGURES

---

Figure 1.1 – Divers vs. Submersible and ROV Pilots [from Westwood 1993] .....	23
Figure 1.2 – The most important first generation drill support ROV, the Scorpio [from Given 1991] .....	24
Figure 1.3 – The most important first generation inspection ROV, the RCV 225 [from Bell 1996] .....	24
Figure 1.4 – Triton <i>Diverov</i> Diver Support ROV [from Given 1991] .....	25
Figure 1.5 – An important second generation inspection ROV, the Hyball [from Hydrovision brochure] .....	26
Figure 1.6 – An important second generation drill support ROV, the Diablo [from Hydrovision brochure] .....	27
Figure 1.7 – A typical AUV conducting pipeline survey [from OPL NGUV] .....	27
Figure 1.8 – A typical ROV trencher burying a seabed cable [from OPL NGUV] .....	28
Figure 1.9 – A typical towed plough burying a seabed cable [from OPL NGUV] .....	28
Figure 1.10 – Deployment of an ROV from a platform [from Van Den Hooff 1988] ...	29
Figure 1.11 – Deployment of an ROV from a vessel [from Shirasaki 1988] .....	30
Figure 1.12 – Typical ROV control cabin [from Hattori 1988] .....	31
Figure 1.13 – Typical tubular construction of a jacket [from Van Den Hooff 1988] .....	32
Figure 1.14 – Manipulator mounted ACFM array probe .....	33
Figure 1.15 – Wireline intervention system [from Headworth 1988] .....	34
Figure 1.16 – <i>Wellman</i> Remotely Operated Tool [from Höglund 1988] .....	35
Figure 1.17 – Snorre Remotely Operated Maintenance Vehicle [from Bell 1996] .....	36
Figure 1.18 – Proposed LAMI crawler deployed by a Scorpio [from Evensen 1988] ...	37
Figure 1.19 – Proposed pipe crawler conducting weld inspection [from Hughes 1988]	37
Figure 1.20 – Hardsuit 2000 atmospheric diving suit [from Gibson 2002] .....	38
Figure 1.21 – Sonsub AROWS [from Sonsub IRST] .....	39
Figure 1.22 – Key to Sonsub AROWS components [from Sonsub IRST] .....	40
Figure 1.23 – Diagram of AROWS nodal weld cleaning [from Harman 1988] .....	41
Figure 1.24 – Photograph of AROWS nodal weld cleaning [from Sonsub IRST] .....	42
Figure 1.25 – Photographs of early and late Triton AROWS [from Sonsub IRST and Sonsub Triton ROVS] .....	42

Figure 2.1 – Test arrangement of ASEA robot in pressure chamber [from Aust 1988].	45
Figure 2.2 – Shell Laboratories' ASEA robot and test node [from Van Den Hooff 1988] .....	46
Figure 2.3 – TA9 performing a hybrid position/force task [from Dunnigan 1996] and insertion of subsea mateable connectors [from Lane 1995].	47
Figure 2.4 – Puma robot conducting cylinder location at UCL [from Hughes 1988] ....	47
Figure 2.5 – GE manipulator under PC control at UCL [from Hughes 1988].....	48
Figure 2.6 – ATC Craftsman controlling a Titan II manipulator [from Pegman 1999] .	49
Figure 2.7 – Proposed MARI Advanced Robotic System [from Duncan 1990] .....	51
Figure 2.8 – REMO displaying its various IRM tools [from Stolt Comex Seaway RR]	51
Figure 2.9 – REMO 3D graphical user interface [from Ricci 1996] .....	52
Figure 2.10 – REMO system during ACFM trials [courtesy TSC Ltd].....	53
Figure 2.11 – ATES advanced robotic system [from Sonsub CN3/1].....	54
Figure 2.12 – ATES user interface [from Sonsub ATES RSE and Sonsub CN4/2].....	55
Figure 2.13 – ATES 1 prototype [from Sonsub ATES RSE] .....	55
Figure 2.14 – ARM System carried on an MRV ROV during the NHC trials .....	57
Figure 2.15 – Diagram comparing the working volume of a standard offshore manipulator with that of the ARM System.....	59
Figure 2.16 – ARM Computer System screen showing a Mobil Beryl B node.....	60
Figure 3.1 – Schematic Diagram of a Neuron .....	62
Figure 3.2 – NNW screen during an Interactive Activation and Competition problem .	72
Figure 3.3 – NNW screen during a Constraint Satisfaction problem .....	73
Figure 4.1 – Diagrammatic representation of an IAC Network.....	75
Figure 4.2 – IAC Network retrieving the attributes of an activated node.....	78
Figure 4.3 – IAC Network retrieving a node from a partial description of its attributes	79
Figure 4.4 – IAC Network activating an individual given his attributes correctly .....	79
Figure 4.5 – IAC Network activating an individual given all but one of his attributes correctly .....	80
Figure 4.6 – IAC Network retrieving the correct attributes of an activated node given all information .....	81
Figure 4.7 – IAC Network retrieving the correct attributes of an activated node given partial information .....	81
Figure 4.8 – IAC Network retrieving typical attributes for an activated group node .....	82
Figure 4.9 – Comparison of activation values changing over time for PDP versus NNW .....	83
Figure 4.10 – Diagrammatic representation of a CS Network.....	85
Figure 4.11 – CS Network settling on a right-hand Necker Cube interpretation.....	88
Figure 4.12 – CS Network, Schema Model, after activation of bathtub input.....	91
Figure 4.13 – CS Network, Harmony Model, initial state of electricity problem .....	96



Figure 4.14 – CS Network, Harmony Model, final state of electricity problem .....	97
Figure 4.15 – Diagrammatic representation of a PA Network .....	98
Figure 4.16 – PA Network, Hebb Learning, network state before training .....	101
Figure 4.17 – PA Network, Hebb Learning, network state after training .....	101
Figure 4.18 – PA Network, Hebb Learning, network state after testing with a new pattern .....	102
Figure 4.19 – PA Network, Hebb Learning, network state after three epochs of training .....	103
Figure 4.20 – PA Network, Delta Learning, network state after 3 epochs of training ..	104
Figure 4.21 – PA Network, Delta Learning, network state after 100 epochs of training, despite noise .....	104
Figure 4.22 – Diagrammatic representation of a BP Network .....	106
Figure 4.23 – BP Network, XOR Problem, before training .....	109
Figure 4.24 – BP Network, XOR Problem, testing results after training with 300 epochs .....	110
Figure 4.25 – Comparison of activation values changing over time for PDP versus NNW using <i>double</i> and <i>float</i> floating point representations .....	114
Figure 4.26 – NNW noughts and crosses network .....	115
Figure 5.1 – Docking onto a proprietary template [from Renard 1988] .....	120
Figure 5.2 – Docking onto standard template [after Vinsen 1988] .....	121
Figure 5.3 – SWIMMER AUV (orange buoyancy) with ROV (yellow buoyancy) [from Chardard 2002] .....	121
Figure 5.4 – ARM attachment leg .....	123
Figure 5.5 – ARM Software being used for Access Simulation .....	126
Figure 5.6 – Tartan Alpha: the riser of interest and its mounting brace are directly in front of the ROV .....	127
Figure 5.7 – Access checking on an almost hidden weld (highlighted ahead of the ROV) .....	128
Figure 5.8 – Proposed RACAL toolskid and manipulator design .....	128
Figure 5.9 – RACAL system inspecting the underside (6 o'clock) on a nodal brace ...	129
Figure 5.10 – RACAL system manufactured by Trittech [courtesy TSC Ltd] .....	129
Figure 5.11 – The ARM System on node 3A2, inspecting the 12 o'clock position .....	130
Figure 5.12 – The ARM System on the inside of 6A3, inspecting the 6 o'clock position .....	130
Figure 5.13 – ARM Inspecting an internal nodal weld from outside the platform .....	132
Figure 5.14 – Seal ROV inspecting a weld from the edge of the conductor guide frame .....	132
Figure 6.1 – Illustration of the grid of candidate positions used .....	138
Figure 6.2 – Illustration of the First Scenario (coincident single leg and manipulator) ..	138

Figure 6.3 – Determining a score value from manipulator or attachment leg extension .....	140
Figure 6.4 – Results of ROV Docking Test after 50 cycles.....	145
Figure 6.5 – Results after 100 cycles .....	145
Figure 6.6 – Results after 150 cycles .....	145
Figure 6.7 – Illustration of the Second Scenario (offset port attachment leg) .....	147
Figure 6.8 – Results of second Manual ROV Docking Test after 30 cycles .....	153
Figure 6.9 – A Plot of Goodness Against Number of Cycles for the Second Manual Scenario, Schema Model .....	155
Figure 6.10 – A Plot of Goodness Against Number of Cycles for the Second Manual Scenario, Boltzmann Model, Annealing over 250 cycles .....	159
Figure 6.11 – A Plot of Goodness Against Number of Cycles for the Second Manual Scenario, Boltzmann Model, Annealing over 500 cycles .....	161
Figure 7.1 – Creation of candidate positions (-1 to +1 in X, Y and Z, 1m grid spacing) .....	166
Figure 7.2 – Elimination of candidate positions in collision with workpiece.....	166
Figure 7.3 – Increased grid density from automated method .....	168
Figure 7.4 – Checking a position for sticky feet attachment .....	170
Figure 7.5 – Checking a position for ROV collision .....	171
Figure 7.6 – Checking a position for manipulator kinematic access .....	175
Figure 7.7 – Accessing the far side of a brace using the manipulator deployment system .....	176
Figure 7.8 – Docking Settings dialog box.....	178
Figure 7.9 – Docking Results dialog box.....	179
Figure 7.10 – Timing Results for Docking Optimisation Phases .....	181
Figure 8.1 – Location of North Rankin Alpha platform [from Batten 1988] .....	185
Figure 8.2 – Woodside MOC-1 inspection and cleaning toolskid [from Batten 1988] .....	186
Figure 8.3 – Original NICS design (note toolskid hinge/tilt function) .....	186
Figure 8.4 – Covus ARM System inspecting 10 to 10.30 on Woodside weld 9 .....	188
Figure 8.5 – Covus ARM System on the weld 9 node, showing automated docking positions considered .....	190
Figure 8.6 – The Covus NICS toolskid.....	190
Figure 8.7 – The camera/probe mounting with touch switch and 45° mirror at left.....	192
Figure 8.8 – ARM model of the node 16A5 (welds to be accessed are marked thus*) .....	193
Figure 8.9 – RovTech system inspecting 9 o'clock on weld 22A9 .....	193
Figure 8.10 – Sonsub system inspecting 4 o'clock on weld 16A13; the manipulator is rolled over to 125° .....	194
Figure 8.11 – SSOL system inspecting 6 o'clock on weld 16A13; manipulator is mounted upside down.....	194

Figure 8.12 – DSND system inspecting the top and underside of weld 16A13 .....	195
Figure 9.1 – FATs in England of ARM Software controlling a Titan 3 manipulator...	199
Figure 9.2 – Factory Tests in Australia of ARM Software controlling a Titan 2 manipulator on the NICS skid .....	200
Figure 9.3 – Shelf Supporter ROV support vessel, forward and aft views.....	201
Figure 9.4 – Shelf Supporter ROV support vessel [from Batten 1988].....	201
Figure 9.5 – The ROV station on Shelf Supporter, with the launch system (orange) folded over the Triton ROV (yellow) .....	202
Figure 9.6 – ROV control room with ARM laptop computer in the foreground.....	203
Figure 9.7 – East face of North Rankin Alpha showing inspected nodes [courtesy T. Heale] .....	204
Figure 9.8 – Triton ROV mounting the NICS skid being deployed beside North Rankin .....	205
Figure 9.9 – Pan and tilt camera view showing inspection of 5.00 position on node 4G2 (manipulator is rolled over to 135°) .....	205
Figure 9.10 – Equivalent ARM view (weld from 2.30 to 5.00 is highlighted).....	206
Figure 10.1 – Relationship between thesis work and existing ARM software.....	213
Figure B.1 – WinNeural Version 0.13 running on Windows 3.1 .....	247
Figure B.2 – WinNeural Version 0.16 running on Windows 95 .....	248
Figure B.3 – NNW Version 1.0 running on Windows 98.....	249
Figure B.4 – NNW Version 1.1 running on Windows 2000.....	249
Figure C.1 – File New / New Neural Net dialog box .....	251
Figure C.2 – Edit Input Values / Set Input dialog box.....	251
Figure C.3 – Edit Neuron Weights / Set Weight Value dialog box.....	252
Figure C.4 – Run Results dialog box .....	252
Figure C.5 – System Settings dialog box.....	253
Figure C.6 – Display Settings dialog box .....	254
Figure C.7 – Strength Parameters dialog box .....	254
Figure C.8 – Activation Parameters dialog box.....	255
Figure C.9 – Rate and Other Parameters dialog box.....	255
Figure C.10 – Enter Pattern dialog box.....	256
Figure C.11 – Help / About NNW dialog box .....	259
Figure F.1 – NICS on deck, fitted with twin HP water jets, awaiting deployment .....	284
Figure F.2 – ARM NICS System deploying into the water .....	285
Figure F.3 – ARM view of node 4E2, 2.30 – 6.00 on weld highlighted .....	286
Figure F.4 – Inspection of 5.30 position on node 4G2 using right-angled probe mounting.....	287
Figure F.5 – ARM NICS System showing the underside claw and ROV in rear position .....	288

Figure F.6 – Inspection of 11 o'clock position on node 3C2 .....	288
Figure F.7 – ARM view of inspection of 11 o'clock position.....	289
Figure F.8 – Inspection of interstitial weld on node 3C2.....	290

---

## LIST OF TABLES

---

Table 3.1 – Table of Neural Network Types .....	71
Table 4.1 – Comparison of Necker Cube results, input strength ( <i>istr</i> ) = 0.4 .....	90
Table 4.2 – Comparison of Necker Cube results, input strength ( <i>istr</i> ) = 0.01 .....	91
Table 4.3 – NNW results showing activated units for given activated inputs .....	92
Table 4.4 – Comparison of Necker Cube results, annealing over 20 cycles .....	94
Table 4.5 – Comparison of Necker Cube results, annealing over 400 cycles .....	95
Table 6.1 – Scoring docking positions manually, first scenario .....	142
Table 6.2 – Network strengths and unit names in <code>Rov.str</code> file .....	144
Table 6.3 – Network weights in <code>Rov.wts</code> file .....	145
Table 6.4 – Scoring docking positions manually, second scenario .....	149
Table 6.5 – Network weights in <code>Rov2.wts</code> file .....	151
Table 6.6 – Results from 50 runs of the second scenario network definition .....	153
Table 6.7 – Summary of results from 100 runs using the Schema Model .....	154
Table 6.8 – <code>Rov.str</code> file with Boltzmann mode on and annealing schedule defined ..	158
Table 6.9 – Summary of results from 100 runs using the Schema Model .....	159
Table 6.10 – Summary of results from 100 runs using the Boltzmann Machine .....	161
Table 7.1 – Timing Results for different optimisation phases .....	180
Table 9.1 – Summary of Planned versus Achieved weld access .....	208
Table B.1 – Milestones in the Development of WinNeural .....	248
Table B.2 – Milestones in the Development of Neural Networks for Windows .....	250

---

## LIST OF ACRONYMS

---

3D	3 Dimensional
AAD	Articulated Attachment Device
ACFM	Alternating Current Field Measurement (crack detection probe)
ACG	Automatic Control Group
ACPD	Alternating Current Potential Drop (an NDT method for detecting surface cracks in a metal)
ADS	Atmospheric Diving Suit
ALIVE	Autonomous Light Intervention VEHICLE
AMBNNC	Adaptive Model-Based Neural Network Controller
ANN	Artificial Neural Network
ANNC	Artificial Neural Network Controller
AROWS	Advanced Remotely Operated Work System
ARM	Advanced Robotic Manipulator
ATES	Advanced TElemanipulation System
AUSS	Advanced Unmanned Search System
AUV	Autonomous Underwater Vehicle
BP	Back Propagation (neural network)
CAT	Computer-Aided Telemanipulation
CLEROV	Cleaning ROV
CMAC	Cerebellar Model Articulation Controller (neural network)
CP	Cathodic Potential (a measure of the potential for corrosion of a metallic structure)
CS	Constraint Satisfaction (neural network)
CURV	Cable controlled Underwater Research Vehicle
DOF	Degrees Of Freedom
DP	Dynamic Positioning (a computerised position system used by support vessels)
DSV	Dive Support Vessel
FMD	Flooded Member Detection

GE	General Electric
HP	High Pressure
HPR	Hydroacoustic Position Reference
IAC	Interactive Activation and Competition (neural network)
IMCA	International Marine Contractors Association
IRM (IMR)	Inspection, Repair and Maintenance (Inspection, Maintenance and Repair)
ITT	Invitation To Tender
JCV	Jacket Cleaning Vehicle
LARS	Launch and Recovery System
LMS	Least Mean Square associator (neural network)
MIMIC	Modular Integrated Man-machine Interaction and Control
MIT	Massachusetts Institute of Technology
MOC	Modular Offshore Cleaner
MPI	Magnetic Particle Inspection (an NDT method used to locate cracks at or near the surface of magnetic materials)
MRU	Motion Reference Unit
MRV	Multi-Role Vehicle
NARRC	National Advanced Robotics Research Centre
NDN	Neural Dynamic Network
NDT	Non-Destructive Testing
NKN	Neural Kinematic Network
NNW	Neural Network for Windows (thesis software)
NICS	Nodal Inspection and Cleaning System
NRA	North Rankin Alpha (oil platform)
NTO	Neural Trajectory Optimisation
OED	Optimal Experiment Design
PA	Pattern Associator (neural network)
PC	Personal Computer
PDP	Parallel Distributed Processing
PID	Proportional/Integral/Derivative control system
PSS	Pattern Sum of Squares
RBF	Radial Basis Function
ROMV	Remotely Operated Maintenance Vehicle
ROV	Remotely Operated Vehicle
ROT	Remotely Operated Tool
ROTV	Remotely Operated Television

<b>SCS</b>	<b>Stolt Comex Seaway</b>
<b>SHARPS</b>	<b>Sonic High Accuracy Ranging and Positioning System</b>
<b>SSOL</b>	<b>Subsea Offshore Limited</b>
<b>SWIMMER</b>	<b>Subsea Works Inspection and Maintenance with Minimum Environment ROV</b>
<b>TMS</b>	<b>Tether Management System</b>
<b>TSC</b>	<b>Technical Software Consultants Limited</b>
<b>TSS</b>	<b>Total Sum of Squares</b>
<b>TWI</b>	<b>The Welding Institute</b>
<b>UCL</b>	<b>University College London</b>
<b>UUV</b>	<b>Unmanned Underwater Vehicle</b>
<b>WAM</b>	<b>Whole-Arm Manipulator</b>
<b>XOR</b>	<b>Exclusive-Or</b>



### Introduction

A number of initiatives and projects have looked at the problems of complex weld inspection by ROV, and have produced systems that have achieved varying degrees of success at the task. These systems typically consist of ROVs mounting one or more advanced manipulator arms plus a number of attachment legs for fixing to the structure (these systems will be reviewed in this thesis). The overall systems frequently also incorporate some form of advanced computer control system with a 3D graphical model of the worksite and environment.

However, they all have one particular problem in common, in that the success of a particular inspection operation is crucially dependent on the initial choice of location for the ROV system to dock onto the worksite (due in part to the complex geometry of the worksite, and hence the limited access for the ROV, and in part to the limited reach of the manipulator arms). Since the choice of docking location needs to consider multiple – often conflicting – constraints on the system it is generally impossible for a human operator to select the best position without aid, instead relying on time-consuming trial and error.

This thesis concentrates on this problem of optimising the docking location for an ROV and examines and compares three techniques for selecting the best docking position given multiple constraints:

1. Manual selection, which is often accompanied by iterative further guesses at a best location.
2. Automated selection using an artificial neural network to make a ‘best guess’ selection.
3. Automated selection using the computer control system to generate a large number of possible locations and then eliminate all positions that violate

constraints, such as manipulator reach, unwanted collisions and insufficient attachment leg positions.

## **Background**

The work for this thesis was begun with registration for a part-time PhD in October 1992, just over a year after the author joined the Control and Robotics Group of Technical Software Consultants (TSC) Limited as a Robotics Engineer working on the Automated Robotic Manipulator (ARM) Project. This was under the supervision of Dr, later Professor, David Broome, then Reader in Automatic Control in the Mechanical Engineering Department of University College London, and also head of the Control and Robotics Group, and a Director, at TSC.

The work conducted in 1992 and early 1993 was largely research into the field of neural networks, as well as a more wide-ranging literature survey. The development of new neural network software was begun in May 1993 with Interactive Activation and Competition networks completed in 1993, Constraint Satisfaction and Pattern Associator networks in 1994, and Back Propagation networks in 1995 (although small improvements continued thereafter). The Control and Robotics Group separated from TSC Ltd in March 1996 to form General Robotics Limited (GRL) with David Broome as Managing Director and the author as General Manager.

The work on using neural networks to select a docking location was conducted in 1996 and early 1997. The work on using the automated numerical pre-processing to optimise the docking location took place from February 1997 to March 1998. Following David Broome's death in April 1998, after which the author took on the role of GRL Managing Director, there was a hiatus. Work continued again from February 1999, concentrating on the automated optimisation system outputting neural network files in a suitable format to be read in and solved by the neural network software. Commercial access simulation work using the automated optimisation system was conducted in 1999 and early 2000, and offshore operational work using it took place during May and September 2000. Revisiting of earlier work, writing up and a further literature survey were conducted up until completion in the summer of 2002.

## **Summary of Chapters in the Thesis**

Chapter 1 describes underwater intervention, and particularly inspection, by ROV and compares this with other methods. Chapter 2 looks more specifically at the robotic

manipulator solutions developed to improve access to welds and to carry out NDT inspection of complex weld shapes.

Chapter 3 describes the history and development of neural networks, and looks at their application to the control of robotic manipulators, and to general offshore and oceanographic use. Chapter 4 describes the author's development of software to model the four main types of neural network (Interactive Activation and Competition, Constraint Satisfaction, Pattern Association and Back Propagation), and Chapter 5 details the background theory, testing and verification of the software.

Chapter 6 describes the main methods for docking ROVs onto underwater structures, and looks at manual ways of planning docking locations. Chapter 7 looks at using neural networks to select docking locations, while Chapter 8 looks at an automated method of optimising docking locations using a numerical software method (and concludes that this is the best of the three methods).

Chapter 9 describes the use of the automated docking optimisation system on two ROV access simulation tasks. Chapter 11 describes an operational offshore ROV inspection programme that made extensive use of the automated optimisation system; Chapter 12 covers conclusions and possible future improvements.

The offshore inspection programme covered in Chapter 11 was a commercial contract, as were the access simulations described in Chapters 9 and 10, of which the automated optimisation work was just one, albeit highly significant, element. With these exceptions, all the work described in this thesis was done as PhD research, and exclusively by the author.

A selected list of publications by the author is given at Appendix A. A CD-ROM containing the source files for the neural network software (some 300 files), plus data files, is attached – for more details see Appendix G.

This thesis is 69,000 words long.

A handwritten signature in black ink, appearing to read 'Trevor Larkum', with a long horizontal line extending from the end of the signature.

Trevor Larkum

---

## CHAPTER 1: INTERVENTION BY ROV

---

### 1.1. Introduction

For many years, all undersea work was performed by human divers. The last few decades, however, have seen the emergence of the Remotely Operated Vehicle or ROV. The ROV is controlled from the surface, but can dive down to undersea work sites to carry out work previously done by divers. This has produced great improvements in safety, and cost effectiveness.

ROVs were originally developed for military purposes, but rapidly began to be used more for civilian purposes in support of the offshore oil and gas industries, particularly in the North Sea during the late 1970s and early 1980s [Marsh, R. 1996]. In fact, without the driving force of the offshore industry there would have been no ROV industry as no other outlets – even defence, scientific, inshore or nuclear put together – provide a sufficient market to make it self-supporting [Hayward 1991].

### 1.2. History and Background

As early as 1953 in the USA, the development of a diver propulsion vehicle by a company called Rebikoff produced an ROV called Poodle which was used to locate shipwrecks [Bell 1996]. ROVs first showed their worth in 1966 when the US Navy used CURV (Cable controlled Underwater Research Vehicle) to recover a lost nuclear bomb off Palomares, on the Costa del Sol, Spain. Although primitive by modern standards, CURV was able to grapple the bomb's parachute shrouds at a depth of 860m and bring it safely back to the surface. It was again used in a vital operation in 1973 after the manned submersible Pisces III sank in 475m of water off Cork, Ireland. CURV attached a lifting line to the vehicle and allowed a successful recovery [Last 1991].

During the 1980s there was an increasing use of ROVs for mine detection and disposal purposes. These ROVs normally employed a short-range sonar set in the nose, frequently backed up by a TV camera for identification. Having identified the mine, it

could either be cut loose or detonated on the seabed by a charge [Blake 1989]. However, as the larger commercial ROV market developed, independent mine-specific ROV designs were eventually replaced by adaptations of commercial designs.

The first fields in the northern North Sea, Hamilton Argyle and BP Forties, came on-stream in 1974, when records show that 700 divers were at work in the North Sea. Numbers peaked at 1,400 in 1985 then began a decline (e.g. back to mid-'70s levels in the early 1990s). There were a number of factors responsible for the decline in divers, in particular the greatly improved efficiency of vessels, equipment and operational techniques, and the move by the oil companies towards designing out the need for divers, as well as a major change in technology [Westwood 1993]. Manned submersibles were also widely used, operating in either intervention or lock-out diver mode. In intervention mode they were used for inspection, survey and general manipulation tasks (as ROVs were to do later), and in diver lock-out mode they were used as a means of carrying and supporting divers in saturation [Bell 1996].

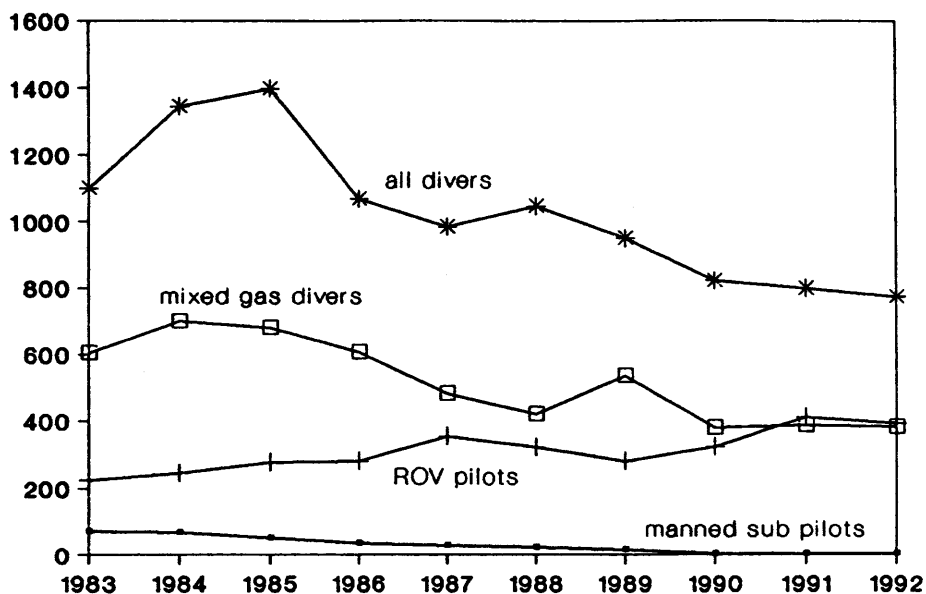


Figure 1.1 – Divers vs. Submersible and ROV Pilots [from Westwood 1993]

ROVs were introduced into commercial North Sea operations in the late 1970s and met with a hostile reaction from the diving companies and operators of manned submersibles. By the mid 1980's the manned submersible operators were driven out of business by dynamically positioned (DP) dive support vessels (DSVs) and pipeline inspection ROVs [Westwood 1993]. As the numbers of divers and submersible pilots declined so, inexorably, the number of ROV pilots increased (see Figure 1.1).

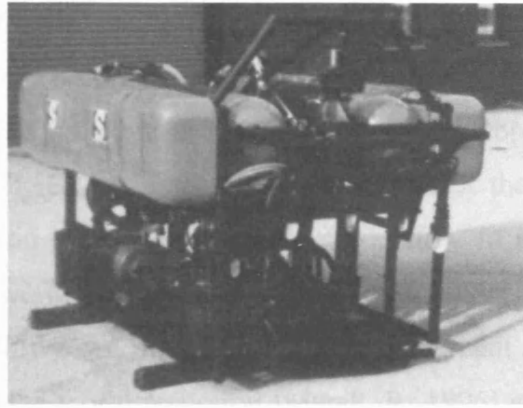


Figure 1.2 – The most important first generation drill support ROV, the Scorpio [from Given 1991]

Commercially, the early ROVs were used in cable survey and recovery. They began to be used regularly in offshore operations in the late 1970s and have now become "indispensable" [Last 1991]. There were just 3 ROVs in commercial operation in 1976, but this rose to 300 in 1986 and to almost 2500 in the early 1990s [Westwood 1993]. The most significant first generation models were the RCV 225 inspection ROV and the AMETEK (later Perry Triton Inc.) Scorpio used for drill support (see Figure 1.2) [Marsh, R. 1991; Westwood 1993; Pedlow 1996]. As discussed above, the origins of both of these early vehicles were military. The Scorpio had US Navy backing and was initially intended as a mine recovery vehicle while the RCV 225 (see Figure 1.3) was designed to be 21 inches in diameter so that it could be deployed through a torpedo tube [Bell 1996].

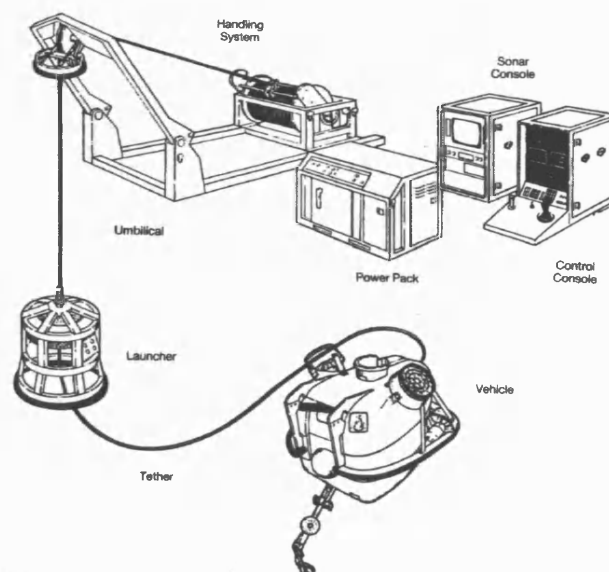


Figure 1.3 – The most important first generation inspection ROV, the RCV 225 [from Bell 1996]

By the late 1990s ROVs were commonplace in many markets, including oceanography, fishing, civil engineering, security, mineral prospecting as well as several other niche areas [Marsh, R. 1996]. This led to fast and significant development in the designs and technology used, which led to increased reliability, such that the ROV became an invaluable tool in oil and gas exploration and production. In the early 1980s building ROVs was a business worth about \$55million [Hayward 1991]. During the mid 1980s, however, there was a slowdown in technological development due in part to the slump in the price of oil and the world recession [Marsh, R. 1996] and the size of the ROV building industry halved. It was expected that second generation ROVs would come into service to replace the excellent but ageing RCV-225s and Scorpions but due to the collapse in the price of oil in 1986 this did not happen [Marsh, R. 1991]. Where previously the vision was total diver replacement, instead ROVs were largely used in conjunction with divers – largely due to insufficient investment in ROV development and technology [Hayward 1991]. This even led to the rather incongruous situation that pervaded for a short time of ROVs being designed and built to support divers, for example the Triton Diverov (see Figure 1.4) operated by Stolt-Nielsen Seaway [Given 1991].

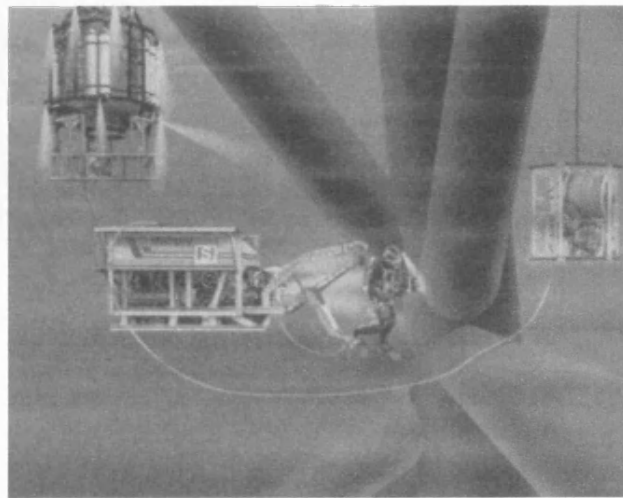


Figure 1.4 – Triton *Diverov* Diver Support ROV [from Given 1991]

Nonetheless even by the late 1980s and early 1990s it was clear that ROVs could be used much more to replace divers, providing improvements in safety and cost: “it is likely that the current oil/gas economic climate will be the catalyst required to stimulate a much broader interest in and acceptance of underwater remote technology... There is little doubt that much work currently being carried out by divers can be done more cost effectively utilizing remote technology...” [Batten 1988]. This was particularly evident

if the through-life cost of a structure was considered since the increased cost at installation of providing ROV-friendly controls would be more than repaid in lower costs during the production and eventual decommissioning phases [Marsh, R. 1991; Westwood 1993]. Furthermore, by this time, in Shell Expro for example, there were several tasks that were standardised as diverless including structural cleaning and inspection, and pipeline survey and stabilisation [Marsh, T. 1992].

It was also clear that further improvements could be made by applying new technology to the ROV systems so that it was predicted that by the end of the 1990's "there should be no technical or economic reason why almost every subsea task is not being done either robotically or by remote control" [Marsh, R. 1991].

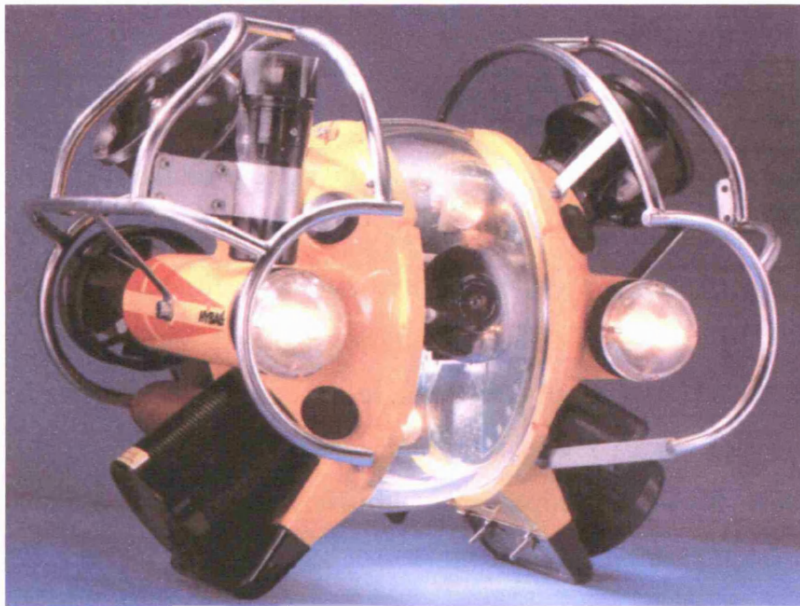


Figure 1.5 – An important second generation inspection ROV, the Hyball [from Hydrovision brochure]

By the middle of the 1990s there had been some further developments. The first generation ROVs were now being replaced by vehicles that both had a higher performance and were significantly less expensive, such as the Hydrovision Diablo and Hyball (see Figures 1.5 and 1.6) systems [Pedlow 1996; Bell 1996]. ROVs were also being used more and more as intervention tools carrying out tasks traditionally undertaken by divers. More advanced tooling skids were being developed to undertake more difficult tasks and ROVs were more capable of manipulative tasks such as valve operation, subsea assembly and salvage work [Pedlow 1996].



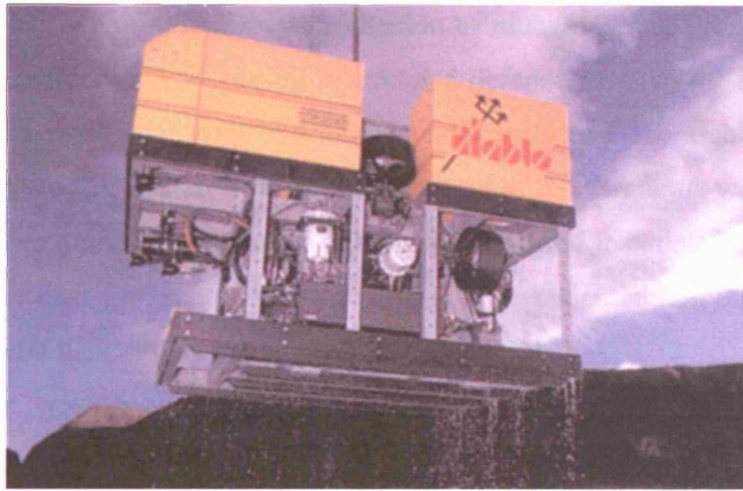


Figure 1.6 – An important second generation drill support ROV, the Diablo [from Hydrovision brochure]

Concurrently there had been developments in the Autonomous Underwater Vehicle (AUV), essentially an untethered ROV using batteries for motive power and following pre-programmed instructions (see Figure 1.7). These vehicles are also often known as Underwater Unmanned Vehicles (UUV) although, technically, a UUV need not be autonomous. One of the earliest AUVs was the US Navy's Advanced Unmanned Search System (AUSS) originally developed from the mid-1970s, a torpedo shaped vehicle with a 10 hour endurance at a maximum speed of 5 knots [Westwood 1993]. Despite two major problems – the difficulty of carrying a power supply to give an adequate range and endurance, and the bandwidth for adequate real-time control and data communication – AUVs started to see increasing commercial use during the 1990's for offshore pipeline survey and similar tasks.

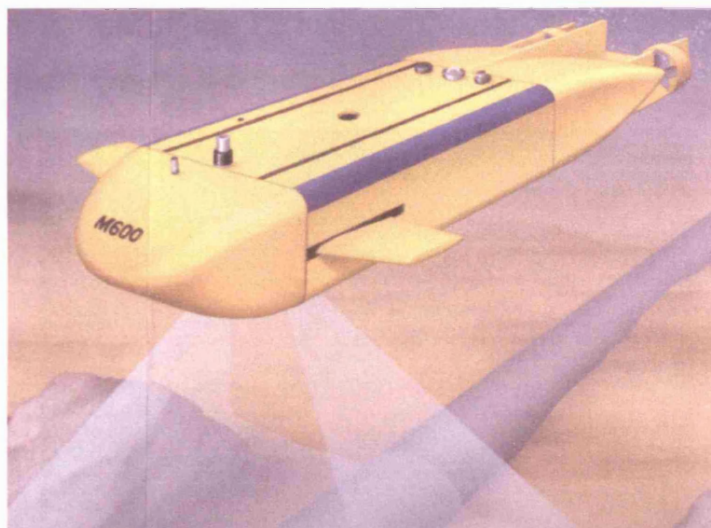


Figure 1.7 – A typical AUV conducting pipeline survey [from OPL NGUV]

The turn of the millennium saw the introduction of much new ROV hardware, and the increasing diversification from the oil related and defence roots of the subsea industry. Oceanographers, civil engineers, marine archaeologists and the fishing industry all began to view the ROV or the AUV as a routine tool of their trade. A welcome effect of the Internet is the tremendous amount of subsea fibre optic installation planned around the world in the next few years, the source of an unprecedented amount of activity for ROVs, AUVs and their support vessels [Marsh, R. 2000].

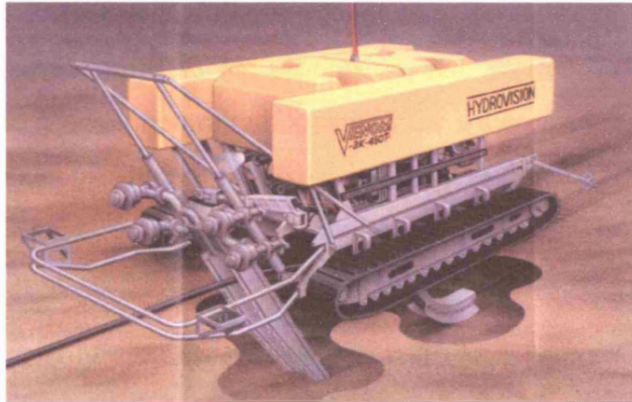


Figure 1.8 – A typical ROV trencher burying a seabed cable [from OPL NGUV]

In the early 21<sup>st</sup> Century, ROV development progress, as a mature technology, is steady rather than spectacular. There are more capable and powerful vehicles with better payloads, and a move back to electric vehicles of all sizes, which is where the industry began a quarter of a century before [Marsh, R. 2002]. More significant is that the increasing move to oil exploration in deeper waters has eliminated the ROV versus diver debate – only ROVs can operate at the required depths (600-3000m). Other related systems, such as ploughs and trenchers (see Figures 1.8 and 1.9), proliferate due to the inevitable global growth of the IT industry [Marsh, R. 2002] – however, further consideration of these systems is beyond the scope of this work.

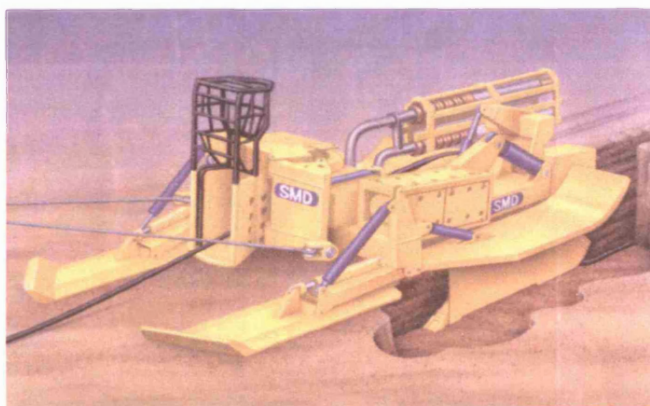


Figure 1.9 – A typical towed plough burying a seabed cable [from OPL NGUV]

The late 1990s and early 2000s saw the early development of new types of vehicles, the 'Hybrid Vehicle' and the 'Work AUV', having some of the attributes of the ROV and some of the AUV [Chardard 2002]. Although these systems are just at the prototype stage, they incorporate some interesting docking functions and will be covered in more detail in *Section 5.1.2. Docking Using Pre-Defined Attachment Points*.

### 1.3. Description

A typical workclass ROV (see, for example, Figure 1.6) consists of an aluminium frame which mounts six-seven thrusters for propulsion, and carries on top large blocks of buoyancy in order to make it neutrally buoyant in the water. The frame contains control electronics and an electric pump which is powered by an umbilical cable from the surface and provides high-pressure hydraulic flow to power the thrusters, any manipulators at the front of the vehicle, and any other hydraulic equipment onboard.

The manipulators are controlled by an operator at the surface via signals sent down the umbilical – a process known as 'teleoperation' or 'telemanipulation'. The ROV also carries many sensors including an obstacle avoidance sonar (and possibly sidescan and profiling sonars), a depth gauge, an altimeter, a gyro compass, a high-resolution zoom colour camera on a pan and tilt mounting, a low light level Silicon Intensified Target (SIT) black and white camera, and possibly a number of other cameras (plus high intensity lights).

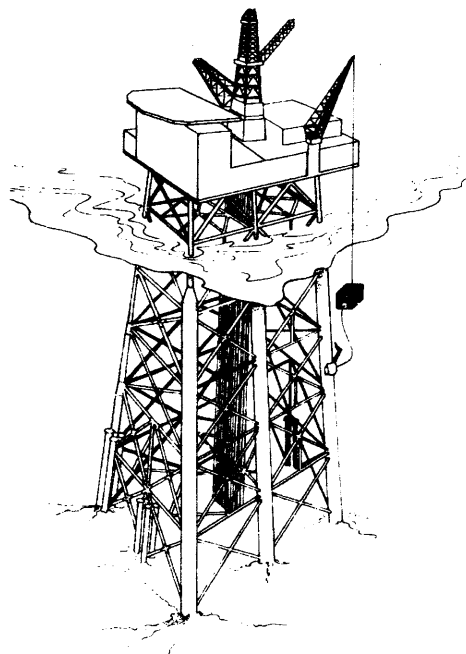


Figure 1.10 – Deployment of an ROV from a platform [from Van Den Hooff 1988]

The ROV is deployed from a platform (see Figure 1.10) or a vessel (see Figure 1.11). Operating from a structure is not as weather dependent as operating from a vessel; for safety reasons, a vessel can only work where the wind would blow it clear of the platform in the event of a power failure. Vessel operations do, however, have an advantage in that the vehicle does not need to pull the umbilical through or round the complex structure (or 'jacket'), which reduces the possibility of umbilical entanglement [Last 1991]. For deep operations the umbilical will typically not go direct to the ROV but instead to a Tether Management System (TMS), an intermediate winch held in the water at the ROV's operating depth, and a smaller tether umbilical then connects the TMS to the ROV.

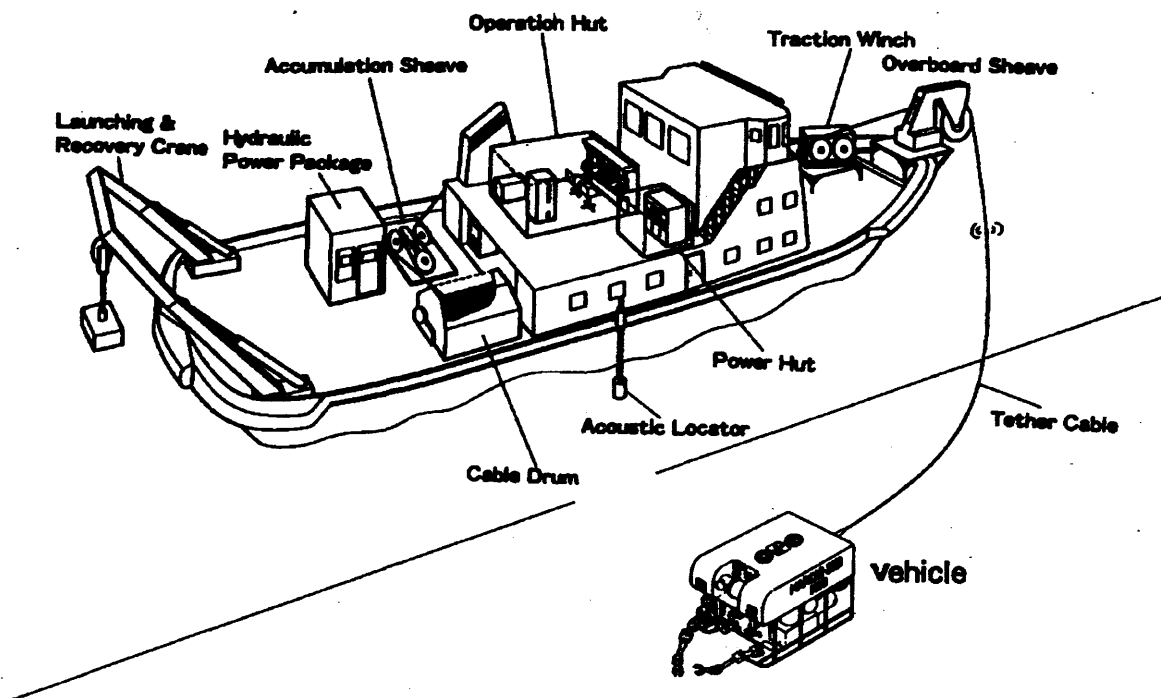


Figure 1.11 – Deployment of an ROV from a vessel [from Shirasaki 1988]

The ROV is controlled from a cabin, usually a specially equipped container, on the platform or vessel. This typically has at least two control stations, one for the ROV pilot and one for the manipulator operator (see Figure 1.12). The ROV is launched and recovered over the side using a Launch And Recovery System (LARS), usually either an A-frame or a knuckle-boom ('Hiab' type) crane.

The International Marine Contractors Association (IMCA) classifies ROVs into five main types: (1) Pure observation class; (2) Observation with payload option; (3) Workclass systems; (4) Towed or bottom-crawling vehicles; and (5) Prototype or

development vehicles. With the exception of certain observation vehicles proposed as deployment systems for inspection probes, generally only Class 3 vehicles, workclass ROVs, are capable of conducting nodal weld inspection and will be the focus of this work.

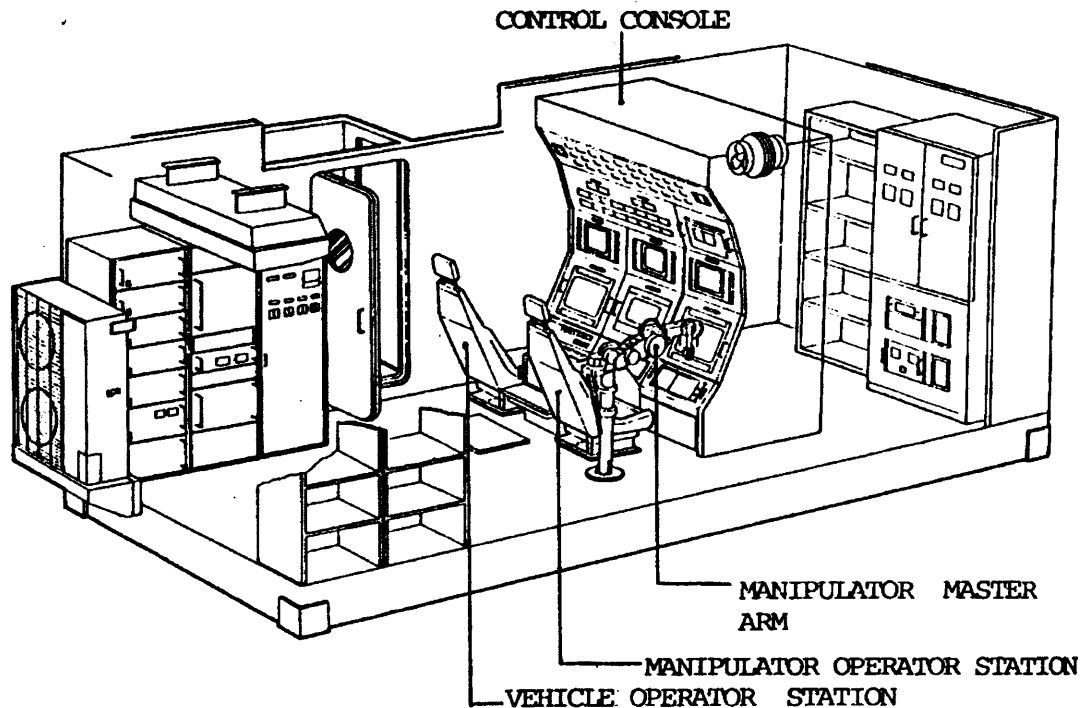


Figure 1.12 – Typical ROV control cabin [from Hattori 1988]

#### **1.4. Subsea Inspection**

Structural inspections are carried out to provide information on the condition of the structure and its features. The information is required for certification and maintenance purposes. Offshore installations are subject to constant stress, from the static loading on their decks, the dynamic loading of wind and sea and the effects of corrosion and accidental damage. In the early days of North Sea exploration there were a number of serious structural failures on rigs and platforms. Such failures can be catastrophic, both in human and in economic terms. Regular inspection is designed to identify early signs of failure and allow the operators to take remedial action.

Each country enforces inspection requirements, for example in the UK the Offshore Installations (Construction and Survey) Regulations 1974 require each installation to have a Certificate of Fitness, valid for five years [Last 1991]. To obtain a new certificate, the installation must undergo a major survey, or a series of annual surveys,

during the five year period of its current certificate. The certificate must be issued by an approved body, such as Lloyds, Det Norske Veritas or the American Bureau of Shipping. The detailed requirements for each survey are agreed with the certifying authority. They are different for each installation and depend upon the type of structure and the results of previous inspections.

A typical survey includes [Last 1991]:

1. Inspection of a representative number of welds.
2. A corrosion survey, which includes cathodic potential (CP) readings and a survey of the protection system.
3. A full survey of risers, conductors, caissons, etc., and their protection systems. .
4. A survey of the seabed. This includes a check for scouring, and the accumulation of debris. Debris may cause damage directly, by impact, or by increasing corrosion.
5. A survey of any physical damage.
6. A marine growth survey. Marine growth adds significantly to the static loading on the structure, increases tidal or current drag, blocks inlets and outlets, and may cause corrosion.

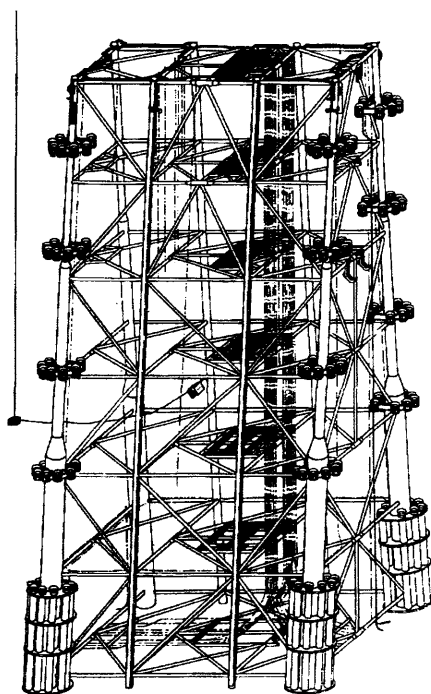


Figure 1.13 – Typical tubular construction of a jacket [from Van Den Hooff 1988]



Fatigue is the commonest cause of cracking in steel structures. It occurs when a structural member is subject to alternating or cyclic loads over a prolonged period. The greater the peak loading, and the greater the frequency of oscillation, the sooner the member will fail. Cracks normally occur at areas of stress concentration, for example welds or areas of damage, and the effects are accelerated by corrosion. The most important areas to inspect are the welds where the tubulars ('braces') that make up the underwater structure join the structure legs ('chords') in groups called 'nodes' (see Figure 1.13). Unfortunately, these nodal welds are also the most difficult to access, and developing systems to aid in the inspection of these welds by ROV is the subject of this work.

Most non-destructive testing (NDT) weld inspection by diver is done using Magnetic Particle Inspection (MPI) where the metal is magnetised and then sprayed with an ink containing fluorescent particles – the crack becomes visible because its high magnetic flux density attracts the ink. This method is generally not appropriate for ROV deployment because of the dexterity it requires, and the difficulty of interpretation by camera. Instead, the most appropriate ROV technique is generally considered to be Alternating Current Field Measurement (ACFM) developed by UCL and TSC [Raine 1996a; Raine 1996b; Pennison 1997]. This injects a magnetic field into the surface of the metal and, by measuring disturbances in the field, picks up the presence of very small defects (and unlike other systems based on eddy current principles it can be used to size defects as well as detect them).

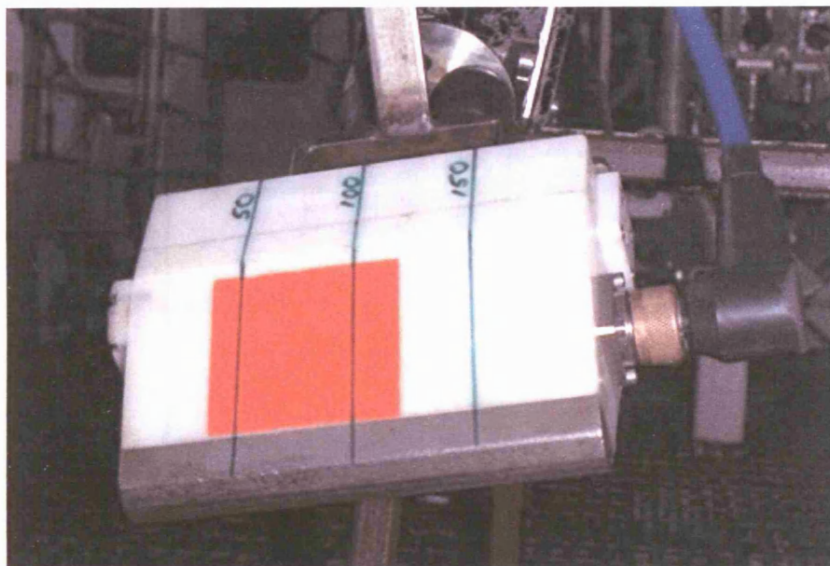


Figure 1.14 – Manipulator mounted ACFM array probe

The system is controlled by an operator in the ROV cabin using special computer software. The standard ACFM probe is required to be scanned along the weld surface but a specially developed ACFM array probe can be placed at intervals along the weld instead (see Figure 1.14). This can be done by ROV manipulator, although the requirements for even spacing and correct orientation generally require a computer controlled manipulator system. The rest of this work will assume that the ROV docking location is being determined for ACFM inspection, but nonetheless the method is equally applicable for eddy current, ultrasonic, automated MPI, or other NDT techniques, or even for other weld intervention tasks such as cleaning or grinding.

### 1.5. Non-ROV Methods

Before concentrating further on the details of ROVs it is worth noting that other systems are available for underwater intervention tasks. These range in complexity from wireline systems, through remotely operated tools, to one atmosphere diving suits.

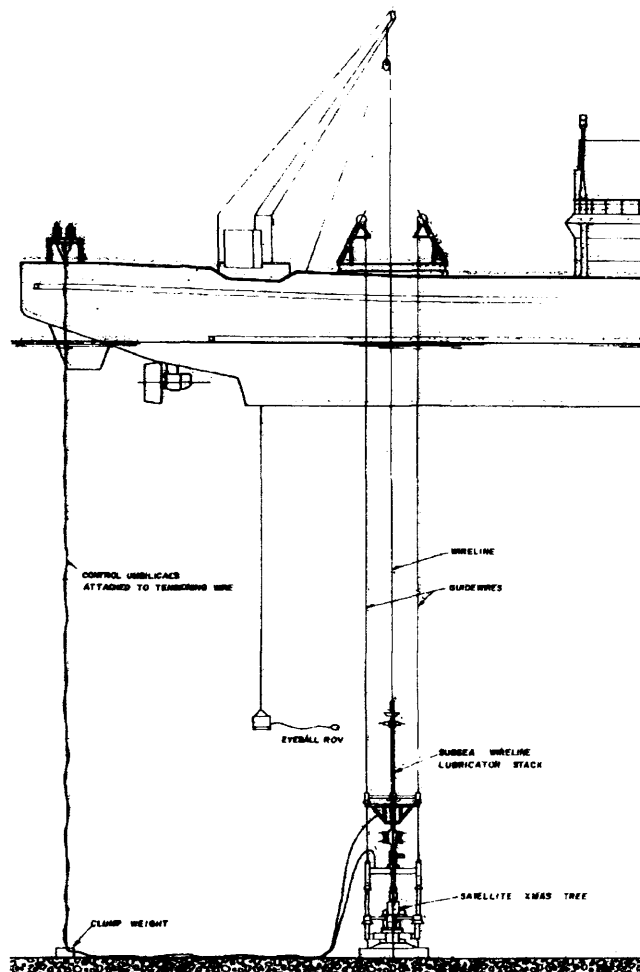


Figure 1.15 – Wireline intervention system [from Headworth 1988]



In wireline systems tooling is lowered down on a wire to the worksite; usually the equipment also has four guidewires, one at each corner. The guidewires are fixed in place for the duration of the task and ensure that the system is correctly positioned and oriented to mate with whatever equipment the system is going to work on. The wireline carries the weight of the system and is used to deploy and recover it (see, for example, Figure 1.15) while an external umbilical provides the control link.

Remotely Operated Tools (ROTs) are similar but they are deployed on an armoured umbilical from the surface which provides the control link as well as lifting and recovering the system. They often also use guidelines, but for very deep water can incorporate thruster systems, similar to those used on ROVs, to manoeuvre themselves into position. Their ability to conduct difficult intervention tasks is thus improved, and the more advanced ones may even incorporate manipulator arms (see, for example, Figure 1.16).

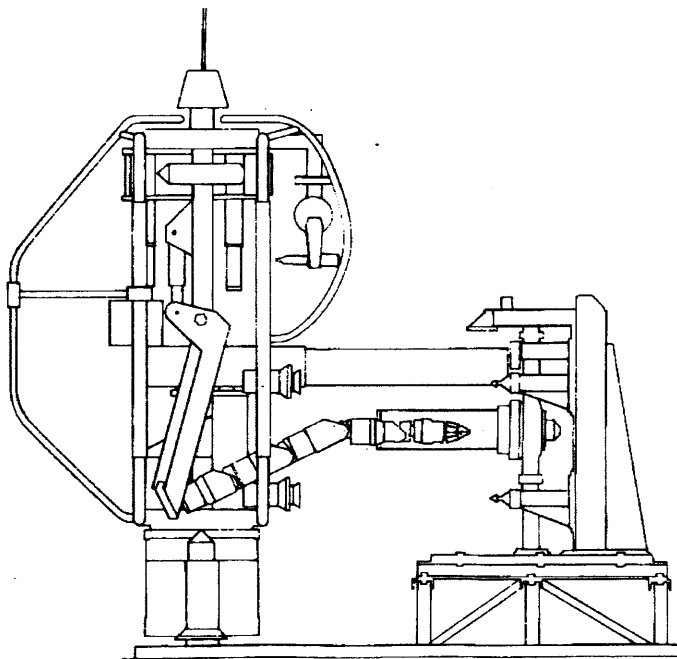


Figure 1.16 – *Wellman* Remotely Operated Tool [from Höglund 1988]

Although many of the functions originally performed by ROTs have since been taken over by ROVs, they are still used where this intervention technique has clear advantages – for example, the ability to deploy systems that are typically much heavier than can be carried by ROVs. ROT development continues, and recent improvements include the addition of 3D graphical visualisation systems to aid with mating in deep water [Wright 2002].

Where an ROT is deployed onto a subsea installation, and is capable of moving around within the structure and carrying out several tasks, it is known as a Remotely Operated Maintenance Vehicle (ROMV) [Bell 1996]. An example of this is the Saga Petroleum Snorre Subsea Production System, see Figure 1.17. While quite sophisticated, these vehicles are specific to particular structures and are not generally applicable elsewhere.

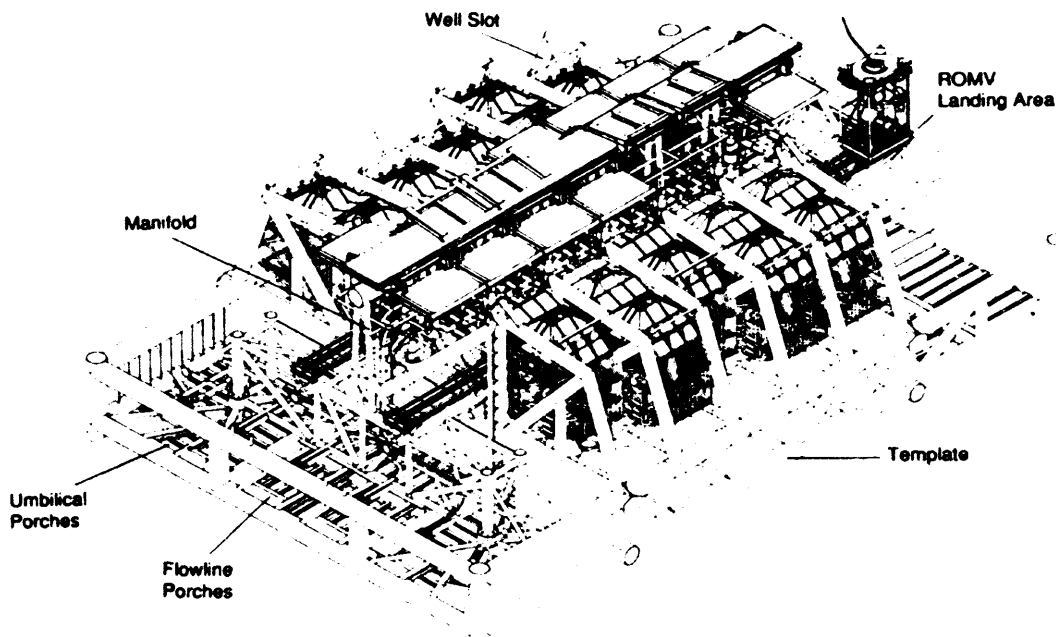


Figure 1.17 – Snorre Remotely Operated Maintenance Vehicle [from Bell 1996]

Another approach that was proposed in the late 1980s was the deployment of mini crawling vehicles to conduct weld inspection in difficult access areas (see Figures 1.18 and 1.19). None of these systems, however, are known to have come to fruition (although similar systems are now widely used for internal pipe inspection). Their disadvantages are quite clear: they require a workclass ROV to deploy them – so the ROV could probably be used to conduct the work anyway – and their small size limits their payload (i.e. inspection equipment) and manipulative ability below that available to a workclass ROV.

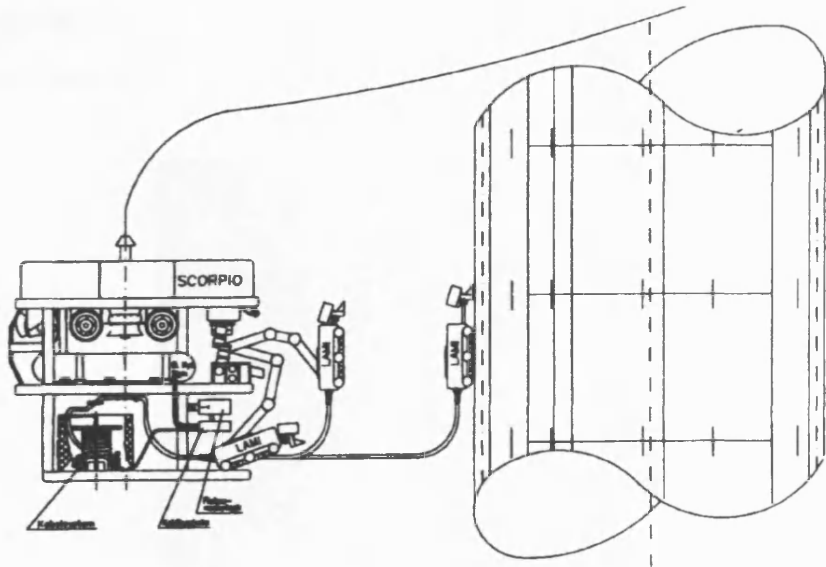


Figure 1.18 – Proposed LAMI crawler deployed by a Scorpio [from Evensen 1988]

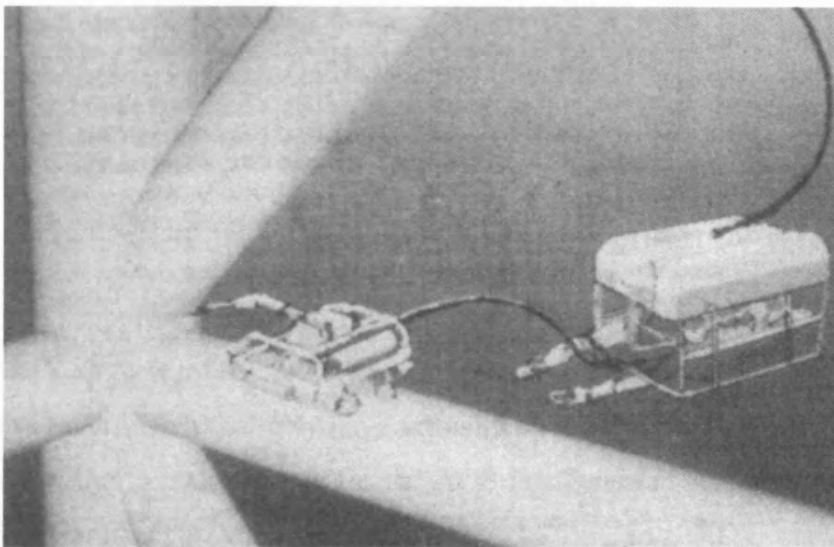


Figure 1.19 – Proposed pipe crawler conducting weld inspection [from Hughes 1988]

One of the most sophisticated non-ROV intervention systems is the atmospheric diving suit (ADS) in which a human deploys to the work site in a one-man submersible, usually fitted with very similar equipment to ROVs (thrusters, sonar, cameras and lights, etc.). The ADS has a long and distinguished history, from the original 1715 Lethbridge 'diving engine' through the successful and widespread JIM suits of the 1970s to the Newtsuits of the 1980s [Thornton 2001]. The Newtsuit was used in the early 1990s for tasks such as installation of equipment and umbilicals, making and breaking connections, and pulling-in of stab plates on riser packages [Middleton 1993]. It was also trialled by BHP Petroleum in Australian waters conducting ACFM array weld

inspection [Pennison 1997]. However, even by then it was losing out to "the capability, reliability and above all safety of ROVs" [Mills 1993].



Figure 1.20 – Hardsuit 2000 atmospheric diving suit [from Gibson 2002]

For oil and gas support, the fact that ADS systems are comparatively expensive, have an equipment back up (control cabin, LARS, TMS, etc.) virtually the same as an ROV, and put a human at risk, mean that they have virtually entirely lost out to ROVs. Currently the only market that remains strong for the ADS is submarine rescue where the latest version, the Hardsuit ADS2000 (see Figure 1.20), "has ROV-like range and manoeuvrability, with the advantage of being small enough to gain access to restricted spaces and perform intricate useful work, previously the exclusive domain of divers" [Gibson 2002]. Just as significant, though not stated, is that cost is less of an issue, and so is human safety since, by the nature of the operation, human lives are already at stake.

None of the non-ROV intervention methods considered in this section, from the Wireline or ROT to ADS, have shown that they can conduct nodal weld intervention (cleaning and inspection) as effectively, cheaply and safely as ROV/manipulator systems. The remainder of this work will therefore concentrate on these ROV/manipulator systems, and the others will not be considered further.

## 1.6. AROWS

Although most offshore oil and gas regions have driving forces for using ROVs in place of divers, the region that has seen the most innovation in this area is almost certainly Australia. The philosophy there is to not use divers unless essential and so remote methods of intervention are much preferred. Because of its remote geographical location and limited offshore activity compared to most other oil and gas producing areas of the world, many of the services readily available at short notice and nominal mobilisation time/cost in other areas, such as DSVs, are not available in Australia. In addition, due to a combination of local regulatory requirements and maritime union requirements with respect to onboard living and working conditions, a DSV used in Australia would probably need to be a North Sea vessel. As a result of the local unavailability of DSVs and the time and cost of mobilisation and demobilisation, doing work by ROV can be significantly more cost effective than saturation diving in Australia [Batten 1988].

Almost certainly the most advanced manually controlled ROV and manipulator system developed before the advent of computer controlled systems (as described in the next chapter) was the Advanced Remotely Operated Work System (AROWS) usually seen in its Jacket Cleaning Vehicle (JCV) configuration. It was the archetype for the advanced ROV and manipulator combinations that are the subject of this work, and will be described here in some detail.

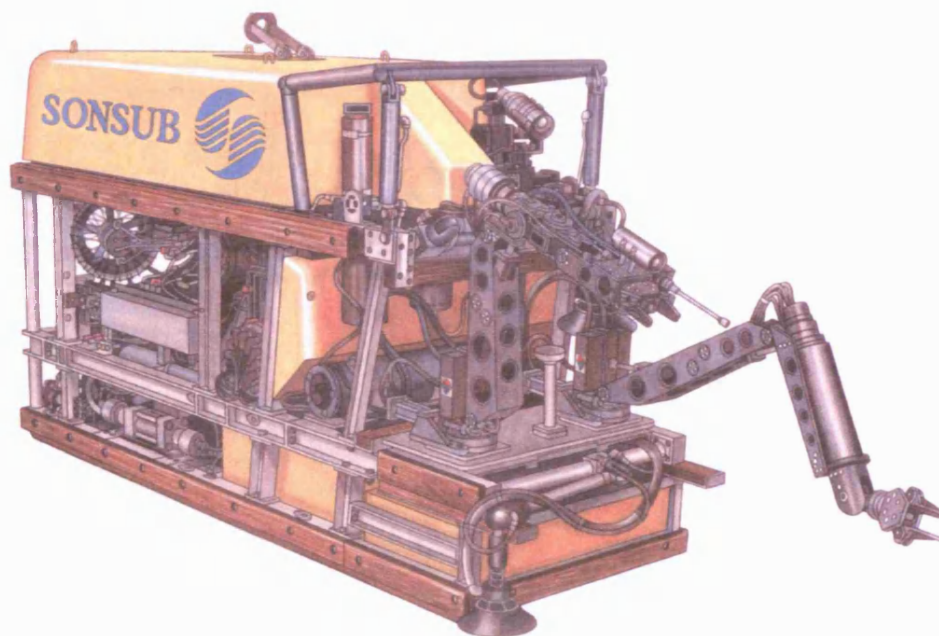


Figure 1.21 – Sonsub AROWS [from Sonsub IRST]



The AROWS idea was developed by Sonsub at their headquarters in Perth, Western Australia, during the mid-late 1980s. AROWS essentially consisted of a powerful ROV with a built-in toolskid mounting the following equipment (see Figure 1.21, and the key at Figure 1.22):

- Two manipulators on an extending and rotating base; on the right a 7 function Kraft spatially correspondent manipulator mounting a video camera and a high pressure water cleaning lance, and on the left a 7 function rate manipulator mounting a video inspection camera.
- Two legs with suction feet for attaching to structures, known as Stabilization Suction Manipulators or Articulated Attachment Devices (AAD).
- A sector scanning sonar for obstacle avoidance when navigating into structural jackets.
- A pan and tilt unit, mounting a zoom video camera, that could be elevated hydraulically above the level of the buoyancy, along with the sonar, to allow the vehicle to manoeuvre backwards as effectively as forwards.
- Optional equipment for cathodic protection (CP) monitoring, sub-bottom profiling, sidescan sonar, non-destructive testing (NDT), pipeline tracking and leak detection.

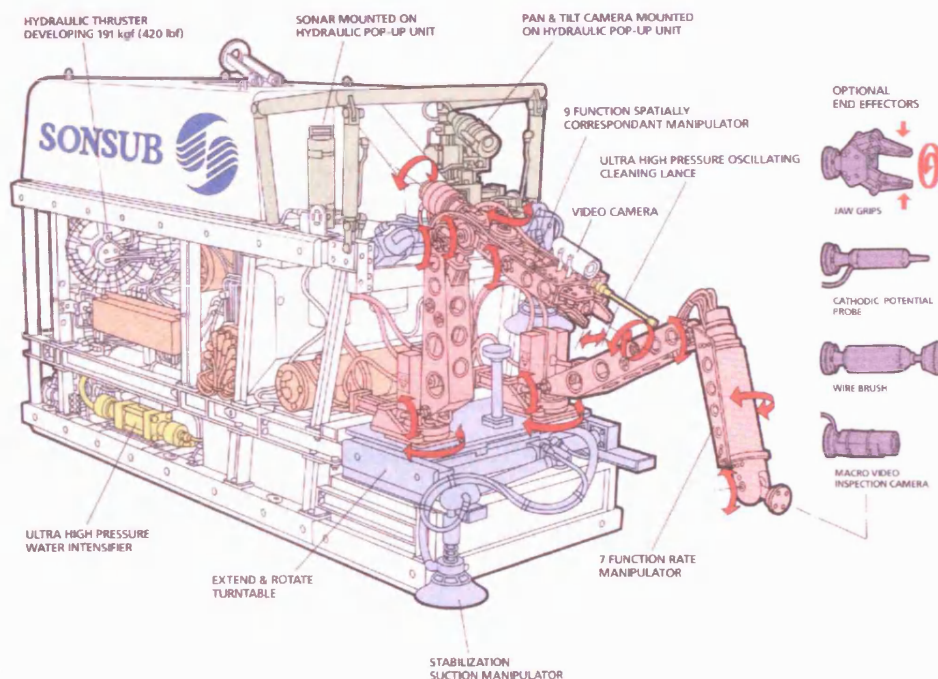


Figure 1.22 – Key to Sonsub AROWS components [from Sonsub IRST]

The first AROWS was originally based on the Sonsub Challenger ROV carrying a 7 function Kraft manipulator (as in Figures 1.21 and 1.22). However, for dextrous work such as cleaning nodal welds, it could be fitted with a 9 function Slingsby TA33 manipulator; this version saw extensive service working for Esso in the Bass Strait conducting nodal weld cleaning, close detailed inspection and still photography [Harman 1988], see Figures 1.23 and 1.24.

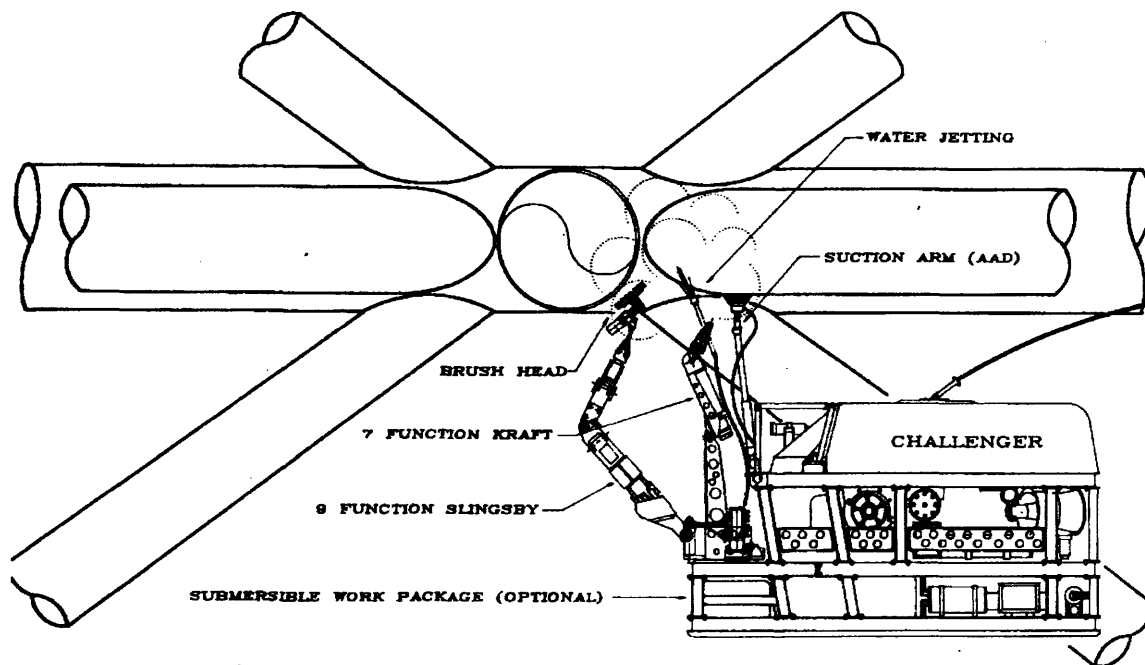


Figure 1.23 – Diagram of AROWS nodal weld cleaning [from Harman 1988]

The Challenger AROWS was developed in conjunction with Perry in Florida, and Perry later developed the Triton AROWS (see Figure 1.25). Although originally fitted with the same 7 and 5 function manipulators as the Challenger AROWS, later versions were fitted with Schilling Titan and Rigmaster manipulators respectively.

As well as seeing use in Australia, both Challenger and Triton AROWS systems were operated by Wilsb AS in Norway, in conjunction with its UK parent company, Sonsub Services Limited. Operations included the 1986-1988 node inspection and Cleaning ROV (CLEROV) contracts for Elf Aquitaine Norge [Harman 1988].

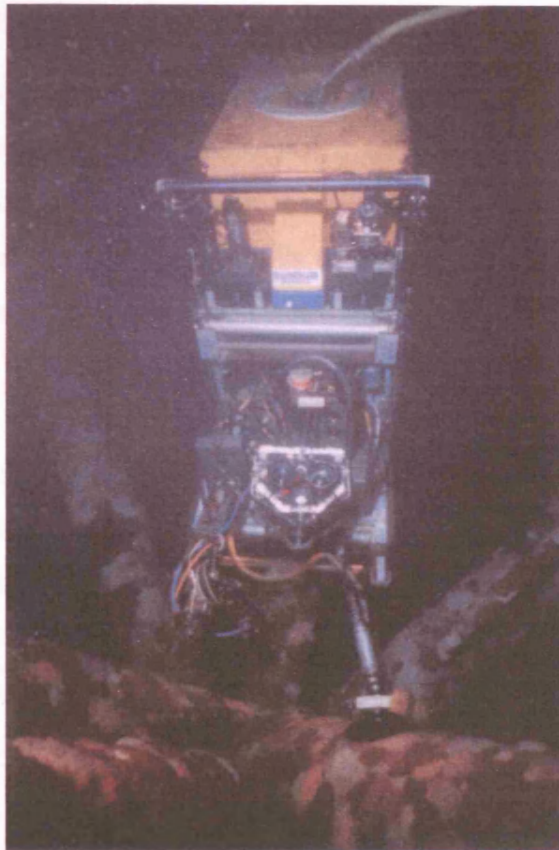


Figure 1.24 – Photograph of AROWS nodal weld cleaning [from Sonsub IRST]

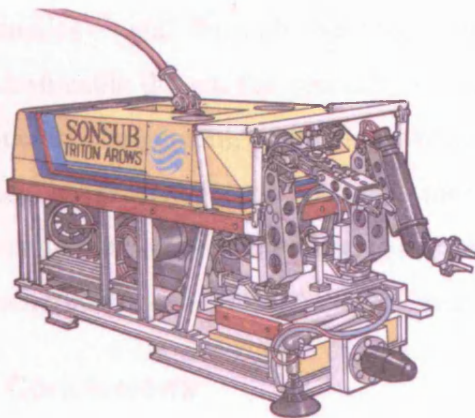


Figure 1.25 – Photographs of early and late Triton AROWS [from Sonsub IRST and Sonsub Triton ROVS]

Aside from the unique ability to elevate the pan and tilt unit and sonar, the AROWS configuration was the forerunner of all the ROV/manipulator intervention systems that are the subject of the remainder of this work – most of them differing only in manipulator configuration (sometimes having just one central arm) and in the degrees of freedom used in the manipulator mounting base (either yaw or roll, with or without extension, etc.).



## **1.7. Force Controlled Manipulators**

The next chapter will look at the benefits of applying robotic control to ROV manipulators but most early work concentrated on improving manipulation by giving force feedback to the human operator. Force feedback manipulator systems were developed by General Electric in the 1960s to prevent damage to components during manipulation [Schilling 1996] and have been widely used in the nuclear industry [Carre 1991]; during the 1970s and 1980s the technology was applied to developing underwater force feedback manipulators. They are claimed by the manufacturers to be capable of performing tasks of significantly greater complexity than conventional manipulators, in a much shorter period of time, and with less chance of damage to the arm or worksite [Harbur 1999]. However, they have never received widespread acceptance due to perceived problems of operator fatigue (from operating a master arm controller against forced resistance), reliability, and cost, and so the majority of subsea manipulators continue to be operated by conventional master arms.

However, research in this area continues for terrestrial teleoperation – for example, the Massachusetts Institute of Technology (MIT) has developed a very advanced 'Whole-Arm Manipulator' (WAM) for bomb disposal and similar tasks [Townsend 1999]. The WAM master and slave arms are essentially identical – and may have redundant kinematics – and through the combination of force sensors at each joint, and low backlash cable drives, the operator is able to push and pull each link of the slave into a particular arrangement, and feel any resulting forces, through proxy manipulation of the master. Currently some of the more interesting work on force control for telemanipulation is for telesurgery, in particular looking at the problems of force control on compliant objects [Dhruv 2000, Çavuşoğlu 2001].

## **1.8. Conclusions**

This chapter has described the history and development of ROVs for the oil and gas industry and has shown the superiority of advanced ROV/manipulator combination systems for difficult intervention tasks such as nodal inspection. It has shown that systems like AROWS have nearly all the capabilities required for detailed NDT of nodal welds, though they lack the dexterous probe handling required for techniques such as ACFM. The promise of improved manipulator dexterity through force feedback never materialised and so the next chapter will show how the application of robotic control technology can provide the means for nodal weld inspection to be effectively conducted by ROV/manipulator combination systems.

---

## CHAPTER 2: ROBOTIC ROV MANIPULATOR SYSTEMS

---

### 2.1. Introduction

From early on it was recognised that "for reasons of cost, safety and inspection reliability, it would be advantageous to develop an automated underwater structural non-destructive testing system" [Van Den Hooff 1988]. However, the manipulators used on early ROVs had been of a relatively primitive design. This meant that ROVs had largely been restricted to tasks requiring little manipulative dexterity, such as debris removal and some forms of cleaning, whereas a large amount of the work remaining to be done, such as weld inspection, required precise and repeatable positioning [Savut 1985; Broome 1986; Broome 1989]. In order to achieve a considerable improvement in the capability of an ROV system to undertake such tasks, two main areas had to be addressed - the design and construction of improved manipulators, and the design and implementation of suitably advanced control systems [Broome 1991].

A study in the UK in the 1980s examined the potential for full inspection of platforms by ROVs and the use of geometric modelling to enable manipulator arms to be operated in real-time [Hayward 1991]. The conclusions at that time were that it was beneficial to secure the ROV to the structure by a clamp, about which it could then move. Also, geometric modelling was a help, though not a necessity. At the same time, Japanese companies were looking at similar problems; Mitsui Industries designed a system with detachable multifunctional legs allowing a vehicle to attach itself to uneven, asymmetric structures using suction pad "feet" [Hayward 1991].

Also, it was being predicted that: "In three years whole new tasks could be performed robotically as a routine and very much more cost effectively than any would dare hope at present" [Marsh R. 1991]. A number of advanced robotic concepts were proposed early on, such as the 'Underwater Robot' (UROB) [Hansen 1988] and the 'advanced underwater robotic system' [Russell 1990]. Most savings were expected from developing full NDT capabilities, and also remotely controlled welding – "by the end of

this decade there should be absolutely no technical or economic reason why almost every subsea task is not being done either robotically or by remote control” [Marsh R. 1991].

## **2.2. Research and Development**

Meanwhile there was significant development work taking place in research establishments aimed at producing robotic manipulator arms for underwater intervention tasks. These were typically based around conventional (electric) factory robots or adapted commercial ROV (hydraulic) manipulators. For example, research was being conducted at GKSS in Germany and Shell Laboratories in Amsterdam using ASEA factory robots, and at University College London (UCL) using a Puma factory robot. Concurrently, UCL were also conducting experiments using a GE ROV manipulator and Heriot-Watt University were doing research with Slingsby TA9 manipulators.

The team at GKSS concentrated on using a robot for welding, and tested it in a hyperbaric chamber to a pressure of 110bar [Aust 1988], see Figure 2.1. GKSS later went on to conduct tests with a number of other electric factory robots – not just at high pressure but, after appropriate adaptation and marinisation, in water. These included the Siemens Manutec r15 robot, which was demonstrated operating successfully in dry and wet tests at up to 110bar [Aust 1995], and eventually the Neos Tricept robot in the European RobHaz project (in which this author was also involved, and provided the robot interface program).

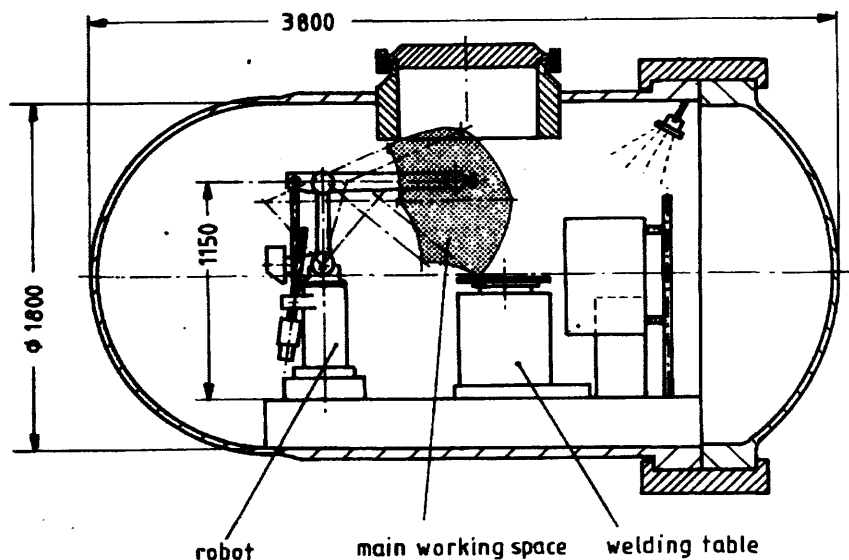


Figure 2.1 – Test arrangement of ASEA robot in pressure chamber [from Aust 1988]

The Shell Laboratories system was developed in dry lab tests and used an eddy current sensor mounted on the robot [Van Den Hooff 1988], see Figure 2.2. Small adjustments were made during weld tracking by measuring with a camera the offset of a target projected by laser onto the weld surface.

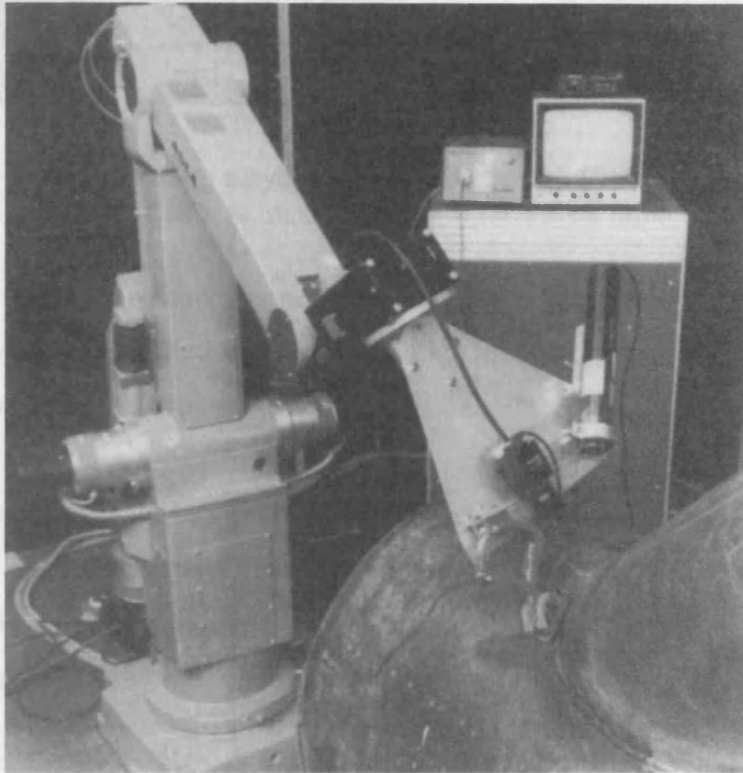


Figure 2.2 – Shell Laboratories' ASEA robot and test node [from Van Den Hooff 1988]

The Ocean Systems Laboratory group at Heriot-Watt conducted research with TA9 manipulators into a number of relevant areas. They looked at hybrid position/force control where a manipulator under computer control makes use of force interaction information to improve contact tasks [Lane 1991; Clegg 1995; Dunnigan 1996], see Figure 2.3a, and so avoid some of the problems of manual force feedback systems (see *Section 1.7. Force Controlled Manipulators*). They also looked at co-ordinated computer control of multiple manipulators on an ROV [Kato 1996] and supervisory control and task planning for underwater vehicles [Lane 1994; Lane 1995], see Figure 2.3b. Other work by the group has looked at the use of visual systems for ROV station-keeping and docking, and consideration of the effects of manipulator motion while station-keeping (these are discussed in more detail in *Section 5.1.1. Working Without Docking – Dynamic Positioning*).



Figure 2.3 – TA9 performing a hybrid position/force task [from Dunnigan 1996] and insertion of subsea mateable connectors [from Lane 1995]

A number of areas of research concerned with the problems of weld inspection by manipulator were investigated by the Automatic Control Group (ACG) at UCL under Dr. (later Prof.) David Broome. Work was done on determining the location and orientation of intersecting cylinders (as in a structural node), both the theoretical background [Wray 1985; Hughes 1988] and practical developments in the laboratory using a Puma robot with a touch sensor [Hughes 1988; Greig 1989; Greig 1992], see Figure 2.4. Work was also done on operating a General Electric (GE) hydraulic manipulator in Cartesian co-ordinates under control of an IBM PC [Hughes 1988], see Figure 2.5.

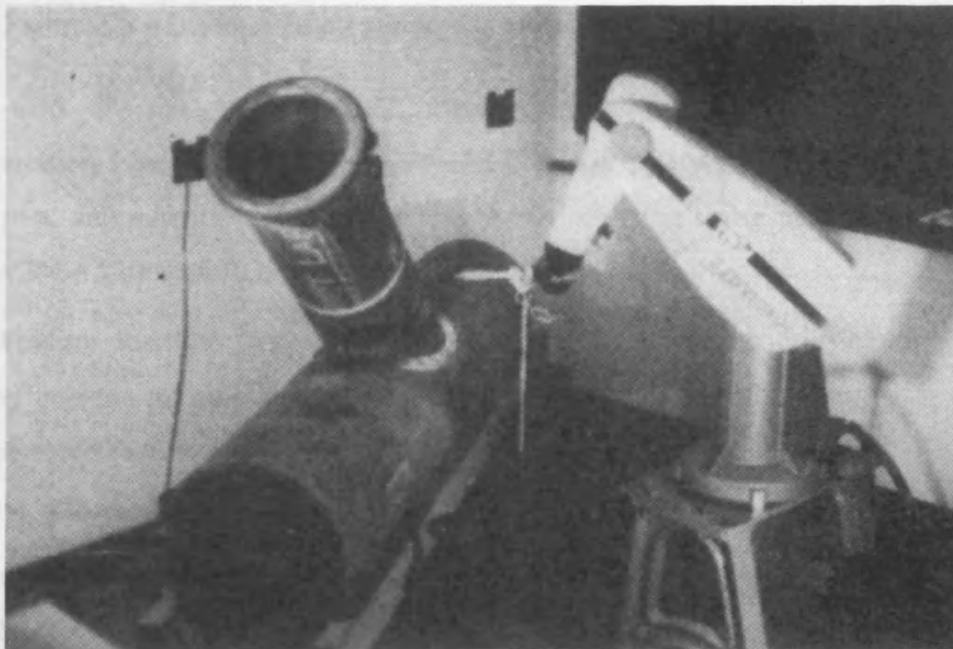


Figure 2.4 – Puma robot conducting cylinder location at UCL [from Hughes 1988]

The research conducted at UCL laid the foundations for the ARM Project (see *Section 2.6. ARM*) which was the original test bed for this thesis work. Although the ARM control software was written from scratch by David Broome's Control and Robotics Group (which included this author) at Technical Software Consultants Limited (TSC), it incorporated many of the ideas developed at UCL, including the cylinder location technique described below.



Figure 2.5 – GE manipulator under PC control at UCL [from Hughes 1988]

Later work by the ACG at UCL included a stereo vision system for motion compensation (see *Section 2.6.1. ARM Development*), in which this author played a small part, and adaptive force control for weld following using neural networks (see *Section 3.4.4. Use with ROVs and AUVs*).

The Welding Institute (TWI) conducted research in conjunction with the Danish welding institute into ultrasonic inspection [Kenzie 1990]. In laboratory trials a 6-axis welding robot successfully deployed an ultrasonic immersion probe around a T-node in a testing tank.

At Cranfield University research was conducted into resolved motion control of a Slingsby TA9 manipulator and, in parallel, into joystick control of an ASEA factory robot with 3D graphical feedback from simulation software [McMaster 1994]. Research at the University of Pennsylvania looked at control of a PUMA robot via an acoustic link and with 3D graphical feedback [Sayers 1994].

UK Robotics Limited developed a system known as ATC Craftsman that combined a joystick control box with an advanced, scripted robot control system [Pegman 1999]. It was demonstrated on Schilling Titan (see Figure 2.6) and Slingsby TA40 manipulators, but it appears to have been targeted primarily at the nuclear industry.

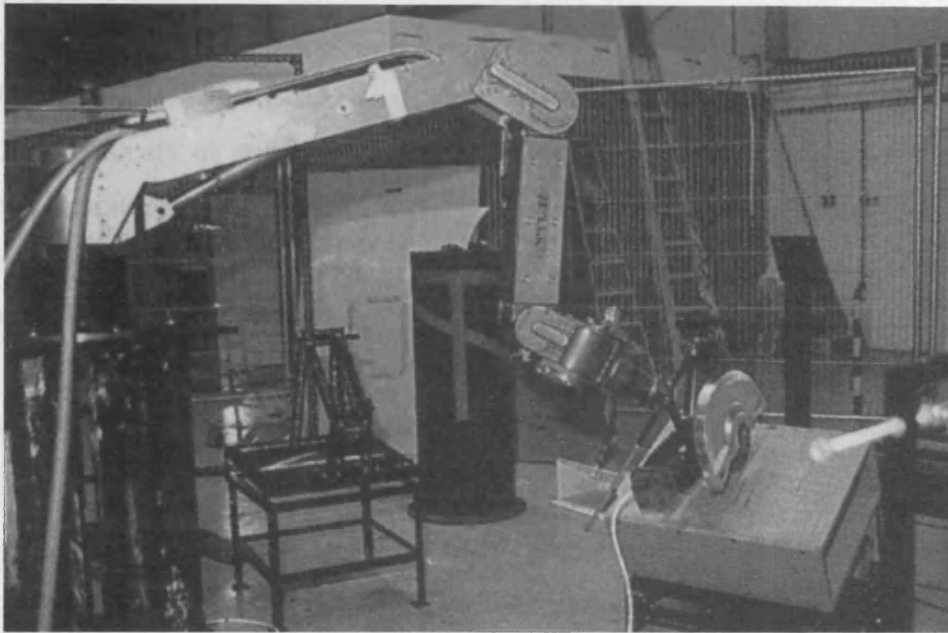


Figure 2.6 – ATC Craftsman controlling a Titan II manipulator [from Pegman 1999]

By the end of the decade it was accepted that in telerobotics, approaches "...with a human operator interacting and co-operating with a computer demonstrate many clear advantages" [Pretlove 1999], a philosophy known as Supervisory Control.

### **2.3. Supervisory Control Systems**

Supervisory Control essentially means that the manipulator is put under computer control but, unlike a terrestrial robot, it remains under the detailed supervision of a human operator at all times. He is able to step in and suspend or modify the operation at any time, an important feature of systems designed to work in unstructured and semi-structured environments such as underwater platform jackets. Manipulators with supervisory control are known as Computer-Aided Telemanipulation (CAT) – or sometimes Computer-Aided Teleoperation – systems.

By the early 1990s it was clear that underwater manipulator systems would benefit from incorporating computer control, for example "Supervisory control of manipulator functions where precise positioning of an end-effector is required, is an area of operational potential" [Mann 1990] and "By extending an existing ROV and



manipulator with computer control, the capability of the system can be considerably increased... A semi-automatic work system requires development of a Supervisory Computer Control System (methodology for combining man and computer control)" [Sortland 1990].

Three main supervisory control ROV/manipulator combination systems were developed for nodal weld inspection; they were known as REMO, ATES and ARM and will be described in the following sections. They each had a hardware configuration that echoed, consciously or not, the AROWS configuration of a workclass ROV carrying advanced manipulators mounted on a deployment system with its own motions or degrees of freedom (DOF), and attaching to structures using suction feet. These attachment systems are known by various names, particularly 'attachment legs' and 'sticky feet' (though the latter term really only applies to the suction pad on the end of the leg), and both terms will be used interchangeably through the remainder of this work.

These three systems spearheaded the use of 3D graphics as part of the ROV/manipulator user interface, although prototype GUI systems had been developed for nuclear teleoperated systems [Carre 1991; Dotan 1991; Even 1991]. Later, the use of 3D graphical interfaces became more common; including Oceaneering's Modular Integrated Man-machine Interaction and Control (MIMIC) system for ROVs [Hallset 1994], the *Magellan* CAT system for the Schilling range of manipulators [Lemoine 1995], GRL's more recent *ROVolution* system for ROVs and manipulators [Larkum 2000; Larkum 2002], as well as various systems for AUVs [Brutzman 1995; Hornfeld 2002]. By 1996 it was accepted that "both offshore and nuclear markets increasingly appreciate the utility of a 3D graphical model of a robotic system that is calibrated to the task environment and linked to the robot or tool controller. This model-based control... is proving to be a vital component in the reliable, efficient performance of remote routine operations..." [Schilling 1996].

## 2.4. REMO

The REMO system operated by Stolt Comex Seaway (SCS), now Stolt Offshore Limited, is believed to have developed from the MARI project funded by Comex UK. This designed an ROV inspection system with a manipulator and "limpet" attachment legs (see Figure 2.7), and specifically developed suitable NDT equipment for it including automated MPI and eddy current probes [Duncan 1990].



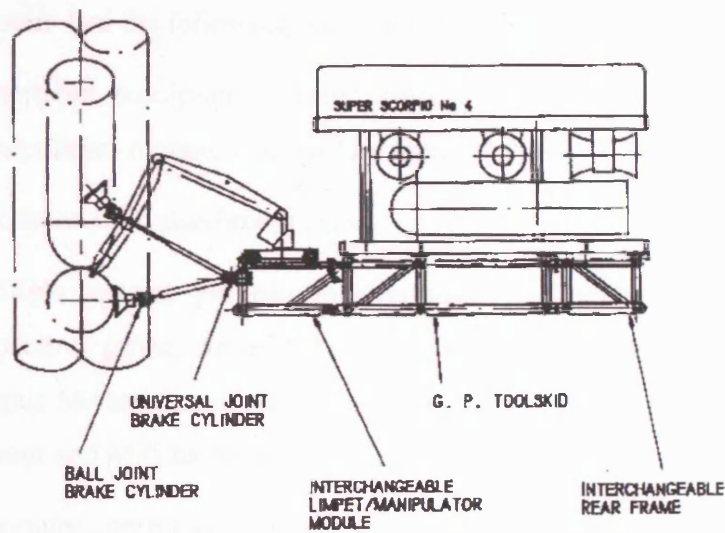


Figure 2.7 – Proposed MARI Advanced Robotic System [from Duncan 1990]

REMO itself was developed in a joint project between Elf Petroleum Norge, Phillips Petroleum Company Norway and Stolt Comex Seaway. It followed from extensive experience Elf had obtained between 1983 and 1987 in using workclass ROVs to clean structures, prior to divers being brought in to inspect them [Ricci 1996]. An initial project from 1990-1992 proved the viability of the computer technology and tool systems, and in the middle of 1993 Elf issued a tender for construction of the system. The contract was awarded to Stolt Comex Seaway and, after evaluation, a special ROV was designed and built from scratch for the project, although it used many components from Stolt's standard SCV ROVs.

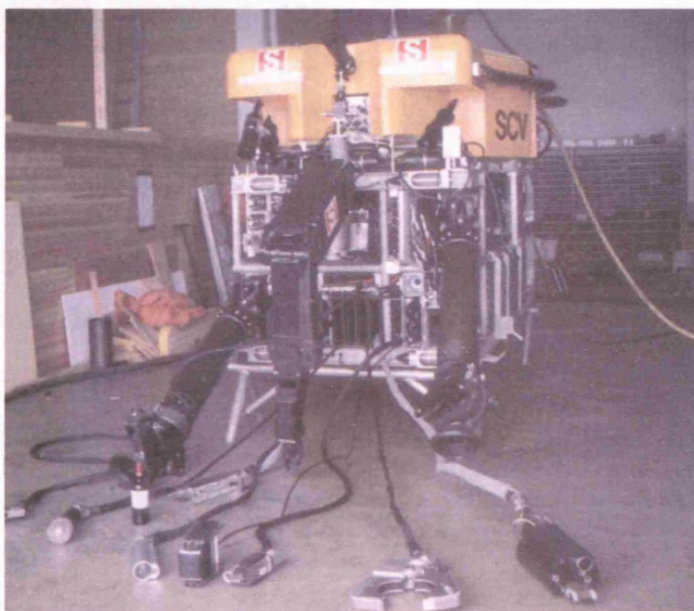


Figure 2.8 – REMO displaying its various IRM tools [from Stolt Comex Seaway RR]

The REMO system had the following equipment (see Figure 2.8):

- Cybernétix manipulator (fitted with a wrist unit from a Schilling Titan manipulator) mounted on a rotary table.
- Two telescopic attachment legs.
- Multiple special purpose intervention tools, including an eddy current inspection probe, an eddy current array probe ('Math Scan'), a Current Output Measurement tool for CP monitoring, and an MPI tool. The eddy current and MPI tools incorporate force feedback.
- Computer control system used in conjunction with stereo cameras to acquire a model of the workpiece.

The computer controller provided facilities for automatic tool handling, trajectory generation, teach and replay operations and manual overlay during automatic trajectory control. It included an advanced man-machine interface with 3D graphics (see Figure 2.9)

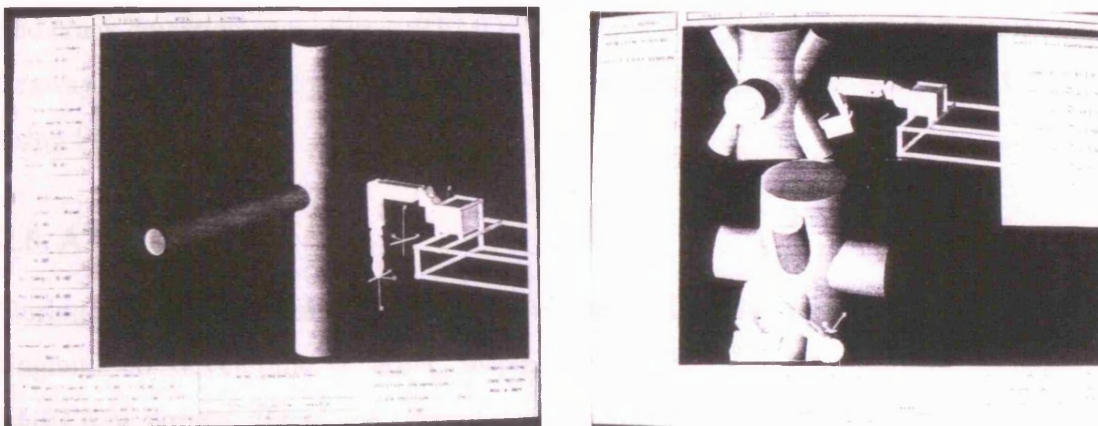


Figure 2.9 – REMO 3D graphical user interface [from Ricci 1996]

The prototype REMO had passive attachment legs which were placed on the structure by the manipulator. They were awkward to install and fragile, and made adjustment of the ROV position difficult, and so were replaced on the production system by hydraulic powered 5 function legs.

REMO underwent dry trials, then shallow water trials (see Figure 2.10), then offshore trials during 1994.





Figure 2.10 – REMO system during ACFM trials [courtesy TSC Ltd]

During 1995 it conducted a 40 day inspection campaign on the Phillips Ekofisk Field during which it cleaned and inspected some 30 nodes: "It satisfied certification authorities and proved superior to using divers with respect to accuracy, repeatability and crack detection capabilities. REMO found four cracks during the job." [Stolt Comex Seaway RR]. Later that year it conducted the annual inspection campaign for Elf on the Frigg Field.

## 2.5. ATES

During the late 1980's and early 1990's Tecnomare in Italy were working with AEA in England on developing a Work Inspection Robot (WIR) as part of a European EUREKA project. This was aimed at designing an advanced ROV system with attachment legs, and a high precision manipulator on a telescopic boom, primarily for nodal weld inspection [Smith 1990]; this system is believed to be the forerunner of ATES.

ATES (Advanced Telemanipulation System) was developed under the European Thermie programme by Saipem, in conjunction with Sonsub<sup>1</sup> and Tecnomare, during a 4 year project running from late 1992 [Brambilla 1996]. The aim of the project was to

---

<sup>1</sup> It is interesting to note that the Sonsub Engineering Manager responsible for ATES testing, Espen Moller, was previously the REMO project manager.

develop a general purpose subsea robotic system based on work performed up to the mid-1980s [Moller 1996].



Figure 2.11 – ATEs advanced robotic system [from Sonsub CN3/1]

The subsea part of ATEs comprised a toolskid carried under a standard ROV and carrying the following equipment (see Figure 2.11):

- Schilling Titan manipulator with force/torque control sensor at the wrist.
- Attachment legs designed to hold the system onto a work site, but not necessarily to hold it rigid.
- Tecnomare TV Trackmeter mounted on a pan and tilt unit. The Trackmeter is a non-contact stereoscopic measuring device which uses an advanced form of real-time photogrammetry to construct a digitised image of the worksite [Sonsub CN]. It is able to register targets in its view and calculate residual motions of the toolskid from the apparent motion of the targets.
- Motion reference unit (MRU) to provide the system with roll, pitch and heading data.
- Computer control system used in conjunction with the Trackmeter to operate the manipulator with automatic compensation for movements of the ROV.



In the surface control cabin there were computers for the ATES supervisory control computer and the TV Trackmeter. The control system included an advanced man-machine interface with 3D graphics (see Figure 2.12), and a joystick for resolved motion control of the manipulator in multiple co-ordinate frames.

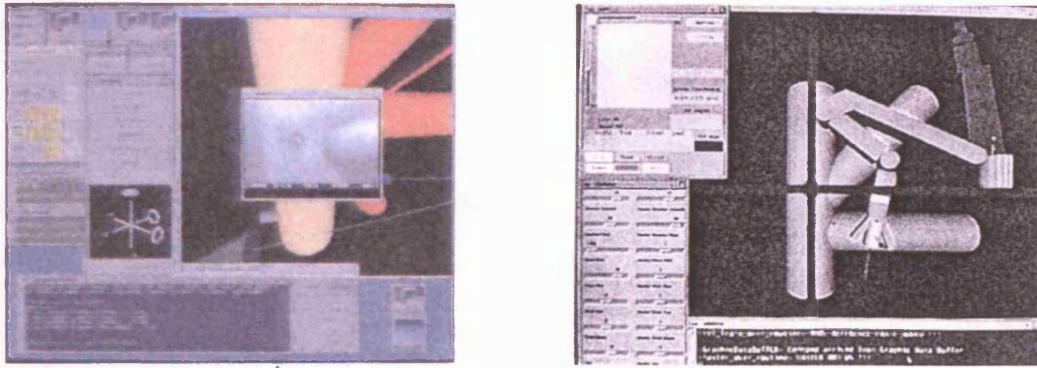


Figure 2.12 – ATES user interface [from Sonsub ATES RSE and Sonsub CN4/2]

The first prototype, ATES 1 (see Figure 2.13), was completed in 1996 and tested mounting a Titan II manipulator and TV Trackmeter. ATES 2 was an upgraded system with a Titan III manipulator and improved Trackmeter, and ATES 3 incorporated improved control software. Two ATES systems were built with the intention that while one was being tested the other was being developed to the next stage, so that at one time there would be an ATES 1 and an ATES 2, then an ATES 3 and an ATES 2, and so on.

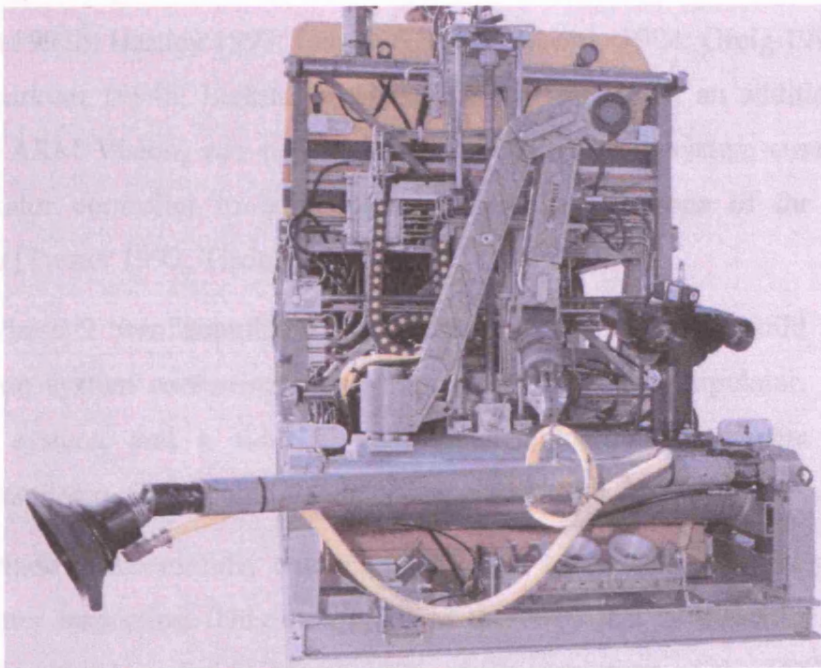


Figure 2.13 – ATES 1 prototype [from Sonsub ATES RSE]

ATES was successfully demonstrated cutting steel plate, stabbing and air/grit cleaning, as well as making and unmaking hydraulic connections, turning a valve, connecting and disconnecting a shackle, and conducting NDT on a test plate [Moller 1996]. These tests were conducted during 1995 at the GMC tank in Aberdeen, at a pier in Stavanger and in the Stena tank in Aberdeen [Brambilla 1996], in each case carried by Sonsub's Triton 16 ROV.

## **2.6. ARM**

The ARM system was the basis for the work undertaken for this thesis and will be described in some detail.

### **2.6.1. ARM Development**

Under the sponsorship of Mobil North Sea Limited and the UK government's Oil and Gas Projects Supplies Office (OSO), a project was begun in 1991 to design and build a new, advanced ROV-based node cleaning and inspection system. It would incorporate a completely new underwater manipulator - to be designed, built and tested by Slingsby Engineering Limited (SEL) - and an advanced computer control system, designed, built and programmed by the Control and Robotics group of Technical Software Consultants Limited (TSC) in conjunction with University College London (UCL). The first build manipulator and control system were completed during ARM Phase 1 which ended in September 1994 [Broome 1993a; Broome 1993b; Hartley 1993; Langrock 1993; Broome 1994; Greig 1994; Langrock 1994; Larkum 1994a; Larkum 1994b]. At about this time, an additional research project, ARM Vision, was conducted into using a vision system connected to the manipulator controller to compensate for residual motions of the ROV when attached [Turner 1993; Tisdall 1994].

ARM Phase 2 was completed in September 1995 with the build of an entire inspection system consisting of an improved Slingsby manipulator, an enhanced control system, and a toolskid (with a manipulator deployment system and attachment legs) which could be under-slung on any standard work-class ROV.

ARM Phase 3 successfully demonstrated a full commissioning trial of automated underwater inspection. This took place in the NHC test tank facility in Aberdeen over a four week period in March 1996, at the same time as the TSC Control and Robotics Group became General Robotics Limited (GRL). The complete ARM

System, carried on a Slingsby ROV known as the Multi-Role Vehicle (MRV) [Hartley 1992], is shown in Figure 2.14. It was deployed on an 18 ton T-piece node and demonstrated that it could successfully attach, conduct inspection work on the node, and find defects in the weld [Broome 1995a; Broome 1995b; Broome 1996; Larkum 1996a; Larkum 1996b; Parkes 1996; Slingsby 1996; Larkum 1998; Heale 1999].



Figure 2.14 – ARM System carried on an MRV ROV during the NHC trials

The complete system was again tested during 1997 conducting large scale ACFM array inspection trials on a library of nodal welds as part of the European EDICS project. These trials were also very successful, with the ARM system producing a Probability of Detection (POD) on defects at least as good as a human diver.

During 1997 and 1998 GRL developed the ARM control system further to allow it to conduct other dextrous tasks as well as ACFM inspection, including subsea welding and grinding. Wet welding trials were conducted for Amerada Hess at the end of 1997 [Allerton 1998] and wet grinding trials took place in 1998.

#### **2.6.2. ARM Description**

The ARM supervisory controller comprises a fast PC surface graphics control unit linked via the ROV umbilical to a subsea arm controller. This provides full control

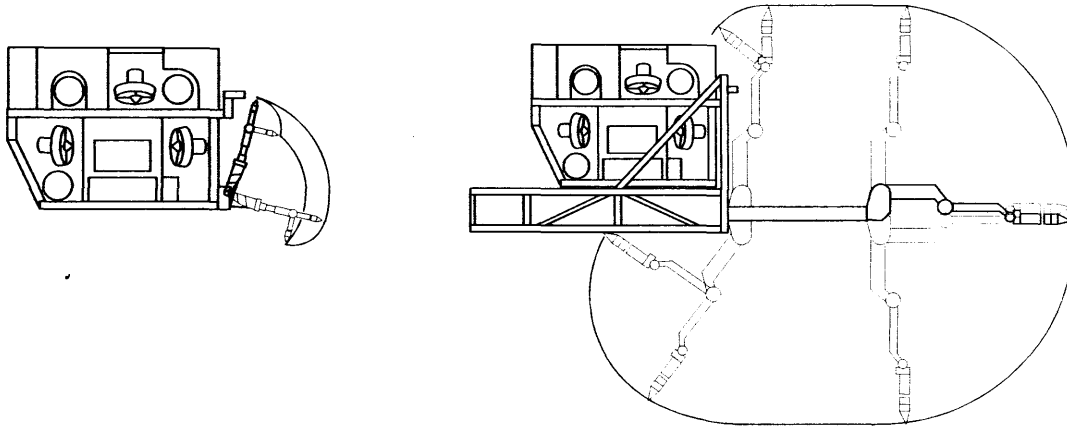


of the manipulator, including manual master-slave telemanipulation and fully automated, robotic task execution. A 3D video representation of the ROV, manipulator and work site is presented to the operator using solid-shaded colour graphics, and can be used to monitor the arm whilst planning or executing tasks. The main graphical view has facilities for panning left and right and tilting up and down and around the work space, and also the ability to zoom into and out from points of interest. A camera view can also be defined on the ROV and the view quickly changed to the camera viewpoint and back again as required. Secondary views can be used to display plan and elevation views of the arm and its work area. The arm can be controlled by a choice of input devices including keyboard, mouse, master arm or joysticks and operated in a variety of co-ordinate systems such as joint, world, tool, workpiece, etc.

All the subsea ARM equipment is mounted in a 3 tonne toolskid capable of being carried on any work-class "ROV of opportunity". This toolskid is an aluminium and steel box frame structure mounting the following equipment:

- An advanced manipulator. It has a long reach of 2.5m, and large angular ranges at each joint (typically 270°) allowing a very high degree of dexterity and excellent access capability.
- An extend/rotate deployment system. This consists of a steel box frame running down the centre of the toolskid and mounting the manipulator. It can be extended up to 2m in front of the toolskid, and it can rotate the manipulator shoulder through 360 degrees. This allows the arm to reach into work sites that the ROV cannot access (see Figure 2.15), and enables the arm to work as easily on its side or upside down.
- Attachment legs. These consist of hydraulic extending legs mounted on the toolskid and terminated with suction feet. Three arms are the minimum required for stable attachment, and they are usually arranged in a tripod for maximum rigidity. This requires one to be attached to the top front face of the ROV but a 'goal-post crossbar' is provided for this so no modification is required to the ROV. The sticky feet can be attached anywhere on the toolskid as required but the usual configuration is for one on either side attached at the required height on the 'goal-post uprights' and the third on the crossbar.





**Figure 2.15 – Diagram comparing the working volume of a standard offshore manipulator with that of the ARM System**

Once the ROV is docked onto a node, the arm follows a pre-programmed sequence of moves to touch components of the node. The information gained through this process – known as 'workpiece modelling' – is used to update the computer model of the node for relative position and orientation. The standard ARM System for workpiece modelling a cylinder, such as the chord, uses a proximity switch to touch it in a sequence of three patches, each patch consisting of a square of four contact points. The software calculates the vector cross product of the normal of each patch to determine the cylinder axis. To completely model a node it is necessary to model the chord and one other cylinder, determining the node centre from the intersection of the two cylinder axes.

### **2.6.3. 3D Graphical User Interface**

The ARM Computer System provides the manipulator operator with a very advanced man-machine interface (see Figure 2.16) that makes use of the Microsoft Windows Graphical User Interface (GUI). It runs on a high specification IBM PC compatible and the graphics are typically displayed on a large screen monitor.

A special purpose CAD facility enables the construction of a range of workpieces, based mainly on cylindrical or plate elements. This permits realistic workpiece models to be generated from simple plate specimens such as t-butts up to complex tubular nodes. In addition, a range of fixtures can be added such as sacrificial anodes, risers, j-tubes, etc. All such items added to the model are checked by the computer for collision with the manipulator during any tasks. Enhancements to the graphical views include the ability to model and display weld seams, showing the weld toes to be cleaned or inspected.

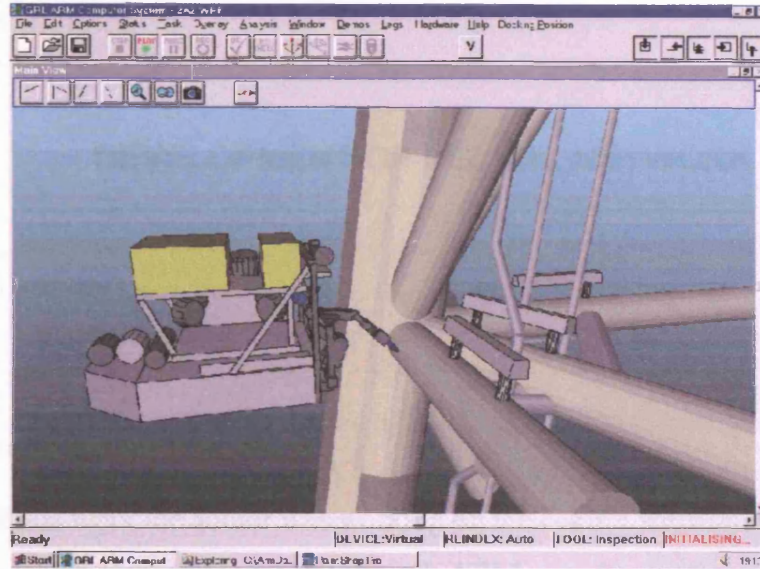


Figure 2.16 – ARM Computer System screen showing a Mobil Beryl B node

The ARM hardware fell into disuse from 1999 but the ARM software was further developed and adapted to the Schilling Titan range of manipulators. This development is described in *Section 9.1. Introduction*, in Chapter 9.

## 2.7. Summary

The requirement to conduct detailed NDT inspection on nodal welds led to the development of three advanced, robotic ROV/manipulator combination systems – REMO, ATES and ARM – during the 1990s. Although they were fairly different in detail, they had many elements in common such as an advanced manipulator, a manipulator deployment system, and attachment legs, and each demonstrated the ability to conduct weld inspection (for example, all successfully took part in ACFM array probe trials as part of the European ICON project [Raine 1996a]).

However, the ability of each system to access required welds for inspection was very dependent on the initial location chosen to dock the ROV with its attachment legs. Often a good location for attaching the legs was a poor one for manipulator access and vice versa. At the same time, attempting to plan locations for docking manually was very difficult because of the many variables in the system – where to attach each leg, how to set the manipulator deployment system, how to choose the manipulator configuration for best access, and so on.

The remainder of this thesis will look at different methods for choosing the optimum docking location, comparing three methods: manual, selection by neural networks, and numerical pre-processing.

---

## CHAPTER 3: DEVELOPMENT OF NEURAL NETWORK SOFTWARE

---

### 3.1. Introduction

At the start of this work the author was working on the control software for the Slingsby ARM system which was still being developed. ARM posed a number of problems and interesting areas of research that were also applicable more widely to terrestrial robotics, such as the solution of manipulator kinematics with redundancy (since the complete ARM system had a 6 DOF manipulator on a 2 DOF deployment system) and real-time collision avoidance in a complex environment.

At the time neural network research was a burgeoning field and it was known that neural networks had been used successfully for solving problems which had shown themselves to be insoluble by other methods – either theoretically or because of practical time or computation constraints. It was therefore decided to investigate the field of neural networks and to look at the development of a neural network system to supplement the ARM control system for particular, appropriate tasks. This chapter will describe the history of neural networks and their application to the control of manipulators, ROVs and related topics. It will then describe the development by the author of new neural network software.

### 3.2. Neural Network History and Terminology

*Artificial Neural Networks* (ANNs) are systems that, unlike conventional computers, have a structure which, at some level, reflects what is known of the structure of the brain. The use of neural networks for solving problems is known as *neural computing*. Neural computing has been defined as the study of networks of adaptable nodes which, through a process of learning from task examples, store experiential knowledge and make it available for use [Aleksander 1990]. However, this is not an ideal definition as it does not take account of many of the simpler configurations of network that are possible. Many of these can be very useful for pattern recognition and memory retrieval

but are not adaptable and do not learn. Better terms are *Parallel Distributed Processing (PDP)* or *Connectionism* since these describe the low level arrangement of processing units but do not limit the implementation or configuration.

The fundamental element of the brain is known as a *neuron*. A diagrammatic view of a neuron is shown in Figure 3.1. It receives input signals from many other neurons, on branches known as *dendrites*. In response it may output a signal along its *axon* which also branches and can therefore pass the signal on to many other neurons. At the sites where the signals arrive there is a small gap between the incoming path and dendrite known as a *synapse*. This transfers the signal across the gap chemically.

While the basic operation of a neuron is very simple, it is also very powerful. The incoming signals may be inhibitory, tending to stop the neuron 'firing', or excitatory, tending to make it 'fire'. The incoming signals are combined and the result may cause the neuron to fire and so to affect further neurons. The effect of each synapse is variable and so the neuron can come to 'learn' to fire in response to certain combinations of input signals.

In 1943 a neurophysiologist, Warren McCulloch, and a logician, Walter Pitts, together proposed a simple model for neuron operation. In this model the effect of each synapse is represented by a weighting value applied to the incoming signal, the resulting weighted signals are summed, and if the result is beyond a certain threshold then the output signal is set to 'on', otherwise it is 'off'. This model forms the basis for the nodes in all neural networks and is known as the McCulloch and Pitts or *MCP* model.

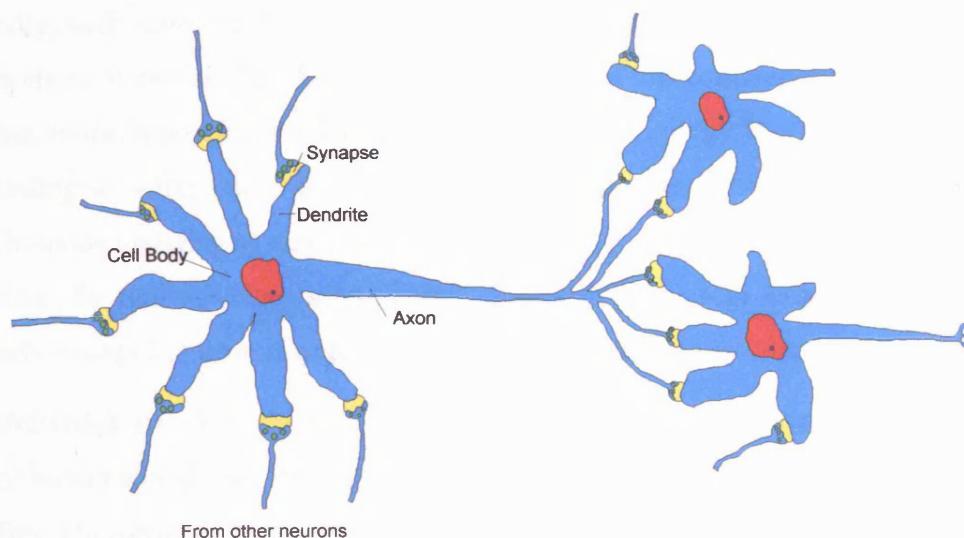


Figure 3.1 – Schematic Diagram of a Neuron

The most influential work on neural nets was undertaken in the mid-1960's by Frank Rosenblatt. He created a network where inputs underwent some simple pre-processing then each was multiplied by an adjustable weight. The resulting signals were summed and if the result was greater than a fixed threshold then an output signal was generated. The input signals were generated from visual information, the operation was electronic and the system was named the *Perceptron* [Rosenblatt 1962]. This system and developments of it are known as *pattern associators* and showed great success in pattern recognition and other areas.

In the late 1960's, however, Marvin Minsky and Seymour Papert demonstrated that there were a number of supposedly simple pattern recognition tasks that the Perceptron could not do. Not only that, but in a detailed theoretical treatise they demonstrated that there were certain so-called *hard learning* tasks that a Perceptron-type network could never achieve [Minsky 1969]. The study of neural networks suffered a major setback. Proponents demonstrated that many tasks impossible to Perceptrons could be achieved if the simple *input layer-output layer* structure was replaced by a more complicated *input layer-hidden layer-output layer* structure. However, no-one could demonstrate a means of teaching the hidden layer neurons to change their weights and therefore to learn.

It was not until the 1980's that interest in neural networks was rekindled. In 1982 John Hopfield published a very influential paper on the subject and drew attention to two properties of fully interconnected or *auto-associative* nets [Hopfield 1982]. Firstly, there will be stable states which will always be entered if the net is started in similar states. Secondly, such states can be created by changing the interconnection weights in the net. While these nets can be shown to be very useful as content-addressable memory systems, more important was Hopfield's concept of an energy level in the net, with the net tending to settle into a lowest-energy state. This was achieved by introducing an asynchronous updating system. In previous nets, inputs were simply summed together at one time. In this net, the input signals occurred at random times and so the nets regularly changed state and were therefore able to settle into low energy states.

One drawback of *Hopfield nets* was that sometimes the final states were only local energy minima and not the global minima required. This problem was solved by Geoffrey Hinton and Terry Sejnowski in 1986 [Hinton 1986]. They borrowed many ideas from thermodynamics and added a variable, 'temperature', to the system. This was high when the net started and was decreased over time. At high temperatures there was

a great deal of noise and therefore the system tended to jump out of local minima and, as the temperature decreased, the net eventually settled into the global minima, i.e. the correct solution. This technique is known, because of its origins, as *simulated annealing*. The resulting network is known as a *Boltzmann machine* and is one of a range of similar nets known as *constraint satisfaction* networks.

The other drawback of Hopfield nets, and those derived from them, was the requirement that they be fully interconnected, i.e. that every neuron is connected to every other neuron by a variable weight link. Clearly, this became a major overhead as the number of neurons increased. A major breakthrough was made by David Rumelhart, Geoffrey Hinton and Ronald Williams, also published in 1986, which has led to the current resurgence in interest in neural nets [Rumelhart 1986a]. They were part of an important group working on Parallel Distributed Processing, to use their term, and demonstrated a new method for training hidden layer neurons in a multi-layer network that was not fully interconnected. This uses two training steps. First there is a 'forward' phase during which the input is applied and allowed to propagate to the output. The error values of the output units are then calculated and compared to their required target values. During the second phase these errors are propagated 'backwards' and the weights are changed appropriately. This method is therefore known as *back propagation* and is the main technique currently used in neural network research.

### 3.3. Implementation

McCulloch and Pitts proposed their model of the neuron as something that could be built at the time using a summing amplifier and voltage comparator. The greatest difficulty in implementation was the requirement for a variable weight as the only practical method used a motorised variable resistor for each neuron and any reasonably sized machine would potentially require the operation of thousands of motors.

With the advent of digital computers, the majority of work on neural networks is now done in computers using software simulation of neurons. A neuron is simply a variable, though usually implemented as a member of an array. The weights are held in a two-dimensional array representing the potential connection between each neuron and every other neuron. The combination of inputs and the determination of an activation value for each neuron is done mathematically, usually with a transfer function.

Some current implementations requiring high speed extend this method by using extra hardware. This may include extra maths circuitry or the use of processors in parallel; in



either case the principles are the same. These techniques may change in the next few years as practical implementations of neural networks appear on silicon. Special purpose neural network components, particularly Integrated Circuits, promise great potential for the future, particularly in terms of speed. However, they have to overcome the difficulty of providing, at the upper limit, a connection on the silicon between each neuron and every other neuron.

### **3.4. Related Applications**

Neural networks are increasingly being applied in many research areas, particularly for pattern recognition, prediction and data fusion. For example, by the early 1990s they were being used for visual interpretation [Bishop 1993], recipe prediction [Bishop 1993], hybrid expert systems [Kasabov 1993] and other areas of artificial intelligence [Grant 1993]; since then they have become ubiquitous. In the area of control they have been used for process control, adaptive control [Colina-Morles 1993] and robot control.

Research into applying neural nets specifically to manipulators, ROVs and other offshore tasks will be described in more detail here.

#### **3.4.1. Control of Conventional Manipulators**

This is the control of conventional (i.e. non-redundant) robot manipulators using a hybrid controller incorporating a neural network. A number of problems in this area have been addressed. Some use the neural network to calculate an inverse kinematic solution to a world position problem. For example, Ahmad showed that it is possible to use a neural network to produce a 'first guess' to an inverse kinematic solution, the final solution is then obtained by an iterative process [Ahmad 1989]. This has been demonstrated on a PUMA 560. Yeung used a neural net to learn the inverse Jacobian matrix for a PUMA 560 to achieve inverse kinematic control [Yeung 1989]. Cohn proposed a method for optimising the kinematic training using techniques from Optimal Experiment Design (OED) [Cohn 1994].

Others have used the network to learn the moves required to approach or track a moving object, a task which is of great use on an assembly line. Van Der Smagt has used a net to learn how to move a simulated robot to an object [Van Der Smagt 1991]. The object was restricted in position to a horizontal plane and an overhead camera was used to determine the relative position of the object. Walter did similar work but actually controlled a PUMA 562 to move to the point indicated [Walter

1993]. In this system the object could be anywhere in the workspace of the robot, and its position was determined by the use of two cameras. Elsharkawi's work used some of the same ideas but with distance sensors mounted on the robot gripper in place of external vision systems [Elsharkawi 1992]. He used the PDP software developed by McClelland and Rumelhart [McClelland 1988] to build a back propagation network; the original system was implemented in simulation only. A Hopfield network has been used for a motion planning system for obstacle avoidance along complex paths, both for mobile robots and manipulator arms [Glasius 1995].

The dynamic control of a robot manipulator is usually very difficult to achieve by conventional techniques but neural nets have shown some success in this field. The CMAC (Cerebellar Model Articulation Controller) developed by Albus in the 1970's [Albus 1975] was used as the basis for a number of such systems. Miller *et al* used a CMAC system to learn the dynamics of a General Electric P-5 robot during high-speed movements, in order to closely follow a required trajectory [Miller 1990]. Graham used a modular architecture with one CMAC system per link and showed successful results in simulation for a two-link manipulator [Graham 1991]. Johnson implemented an Adaptive Model-Based Neural Network Controller (AMBNNC) with a multi-layer Perceptron architecture [Johnson 1990]. This was taught dynamic control under varying payload conditions and was shown to closely track a given trajectory even with payload changes while moving. It was experimentally evaluated on the third link of a PUMA 560.

A team at the University of Ghent successfully used a Kohonen net to calculate the inverse kinematics for a SCARA robot without an end effector (i.e. ignoring wrist orientation) [Declercq 1994]. The method was straightforward and easily adaptable to different robot configurations; however, it did not always converge to a solution, there was a minimum residual error (inversely proportional to the size of the network), and the system was susceptible to singularities. A theoretical examination of neural networks and similar systems for adaptive non-linear control showed that they could be used to solve conventionally difficult problems such as the dynamics of a multi-link robot arm [Sanner 1994].

More recently, a group at the University of Bonn successfully brought together a number of these ideas. They developed a robot control system for a Siemens Manutec robot with two neural networks. The first, the Neural Kinematics Network



(NKN) solved the inverse kinematics [Dapper 1997] while the second, the Neural Dynamics Network (NDN) was able to provide force control so that the robot could smoothly conduct contact tasks such as using a screwdriver [Dapper 1998]. A later enhancement to the NKN, known as Neural Trajectory Optimisation (NTO), used a modified Radial Basis Function (RBF) network to optimise the kinematic control and calculate not just the joint angles but also the velocities and accelerations required to conform to a defined trajectory such as along a surface [Maaß 1998].

#### **3.4.2. Control of Redundant Manipulators**

Here the aim is the inverse kinematic control of a redundant manipulator, a problem that is very difficult to solve conventionally. Ahmad extended his 'first guess' technique, mentioned above, to redundant manipulators. He used a three-layer Perceptron with back propagation, and the method was demonstrated on a three-link planar manipulator in simulation [Ahmad 1990]. Tanaka used a modular CMAC architecture with one CMAC system per link [Tanaka 1991]. The net was taught on forward kinematic data and was later able to select an inverse kinematic solution. The algorithm used was based on the pseudo-inverse of the Jacobian matrix. The system was again demonstrated in simulation on a three-link planar manipulator.

Work was done at the National Advanced Robotics Research Centre (NARRC) at Salford into the control of a seven-jointed manipulator utilising the redundancy to allow on-line collision avoidance [Boddy C. 1993]. This used a 'configuration control technique' and did not originally make use of neural networks; however, a neural network system was suggested for a theoretical implementation of a controller for a redundant manipulator with provision for collision avoidance [Morasso 1991] – it is not known if this materialised. A more advanced system implemented by Dissanayake used a neural network to control a sixteen-link manipulator moving in a plane while avoiding collisions with a number of objects and was successfully demonstrated in simulation [Dissanayake 1993].

#### **3.4.3. Offshore and Oceanographic Usage**

With regard to general offshore and oceanographic use, a back-propagation Artificial Neural Network Controller (ANNC) was demonstrated in simulation as being able to conduct automatic berthing of a ship [Djouani 1994]. Another back-propagation network was used for current prediction for shipping guidance [Wüst 1994].

Other applications included analysis of phytoplankton in seawater [Boddy L. 1994], visual classification of organic samples [Ellis 1994], filtering of additive noise [El-Hawary 1994], calculating wind speed and direction from scatterometer (microwave radar) data [Mejia 1994], 'meshing' a geographical area (mapping the distribution of hydrological data samples) [Sarzeaud 1994], adaptive signal processing in underwater acoustic communications [Gomes 1995], simulation of auditory neurons in dolphins [Dubrovsky 1994], sea-floor classification [Zerr 1994], and tomography (physical field parameter distribution) data processing [Kamenev 1995; Stephan 1995; Terre 1995].

#### **3.4.4. Use with ROVs and AUVs**

Looking more specifically at applications involving ROVs and AUVs, research by the control group at UCL was conducted into using neural networks to provide adaptive force control between a manipulator and a workpiece, with a view to eventually providing a means for adaptive weld following for an ROV manipulator [Wang 1994; Tisdall 1995; Tisdall 1997].

The 'AUV for Deep-sea Borehole Re-entry' project developed an RBF neural network for use as an AUV controller. It was able, in simulation, to produce the correct demands to control an AUV to re-enter a designated borehole [Feng 1994].

Theoretical work was conducted into using the SIGNAL process control language for programming neural networks for AUV control [Cherruel 1994]. Simulation results showed that a neural network controller for AUV depth outperformed a standard PID controller in the presence of noise or when the mass of the vehicle changed [Sutton 1994].

A multi-layer neural network was used as a directional controller on a test-bed AUV, taking the output from a flux gate compass and successfully controlling two thrusters in the horizontal plane to keep the AUV on a required heading [Guo 1995]. Less propitiously, a neural network was used for multi-sensor fusion on an ROV but was found to be insufficiently robust and was replaced by a conventional Kalman filter [Drolet 2000].

The DeepC project currently being promoted by the German Federal Ministry of Education and Research is aimed at developing and demonstrating an AUV with a decision-making system using fuzzy algorithms and neural networks. It has a core simulation element "for generating strategies of computerised learning and training

neural networks. Based on the operator-guided AUV operations in the virtual world, human behavioural patterns... are transferred to the vehicle" [Hornfeld 2002].

No references have been found in the literature specifically to using neural networks for selecting ROV docking locations.

### **3.5. Development of New Software**

At the outset of this work, and after an appraisal of available neural network systems, it was decided to create new neural network software from scratch for the following reasons:

- To provide software that could be directly linked in to ARM, or equivalent manipulator control software, for example as a library, rather than have to be run independently as a separate program.
- To investigate the main types of neural network and to determine the suitability of each for the tasks considered.
- To provide greater flexibility in the design of the network, and to provide greater opportunities for optimisation of those parts of the system most applicable to the task.
- As a learning method for the author, a technique to learn not just about the uses of neural networks but also to attempt to learn something about their internal programming.

Following on from the first requirement, the software system developed during this work, which was originally referred to as Windows Neural Networks (or "WinNeural"), was designed from the outset to run in combination with another Windows application such as ARM. It was therefore implemented in two parts. One half, the interface application, has libraries for interaction with the user, via dialog boxes and graphical windows. The other half has independent libraries for neural network calculations and file interpretation. These libraries are currently accessed directly by the interface application but could readily be used by a separate application such as ARM. These libraries could also be encapsulated into dynamic link libraries (DLLs), if required, in order to facilitate this process further.

The main types of neural network considered are given in the following table. The nomenclature is that used by McClelland and Rumelhart of the PDP Research Group

[McClelland 1988], but common alternative nomenclature is also given. The PDP nomenclature and terminology will be used in the rest of this work.

PDP Network Type	Subtypes	Alternative Name
Interactive Activation and Competition, IAC (Processing)		
Constraint Satisfaction, CS (Processing)	Schema Model	
	Boltzmann Machine	
	Harmony Model ( <i>Harmonium</i> )	
Pattern Associator, PA (Learning)	Hebb rule	
	Delta rule	Perceptron [Rosenblatt 1962]  LMS Associator
	Auto Associator  (Linear Auto-Associator, DMA Model)	Kohonen net  Brain-State-in-the-Box (BSB)
Back Propagation, BP (Learning)	Feed-forward	
	Cascaded feed-forward  Recurrent  Sequential  Competitive Learning	

Table 3.1 – Table of Neural Network Types

The yellow shaded boxes in Table 3.1 indicate the seven neural network types or subtypes that were fully implemented as part of this work. Their theory, operation and testing will be described in the next chapter. Some experimental implementation of Back Propagation extension types, such as cascaded feed-forward nets, was also conducted (and some of the NNW dialog boxes allow certain parameters for these types to be entered) but they were not proceeded with as they did not appear to be applicable to the tasks under consideration.

### **3.6. Neural Networks for Windows (NNW)**

After initial study of the theory and use of neural networks, particularly the work of the PDP research group, development of the WinNeural software began. It was based on PDP theory but coded entirely by the author. It was originally written in the C language using the Microsoft QuickC compiler and development environment. In order to provide an objective measure of its effectiveness and the accuracy of its results it was designed from the outset to be able to read files in the public format published by McClelland and Rumelhart of the PDP Research Group [McClelland 1988]. Hence the software developed during this work is able to read this format and run neural networks defined by them. It does not use any of the same code, and this can lead to slight discrepancies in the published results and those produced by this software; these discrepancies are examined further in *Section 4.10. Discussion of Deviations*.

The software underwent many years of development and changes, including a move from the C language to C++ (and from the QuickC to the Visual C++ compiler) and a change of name from WinNeural to Neural Networks for Windows (NNW). This development is covered in detail in Appendix B, with a full listing of all versions and their features.

The next sections describe the operation of this software, both the WinNeural and NNW versions. NNW has essentially all the features of, and is fully compatible with, WinNeural although it has an improved user interface. For convenience all versions will be referred to as “NNW”, and all screenshots below are taken from the latest versions, v1.1 and v1.2, running under Windows 2000.

## 3.7. NNW in Use

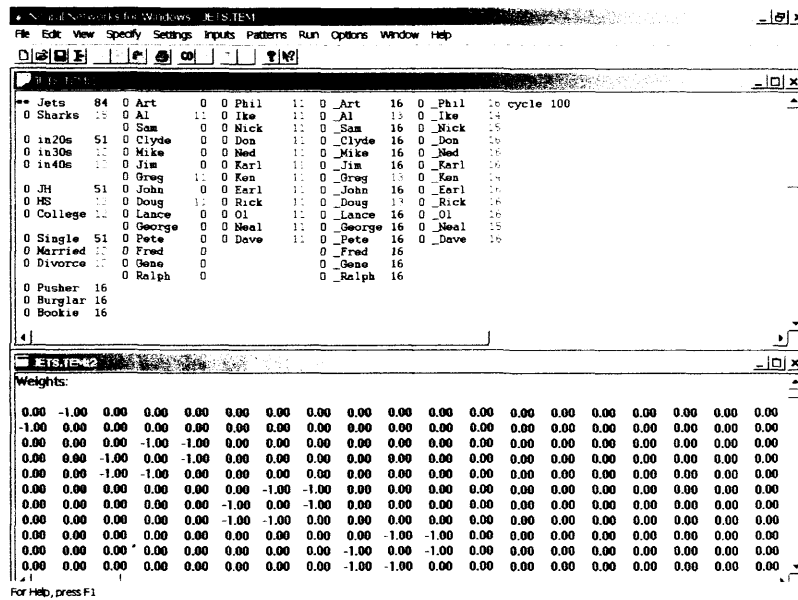


Figure 3.2 – NNW screen during an Interactive Activation and Competition problem

Figure 3.2 shows NNW during operation of a simple PDP IAC network. This example is of a 68 node neural network which is fully connected and is being used as content-addressable memory. Each node represents a person (in this example characters from the musical 'West Side Story') or a property of a person (e.g. their age, marital status or gang). Each person node is linked by a positive weight to the properties that are true for that person and by a negative weight to the properties that do not apply. The properties of this type of network using this example will be described in detail in *Section 4.2. Interactive Activation and Competition Network*.

Figure 3.3 – NNW screen during a Constraint Satisfaction problem

Figures 3.2 and 3.3 also demonstrate the NNW display and net contents windows. The display window shows the problem specified along with any inputs and results, and the contents window shows the internal matrix of weights between all nodes.

---

## CHAPTER 4: TESTING AND VERIFICATION OF NEURAL NETWORK SOFTWARE

---

### 4.1. Background

As each type of neural net was completed in NNW it was tested against published PDP files. The types developed, as detailed above, were the Interactive Activation and Competition (IAC) network, then the Constraint Satisfaction (CS) network, in its Schema, Boltzmann and Harmony variants, then the Pattern Associator (PA), and finally the Back Propagation (BP) network. The results of testing are given in this chapter; in addition some background theory and implementation notes are given. The networks generally are similar in construction, though increasing in complexity, and so, to save repetition, each network type is generally only described in terms of how it differs (in theory and use) from the type(s) described up to that point.

When analysing the displayed results it should be noted that because of display space constraints:

- All numbers are scaled up by a factor defined in the template (.tem) file. This is usually either 10, so 0.9 is displayed as 9, or 100, so 0.99 is displayed as 99 – the scale factor is usually clear from the context.
- 1.0 when scaled by a factor of 10 would take two digits to display whereas all other numbers only require one so it is represented as \* (similarly when scaled by a factor of 100 it is represented as \*\*).
- Negative activations are shown in red.

### 4.2. Interactive Activation and Competition Network

#### 4.2.1. IAC Theory

An IAC network consists of a collection of processing units organised into some number of competitive pools. There are excitatory connections among units in different pools and inhibitory connections among units within the same pool. The



excitatory connections between pools are generally bi-directional, thereby making the process *interactive* as processing in each pool both influences and is influenced by processing in other pools. Within a pool, the inhibitory connections are usually assumed to run from each unit in the pool to every other unit in the pool; there is therefore a kind of *competition* among the units so those that receive strongest activation tend to drive down the activation of the other units. The general arrangement of an IAC network is shown in Figure 4.1 where the units are shown in green, inhibitory connections are in red and excitatory connections are in blue. Connections that are uni-directional are shown with an arrowhead; otherwise connections are bi-directional.

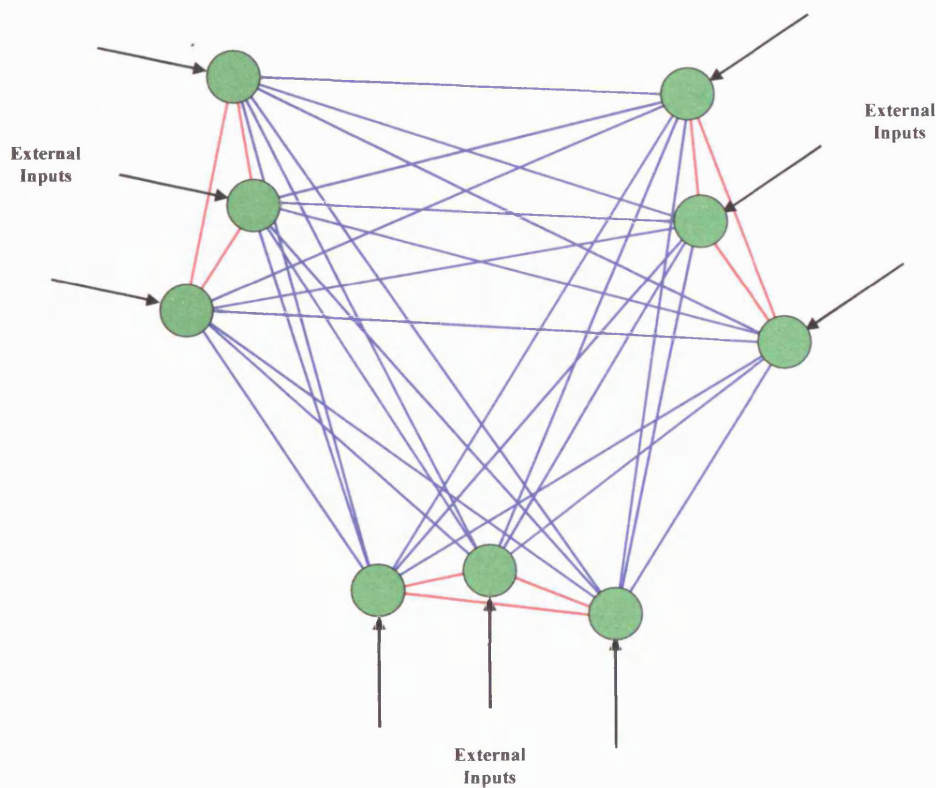


Figure 4.1 – Diagrammatic representation of an IAC Network

The units take on continuous activation values between a maximum value, *max* (default 1.0) and a minimum value, *min* (default -1.0). Their output to other units is the activation minus a threshold value (fixed for IAC networks at 0.0). The activations of the units evolve gradually over time, which is modelled as a sequence of small, discrete steps, known as *cycles*, based on a function that takes into account both the current activation of the unit and the net input to the unit from other units or from outside the network.

The net input to unit  $i$  is the sum of the influences of all the other units in the network plus any external input. The influence of some other unit  $j$  is the product of that unit's output,  $output_j$ , times the strength or weight,  $w_{ij}$ , of the connection to unit  $i$  from unit  $j$ :

$$net_i = \sum_j w_{ij} output_j + extinput_i \quad - \text{Eq. 4.1}$$

where  $output_j$  is the activation of unit  $j$  if positive, otherwise it is zero. The resulting change in activation is given by:

$$\Delta a_i = (max - a_i)net_i - decay(a_i - rest) \quad \text{when } net_i > 0 \quad - \text{Eq. 4.2}$$

$$\Delta a_i = (a_i - min)net_i - decay(a_i - rest) \quad \text{when } net_i \leq 0 \quad - \text{Eq. 4.3}$$

where  $rest$  is the resting activation level to which activations tend to settle in the absence of external input, and  $decay$  determines the strength of the tendency to return to resting level – in general  $0 \leq decay \leq 1$ , and  $min \leq rest \leq 0$ .

In the general case, three further parameters are considered,  $estr$ ,  $alpha$  and  $gamma$ , which can be used to scale the strength of external input, internal excitatory input and internal inhibitory input respectively. The net input is then the external input scaled by  $estr$ , plus the excitatory input from other units scaled by  $alpha$ , plus the inhibitory input from other units scaled by  $gamma$ .

Note that the net input to a unit changes as the unit and other units in the same pool simultaneously respond to their net inputs; one effect of this is to amplify differences in the net inputs of units. The end result is a phenomenon known as the "rich get richer" effect – units with slight initial advantages, in terms of their external inputs, amplify this advantage over their competitors.

#### 4.2.2. IAC Implementation

For the initial implementation of NNW, two generally applicable data structures were created: `VectorN`, a vector array of  $n$  elements, with each value held in the list as a `double` type:

```
typedef struct {
    int      nElements;
    HANDLE   hElementList;
} VECTORN;
```

where `hElementList` is created as follows:

## Chapter 4: Testing and Verification of Neural Network Software

```
hElementList = GlobalAlloc( GMEM_MOVEABLE,  
                             pVectorn->nElements * sizeof( double ));
```

and `MatrixN`, a vector array of  $n \times \text{VectorN}$  vectors:

```
typedef struct {  
    int      nVectorns;  
    HANDLE   hVectornList;  
} MATRIXN;
```

When a network is created, e.g. in response to reading in network definition files, all parameters are initially set to default values, then a series of `VectorNs` is created to hold the values of activation, excitation, inhibition, external input, etc. for all units in the net, plus a `MatrixN` to hold the array of weights between units. Any parameter values specified in the network files are then used to override the defaults.

The rule for calculating the change in activation described above is that proposed by the PDP Research Group. Another rule was proposed by Grossberg, one of the early researchers in this area, of the following form:

$$\Delta a_i = (max - a_i)e - (a_i - min)i - decay(a_i - rest) \quad - \text{Eq. 4.4}$$

where  $e$  is the excitatory input, which drives the activation of the unit up towards the maximum, and  $i$  is the inhibitory input, which drives the activation back down towards the minimum. Both rules have been implemented in NNW and are selected from the `Options` menu (the default is `PDP Group Update`).

A network is created by selecting `File:New` in NNW, choosing a network type, then loading in suitable definition files (typically a strengths file, `.str`, and a template file, `.tem`, which may themselves load further network definition, `.net`, and weights, `.wts`, files). When the network is set cycling (by selecting `Go` on the `Run` menu), at each cycle the new activation of each unit is calculated, based on the net input and the existing activation value, and then the screen display is updated. This continues until the total number of cycles set is reached, after which it can be set cycling again. Alternatively, (using the `Reset` command on the `Run` menu) it can be reset back to its starting state – this resets the unit activations to their starting levels, the current cycle number to zero, and all other parameters to their starting values, before refreshing the display.

Discussion of a scenario representing members of two gangs known as "Jets" and "Sharks" (presumably inspired by *West Side Story*) has been published in some detail [McClelland 1986] as well as example tests and results [McClelland 1988].

This scenario is used here to illustrate the main features of an IAC network and the results of testing with NNW.

#### 4.2.3. Retrieving the Attributes of an Activated Node

Providing a high input to an individual's name and then cycling leads to all of the attributes associated with the individual becoming activated (i.e. gang, age, education, marital status and occupation). NNW produced a set of results (Figure 4.2 shows the state after 100 cycles) essentially identical to the published PDP results<sup>1</sup> – activating Ken to maximum has led to high activation of Sharks, in20s, High School, Single and Burglar respectively<sup>2</sup>.

0 Jets	13	0 Art	14	0 Phil	14	0 _Art	14	0 _Phil	14	cycle 100
0 Sharks	51	0 Al	14	0 Ike	14	0 _Al	15	0 _Ike	12	
		0 Sam	14	0 Nick	14	0 _Sam	13	0 _Nick	24	
0 in20s	38	0 Clyde	14	0 Don	14	0 _Clyde	14	0 _Don	13	
0 in30s	1	0 Mike	14	0 Ned	14	0 _Mike	14	0 _Ned	14	
0 in40s	13	0 Jim	14	0 Karl	14	0 _Jim	14	0 _Karl	12	
		0 Greg	14	** Ken	81	0 _Greg	13	0 _Ken	68	
0 JH	13	0 John	14	0 Earl	14	0 _John	14	0 _Earl	4	
0 HS	52	0 Doug	14	0 Rick	14	0 _Doug	12	0 _Rick	4	
0 College	13	0 Lance	14	0 Ol	14	0 _Lance	14	0 _Ol	14	
		0 George	14	0 Neal	14	0 _George	14	0 _Neal	24	
0 Single	51	0 Pete	14	0 Dave	14	0 _Pete	4	0 _Dave	12	
0 Married	13	0 Fred	14			0 _Fred	4			
0 Divorce	13	0 Gene	14			0 _Gene	13			
		0 Ralph	14			0 _Ralph	14			
0 Pusher	11									
0 Burglar	38									
0 Bookie	11									

Figure 4.2 – IAC Network retrieving the attributes of an activated node

#### 4.2.4. Retrieval From a Partial Description

Providing a high input to the attributes that uniquely identify an individual and then cycling leads to the individual's name becoming activated. NNW produced a set of results (Figure 4.3 shows the state after 100 cycles) essentially identical to the PDP results – activating Sharks and in20s has led to high activation of Ken and \_Ken.

<sup>1</sup> Taking into account that when directly comparing results the PDP and NNW activation values are displayed very slightly differently (though they may be identical internally) because the PDP system truncates to two significant figures where NNW rounds to two significant figures.

<sup>2</sup> It has also led, naturally, to high activation of the \_Ken node which is an instance node (which can be considered to be a form of output node). The instance nodes are hidden nodes in the network and cannot be directly accessed by the user; instead they take their activations directly from the inputs they receive from other nodes.

```

0 Jets      12  0 Art      13  0 Phil     13  0 _Art     15  0 _Phil    15  cycle 100
** Sharks   82  0 Al       13  0 Ike      13  0 _Al      16  0 _Ike     13
0           0 Sam      13  0 Nick     10  0 _Sam     13  0 _Nick    31
** in20s    82  0 Clyde   13  0 Don      13  0 _Clyde   15  0 _Don     14
0 in30s     12  0 Mike    13  0 Ned      13  0 _Mike    15  0 _Ned     15
0 in40s     15  0 Jim     13  0 Karl     13  0 _Jim     14  0 _Karl    13
0           0 Greg     13  0 Ken      34  0 _Greg    13  0 _Ken     66
0 JH         14  0 John    13  0 Earl     13  0 _John    14  0 _Earl    11
0 HS         62  0 Doug    13  0 Rick     13  0 _Doug    14  0 _Rick    11
0 College   14  0 Lance   13  0 Ol       13  0 _Lance   14  0 _Ol      15
0           0 George   13  0 Neal     10  0 _George  14  0 _Neal    31
0 Single    62  0 Pete    10  0 Dave     13  0 _Pete    31  0 _Dave    13
0 Married   14  0 Fred     0  0           0  0 _Fred    31
0 Divorce   14  0 Gene     0  0           0  0 _Gene    13
0           0 Ralph    13  0 _Ralph   15
0 Pusher    11
0 Burglar   25
0 Bookie    11

```

Figure 4.3 – IAC Network retrieving a node from a partial description of its attributes

#### 4.2.5. Graceful Degradation

IAC networks are able to function even in the presence of erroneous information. It has been shown above that providing a high input to the attributes that identify an individual and then cycling leads to the individual's name being activated; however, if sufficient correct attributes are activated then the individual's name will become activated even if one or more of the attributes are incorrect.

In both cases NNW produced a set of results essentially identical to the PDP results – activating Sharks, in20s, High School, Single and Burglar has led to high activation of Ken and \_Ken (0.36 and 0.72); Figure 4.4 shows the state after 100 cycles. However, replacing High School with Junior High has led also led to an activation of Ken and \_Ken but at lower values (0.27 and 0.59); Figure 4.5 shows the state after 100 cycles.

```

0 Jets      12  0 Art      13  0 Phil     13  0 _Art     16  0 _Phil    16  cycle 100
** Sharks   84  0 Al       13  0 Ike      13  0 _Al      16  0 _Ike     14
0           0 Sam      13  0 Nick     11  0 _Sam     14  0 _Nick    27
** in20s    82  0 Clyde   13  0 Don      13  0 _Clyde   16  0 _Don     14
0 in30s     10  0 Mike    13  0 Ned      13  0 _Mike    16  0 _Ned     16
0 in40s     14  0 Jim     13  0 Karl     13  0 _Jim     14  0 _Karl    14
0           0 Greg     13  0 Ken      36  0 _Greg    14  0 _Ken     72
0 JH         15  0 John    13  0 Earl     11  0 _John    14  0 _Earl    26
** HS         85  0 Doug    13  0 Rick     11  0 _Doug    14  0 _Rick    26
0 College   15  0 Lance   13  0 Ol       13  0 _Lance   14  0 _Ol      16
0           0 George   13  0 Neal     11  0 _George  14  0 _Neal    27
** Single    84  0 Pete    11  0 Dave     13  0 _Pete    26  0 _Dave    14
0 Married   14  0 Fred     0  0           0  0 _Fred    26
0 Divorce   14  0 Gene     0  0           0  0 _Gene    14
0           0 Ralph    13  0 _Ralph   16
0 Pusher    12
** Burglar   82
0 Bookie    12

```

Figure 4.4 – IAC Network activating an individual given his attributes correctly

```

0 Jets      47  0 Art      13  0 Phil     13  0 _Art     13  0 _Phil    16  cycle 100
** Sharks   79  0 Al       13  0 Ike      13  0 _Al      12  0 _Ike     10
      0 Sam      13  0 Nick     13  0 _Sam     12  0 _Nick    15
** in20s    85  0 Clyde   13  0 Don      13  0 _Clyde   13  0 _Don     14
0 in30s     15  0 Mike    13  0 Ned      13  0 _Mike    13  0 _Ned     16
0 in40s     15  0 Jim      2  0 Karl     13  0 _Jim     47  0 _Karl    16
      0 Greg     13  0 Ken      27  0 _Greg    15  0 _Ken     59
** JH       84  0 John     2  0 Earl     13  0 _John    47  0 _Earl    14
0 HS        12  0 Doug     13  0 Rick     13  0 _Doug    15  0 _Rick    14
0 College   15  0 Lance     2  0 Ol       13  0 _Lance   47  0 _Ol      16
      0 George    2  0 Neal     13  0 _George  47  0 _Neal    15
** Single   80  0 Pete     13  0 Dave     13  0 _Pete    12  0 _Dave    16
0 Married    1  0 Fred     13  0 _Fred    12
0 Divorce    1  0 Gene     13  0 _Gene    12
      0 Ralph    13  0 _Ralph   13
0 Pusher    15
** Burglar   85
0 Bookie    15

```

Figure 4.5 – IAC Network activating an individual given all but one of his attributes correctly

#### 4.2.6. Default Assignment

IAC networks are able to fill in missing data by giving 'plausible guesses' as to what they might be based on the other information it knows. It has been shown above that providing a high input to an individual's name and then cycling leads to all of the attributes associated with the individual become activated; however if we remove the information about a certain attribute (by setting the weights for the connection between the individual and the attribute to zero) and rerun the network then the system will attempt to fill in the missing information. As before NNW produced a set of results essentially identical to the PDP results – activating Lance to maximum leads to high activation of Jets, in20s, Junior High, Married and, in particular, Burglar has an activation of 0.67; Figure 4.6 shows the state after 100 cycles. However, removing the weight connections for Burglar (setting Lance-Burglar to zero and Burglar-Lance to 0) and rerunning activates the same nodes, but now Burglar has a lower activation of 0.57; Figure 4.7 shows the state after 100 cycles.

```

0 Jets      67 0 Art      14 0 Phil     14 0 _Art     15 0 _Phil    16 cycle 100
0 Sharks    14 0 Al       13 0 Ike      14 0 _Al      29 0 _Ike     16
              0 Sam      14 0 Nick     14 0 _Sam     15 0 _Nick    17
0 in20s     63 0 Clyde   14 0 Don      14 0 _Clyde   15 0 _Don     15
0 in30s     12 0 Mike    14 0 Ned      14 0 _Mike    15 0 _Ned     16
0 in40s     14 0 Jim     13 0 Karl     14 0 _Jim     39 0 _Karl    16
              0 Greg    14 0 Ken      14 0 _Greg    13 0 _Ken     15
0 JH        67 0 John    12 0 Earl     14 0 _John    55 0 _Earl    15
0 HS        14 0 Doug    14 0 Rick     14 0 _Doug    16 0 _Rick    16
0 College   14 ** Lance   81 0 Ol       14 0 _Lance   67 0 _Ol      16
              0 George  13 0 Neal     14 0 _George  39 0 _Neal    17
0 Single    14 0 Pete    14 0 Dave     14 0 _Pete    15 0 _Dave    17
0 Married   55 0 Fred    14              0 _Fred    15
0 Divorce   9  0 Gene    14              0 _Gene    15
              0 Ralph   14              0 _Ralph   15

0 Pusher    14
0 Burglar   67
0 Bookie    14

```

Figure 4.6 – IAC Network retrieving the correct attributes of an activated node given all information

```

0 Jets      65 0 Art      14 0 Phil     14 0 _Art     15 0 _Phil    16 cycle 100
0 Sharks    14 0 Al       13 0 Ike      14 0 _Al      30 0 _Ike     16
              0 Sam      14 0 Nick     14 0 _Sam     15 0 _Nick    17
0 in20s     61 0 Clyde   14 0 Don      14 0 _Clyde   15 0 _Don     15
0 in30s     12 0 Mike    14 0 Ned      14 0 _Mike    15 0 _Ned     16
0 in40s     14 0 Jim     13 0 Karl     14 0 _Jim     36 0 _Karl    16
              0 Greg    14 0 Ken      14 0 _Greg    13 0 _Ken     15
0 JH        65 0 John    12 0 Earl     14 0 _John    55 0 _Earl    15
0 HS        14 0 Doug    14 0 Rick     14 0 _Doug    16 0 _Rick    16
0 College   14 ** Lance   80 0 Ol       14 0 _Lance   59 0 _Ol      16
              0 George  13 0 Neal     14 0 _George  36 0 _Neal    17
0 Single    14 0 Pete    14 0 Dave     14 0 _Pete    15 0 _Dave    17
0 Married   54 0 Fred    14              0 _Fred    15
0 Divorce   5  0 Gene    14              0 _Gene    15
              0 Ralph   14              0 _Ralph   15

0 Pusher    14
0 Burglar   57
0 Bookie    14

```

Figure 4.7 – IAC Network retrieving the correct attributes of an activated node given partial information

#### 4.2.7. Spontaneous Generalisation

IAC networks are able to retrieve appropriate generalisations over groups of individuals, i.e. to provide typical attributes of a group such as members of a gang, or those with a particular education or occupation. Providing a high input to a group node and then cycling leads to typical attributes associated with the group becoming activated. NNW appeared initially to produce a set of results that were essentially identical to the PDP results – activating *Jets* to maximum leads to high activation of *in20s*, *Junior High*, and *Single*. Also, all three occupations are equally activated (value 0.16) - Figure 4.8 shows the state after 100 cycles.

```

** Jets      84  0 Art      0  0 Phil    11  0 _Art    16  0 _Phil   16 cycle 100
0 Sharks    15  0 Al       11  0 Ike     11  0 _Al     13  0 _Ike    14
      0 Sam      0  0 Nick    11  0 _Sam    16  0 _Nick   15
0 in20s     51  0 Clyde   0  0 Don     11  0 _Clyde  16  0 _Don    16
0 in30s     12  0 Mike    0  0 Ned     11  0 _Mike   16  0 _Ned    16
0 in40s     12  0 Jim     0  0 Karl    11  0 _Jim    16  0 _Karl   16
      0 Greg    11  0 Ken     11  0 _Greg   13  0 _Ken    14
0 JH        51  0 John    0  0 Earl    11  0 _John   16  0 _Earl   16
0 HS        12  0 Doug    11  0 Rick    11  0 _Doug   13  0 _Rick   16
0 College   12  0 Lance   0  0 Ol      11  0 _Lance  16  0 _Ol     16
      0 George  0  0 Neal    11  0 _George 16  0 _Neal   15
0 Single    51  0 Pete    0  0 Dave    11  0 _Pete   16  0 _Dave   16
0 Married   12  0 Fred    0          0 _Fred   16
0 Divorce   12  0 Gene    0          0 _Gene   16
      0 Ralph  0          0 _Ralph  16
0 Pusher    16
0 Burglar   16
0 Bookie    16

```

Figure 4.8 – IAC Network retrieving typical attributes for an activated group node

However, at this point an interesting discrepancy was found. When conducting testing on any of the IAC scenarios detailed, both PDP and NNW systems were run to many cycles (typically between 200 and 500) but invariably it was found that the system had settled before reaching 100 cycles – except in this case. Here it was found that the PDP system appeared to settle after 40 cycles, but then suddenly the values for the occupation nodes started to change again from 110 cycles with the Pusher node eventually becoming highly activated.

It was decided to look at this phenomenon in more detail, and both PDP and NNW results were compared after every 10 cycles (and every 5 cycles near the start where the activations changed fastest) – in particular looking at the activation of the Pusher node ("PDP Pusher" and "NNW Pusher" respectively), and the overall activation of the Jets gang (i.e. the average of the activations of the instance nodes of the individuals in the gang – "PDP Gang" and "NNW Gang" respectively). The results are plotted in Figure 4.9. First of all it can be seen that up to 100 cycles the plots are essentially identical (the only variation is actually due to the difference between the truncated PDP activation values and the – on average slightly higher – rounded NNW activation values). However, beyond 100 cycles where the NNW activations remain constant, the PDP results change significantly – the PDP Pusher value increases dramatically until it settles at a value of 0.62, while the PDP Gang values falls away until it settles at 0.05.



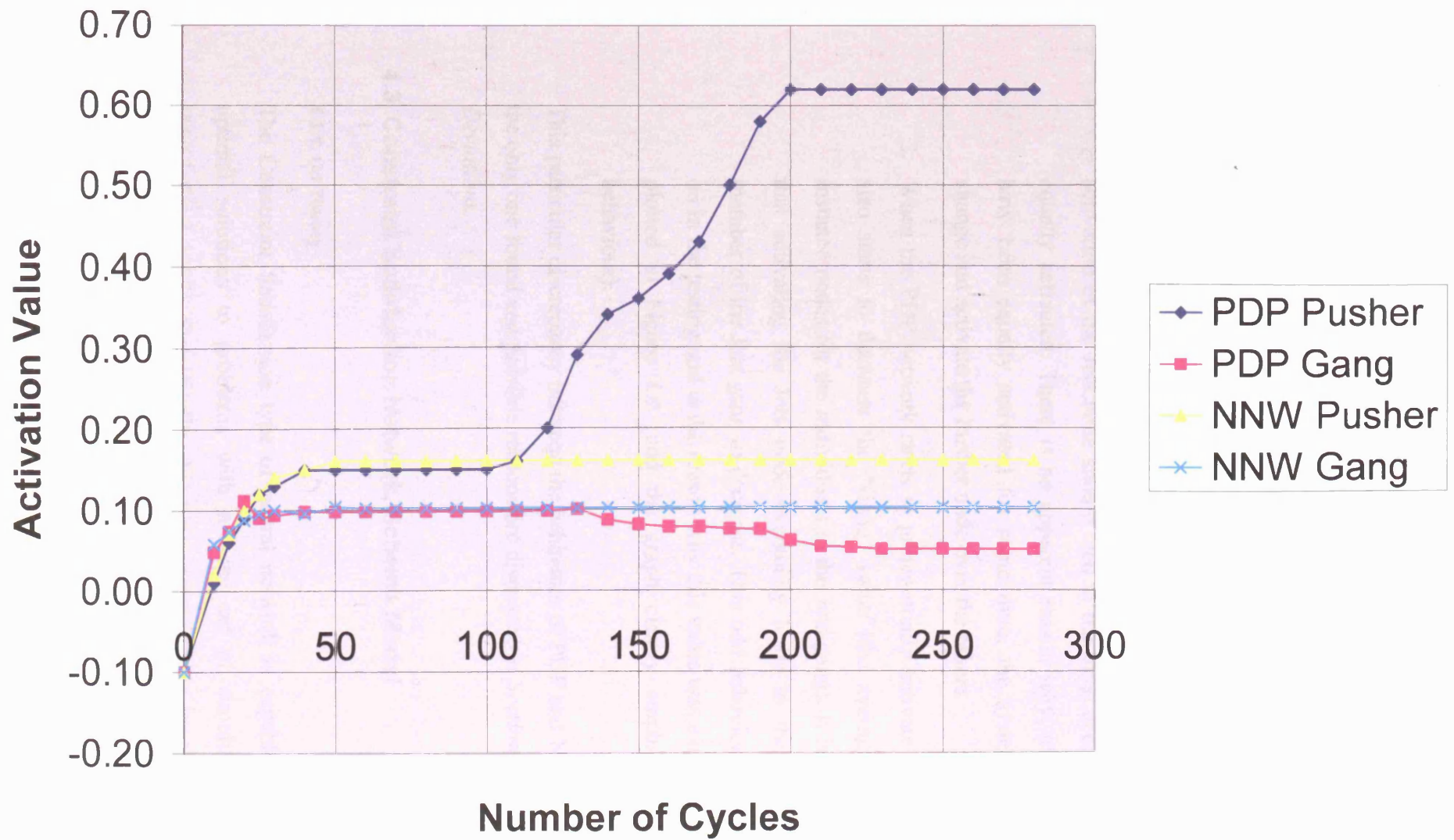


Figure 4.9 – Comparison of activation values changing over time for PDP versus NNW

This phenomenon is not covered or discussed in the PDP literature at all, and appears to be a new result. On close examination the most reasonable conclusion appears to be that the PDP software has produced an invalid result, for three reasons:

1. Although some neural network types can exhibit changeable and unstable trends, it is atypical for an IAC network
2. The 15 members of the Jets gang are equally divided into 5 Pushers, 5 Burglars and 5 Bookies so one would expect, all other inputs being zero, that activation of the Jets node should lead to the three occupation nodes being equally activated. There is no apparent reason why, after the three nodes have been equally activated for some time, the system should suddenly change and activate the Pusher node over the others.
3. When the PDP network starts to preferentially activate the Pusher node, it also starts to decrease the "Gang" value (the average activation of the instance nodes for the individuals in the Jets gang). It is clearly unexpected that activating the Jets node eventually leads to the activations of the member of the Jets gang to decrease. This odd behaviour was noticed early on in the testing and is the reason why this value was examined in detail and plotted in Figure 4.9 (and the graph clearly emphasises this aberrant behaviour).

This particular discrepancy between the behaviour of PDP and NNW results was not the only one found and possible reasons are discussed in *Section 4.10. Discussion of Deviations*.

### **4.3. Constraint Satisfaction Network, Schema Model**

#### **4.3.1. CS Theory**

The Constraint Satisfaction type of neural network is capable of "finding near-optimal solutions to problems with a large set of simultaneous constraints" [McClelland 1988]. Specifically, this type of network is capable of solving 'best match' problems – involving the simultaneous satisfaction of a very large number of constraints, even though there may be no perfect solution in which all of the constraints are completely satisfied (in which case the solution would involve the satisfaction of as many constraints as possible). Furthermore, each constraint may

have an importance value associated with it (reflected by the strength of its connection), in which case the solution involves the simultaneous satisfaction of as many of the most important of these constraints as possible.

In the network each unit or neuron represents a hypothesis and each connection represents a constraint among hypotheses. Units may receive external input which represents direct evidence for certain hypotheses. It can either have a numerical value, called *bias*, which acts to turn the unit on in the absence of other evidence and represents the probability that it is true. Alternatively, the input can be "clamped" which means that this particular unit must be on if the input is positive or must be off if the input is negative (this is set in NNW using the `Clamping On` command on the `Options` menu).

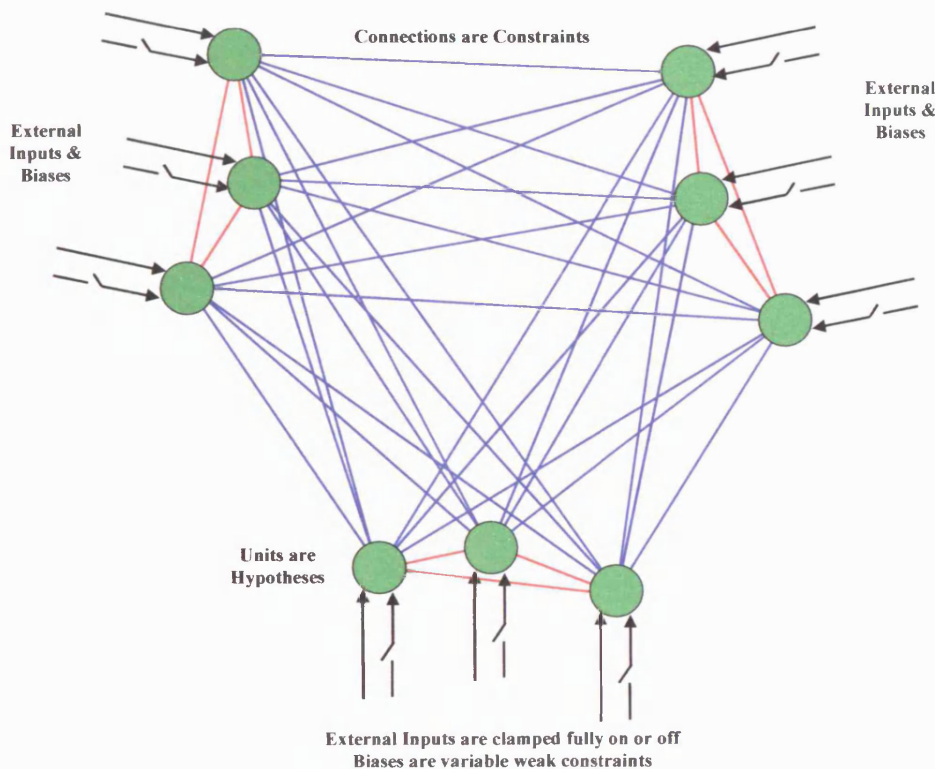


Figure 4.10 – Diagrammatic representation of a CS Network

The degree to which the desired constraints are satisfied is the goodness of fit ('goodness') and this has three elements. Firstly, it depends on the extent to which each unit satisfies the constraints imposed on it by other units. Secondly, the *a priori* strength of the hypothesis is captured by adding the bias to the goodness measure. Finally, the goodness for a hypothesis when direct evidence is available is given by the product of the input value times the activation value of the unit.

Thus the goodness for unit  $i$  is given by:

$$goodness_i = \sum_j w_{ij} a_i a_j + bias_i a_i + input_i a_i \quad - Eq. 4.5$$

and for the whole system is:

$$goodness = \sum_{ij} w_{ij} a_i a_j + \sum_i bias_i a_i + \sum_i input_i a_i \quad - Eq. 4.6$$

Thus the problem is solved when a set of activation values is found that maximises this function. This is straightforward to achieve in a CS network where the weights are required to be symmetric, i.e.  $w_{ij} = w_{ji}$ , so that:

$$goodness_i = net_i a_i \quad - Eq. 4.7$$

$$\text{where } net_i = \sum_j w_{ij} a_j + input_i + bias_i$$

= net input into a unit

Therefore the net input into a unit provides the unit with information as to its contribution to the goodness of the entire solution. Any particular unit can always behave so as to increase its contribution to the overall goodness if, whenever its net input is positive, the unit moves its activation towards its maximum value, and whenever its net input is negative, it moves its activation towards its minimum. Since the global goodness is simply the sum of the individual goodnesses, a whole network of units behaving in this way will always increase the global goodness measure.

#### 4.3.2. CS Implementation

During the implementation of CS networks in NNW, it was found that the previous data structure (see *Section 4.2.2. IAC Implementation*) was becoming increasingly unwieldy, holding large numbers of `VectorN` structures, each one holding a list with a value for each unit (so there were `VectorNs` for activation, excitation, bias, external input, and so on, plus additional `VectorNs` to hold the information scaling and displaying each value).

Using new data types, `STRINGVAR` and `DOUBLEVAR` (which each held the data value and information on how to scale it and where to display it), a new data structure was implemented consisting of a single list of a new data type `NEURON` which held all the data for a particular unit:

```
typedef struct {
    char    szText[MAX_NAME];
    int     nXPos;
    int     nYPos;
    int     nSpaces;
    int     nScale;
} STRINGVAR;

typedef struct {
    double  dValue;
    int     nXPos;
    int     nYPos;
} DOUBLEVAR;

typedef struct {
    STRINGVAR    svName;
    DOUBLEVAR    dvActivation, dvExcitation, dvInhibition;
    DOUBLEVAR    dvTotalInput, dvExternalInput, dvBias, dvBiasFlag;
} NEURON;
```

This change significantly simplified and improved the data handling within NNW and was used throughout the rest of the development.

#### 4.3.3. Schema Model Theory

The simplest type of CS network uses the Schema Model which has the following characteristics:

- Its units can take on any value between their minimum and maximum values.
- Units may not connect to themselves, i.e.  $w_{ii} = 0$ .
- Update is asynchronous, i.e. units are chosen to be updated sequentially in random order.

When chosen, the net input to the unit is computed and the new activation of the unit is calculated using the following rule:

$$a_i(t+1) = a_i(t) + net_i(1 - a_i(t)) \quad \text{when } net_i > 0 \quad - \text{Eq. 4.8}$$

$$a_i(t+1) = a_i(t) + net_i a_i(t) \quad \text{when } net_i < 0 \quad - \text{Eq. 4.9}$$

For a CS network the net input comes from the addition of three sources: the unit's neighbours, its bias and its external inputs, so:

$$net_i = istr(\sum_j w_{ij} a_j + bias_i) + estr(input_i) \quad - \text{Eq. 4.10}$$

where *istr* is a new parameter, the Internal Input Strength, that allows the scaling of the relative contribution of the input from internal sources and is analogous to *estr* for external input (see Section 4.2.1. *IAC Theory*).

Since the units to be updated are chosen in random order, another new parameter is required, the random number seed, which is used for initialising the system's random number generator. This also provides a new way of restarting the processing of a network. The existing `Reset` command (see Section 4.2.2. *IAC Implementation*) now restarts the network with the same random number seed so that it can be run again and the same activations observed. However, a new command (`New Start` on the `Run` menu) restarts the network with a new random number seed, units are updated in a different order and so different activations can be observed.

#### 4.3.4. Necker Cube (Schema Model)

In the Necker Cube problem there are two possible interpretations of a wire-frame isometric representation of a cube, either it is facing down and to the left or it is facing up and to the right. It has been demonstrated that a CS network is able to capture the fact that there are exactly two good interpretations of a Necker cube [Rumelhart 1986b]. A correct right-hand interpretation in NNW is shown in Figure 4.11. The results of a test to compare PDP with NNW, with 100 runs, each of 50 cycles, are given in Table 4.1. In most cases both systems usually settled into a state that was a correct interpretation of the cube, and had a maximum goodness value; sometimes, however, they settled into invalid interpretations (with some nodes activated that represented a different interpretation from the other activated nodes) with a lower goodness value.

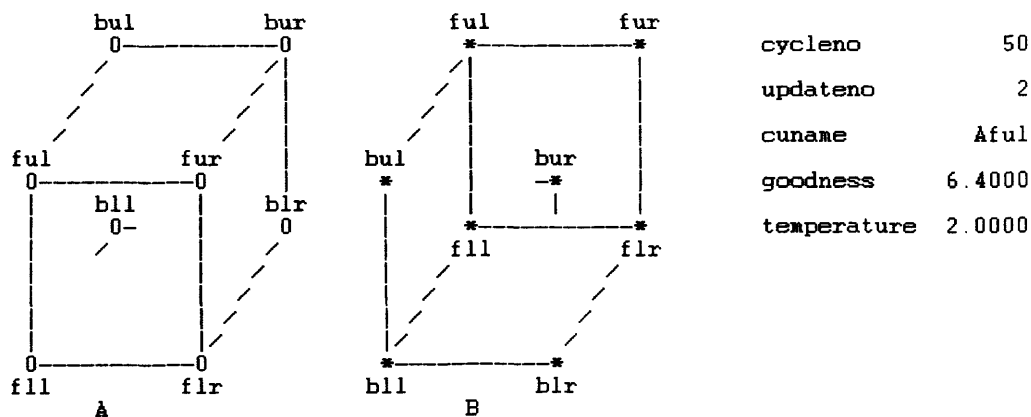


Figure 4.11 – CS Network settling on a right-hand Necker Cube interpretation

When compared, NNW produced a set of results that are very similar to the PDP results. However, there does appear to be a slight trend for PDP to reach global

maxima more often than NNW, i.e. for NNW to get caught in local maxima more often than PDP. Possible reasons for this are discussed in *Section 4.10. Discussion of Deviations*.

Necker Cube Interpretation	PDP Results			NNW Results		
	Goodness/ <i>istr</i>	Occurrences	Total	Goodness/ <i>istr</i>	Occurrences	Total
Left, down	16	45	76 valid	16	38	61 valid
Right, up	16	31		16	23	
Outside edges	12	5	24 invalid	4.8	8	39 invalid
Inside edges	12	2		12	9	
Top left, bottom right	12	2		12	5	
Top right, bottom left	12	4		12	2	
$\frac{3}{4}$ right, $\frac{1}{4}$ left	12	6		12	8	
$\frac{3}{4}$ left, $\frac{1}{4}$ right	12	5		12	7	
TOTAL		100	100		100	100

Table 4.1 – Comparison of Necker Cube results, input strength (*istr*) = 0.4

There is a parameter in the Schema Model for the input strength to each unit (*istr*) that multiplies the weights and biases and that, in effect, determines the rate of activation flow within the model. The probability of finding a local maximum depends on the value of this parameter, and so, in order to reduce the occurrences of local maxima, the Necker Cube problem was retried with various values of *istr* lower than that originally used (0.4). Lowering *istr*, of course, slowed down the rate of change within the network and so therefore required an increase in the number of cycles before the network settled. It was found that about 0.01 was a reasonable compromise for *istr* (getting the correct interpretation at least 90% of the time, and just managing it within 500 cycles) and the results for the two systems with this value are given in Table 4.2.

Necker Cube Interpretation	PDP Results			NNW Results		
	Goodness/ <i>istr</i>	Occurrences	Total	Goodness/ <i>istr</i>	Occurrences	Total
Left, down	16	38	94 valid	16	51	90 valid
Right, up	16	56		16	39	
Outside edges			6 invalid			10 invalid
Inside edges						
Top left, bottom right	12	3		12	1	
Top right, bottom left						
$\frac{3}{4}$ right, $\frac{1}{4}$ left				12	3	
$\frac{3}{4}$ left, $\frac{1}{4}$ right	12	3		12	6	
TOTAL		100	100		100	100

Table 4.2 – Comparison of Necker Cube results, input strength (*istr*) = 0.01

It can be seen that not only do both systems settle into local maxima less often, but NNW results are now nearly as good as the PDP results. Again, possible reasons for this are discussed in *Section 4.10. Discussion of Deviations*.

#### 4.3.5. Room Schemata

A Schema is a higher-level conceptual structure for representing the complex relationships implicit in a knowledge base; Schemata are data structures for representing generic concepts stored in memory. The Schema Model CS network is so named because it is able to, in some sense, represent this idea – information is processed by first finding the schema that best fits the current situation and then using that model to fill in aspects of the situation not specified by the current input. The units of a CS network correspond to hypotheses that certain semantic features are appropriate descriptions of a particular situation; some of these features are available in the input and form the starting place of the interpretation process while others are unspecified and must be filled in during the process of interpretation.



#### Chapter 4: Testing and Verification of Neural Network Software

```

0 ceiling 100  0 very-smal100  0 desk-cha 0  0 fire-pla 0  0 dresser  0
0 walls   100  0 desk        0  0 clock    0  0 drapes   0  0 televisi  0
0 door    100  0 telephon  0  0 picture  0  0 stove    0  ** bathtub 100
0 window   1  0 bed         0  0 floor-la 0  0 sink     100  0 toilet   100
0 very-lar 0  0 typewrit  0  0 sofa      0  0 refriger 0r  0 scale    100
0 large    0  0 book-she  0  0 easy-cha 0  0 toaster  0  0 coat-han  0
0 medium   0  0 carpet    0  0 coffee-c  0  0 cupboard100  0 computer  0
0 small    0  0 books     0  0 ash-tray 0  0 coffeepo 0  0 oven      0

cycleno 100    goodness 8.0895    temperature 2.0000

```

Figure 4.12 – CS Network, Schema Model, after activation of bathtub input

It has been shown that this kind of network could behave as if it contained schemata for five different kinds of rooms – living room, kitchen, bedroom, office and bathroom [Rumelhart 1986b]. The units in this case stood for the hypotheses that a particular room contained an oven, sofa, desk, bathtub, etc. The state of the network in NNW for this example, after running 100 cycles with bathtub activated, is shown in Figure 4.12. The results for activating bathtub, oven, desk and sofa are given in Table 4.3.

Input	Goodness	Activated Units
bathtub	8.09	ceiling, wall, door, very small, sink, cupboard, toilet, scale
oven	21.20	ceiling, wall, window, small, telephone, clock, coffee-cup, drapes, stove, sink, refrigerator, toaster, cupboard, coffeepot
desk	23.78	ceiling, wall, door, large, telephone, typewriter, bookshelf, carpet, books, desk-chair, picture, coffee-cup, ash-tray
sofa	27.01	ceiling, wall, door, window, very large, large, desk, telephone, typewriter, bookshelf, carpet, books, desk-chair, picture, floor-lamp, easy-chair, ash-tray, fire-place, drapes, computer

Table 4.3 – NNW results showing activated units for given activated inputs

Clearly providing input to a particular item tends to activate related items. The NNW results are identical to the PDP results. Further NNW tests on this example gave the same results as PDP and will not be detailed further.

## 4.4. Constraint Satisfaction Network, Boltzmann Machine

### 4.4.1. Boltzmann Machine Theory

A problem found with CS networks using the simple Schema Model is that they frequently tend towards local maxima of goodness rather than the global maximum value. A method that is used in the Boltzmann machine to get around this is *simulated annealing* (see Section 3.2. *Neural Network History and Terminology*) which adds a new global parameter analogous to temperature in physical systems. It acts in such a way as to decrease the strength of connections at the start and then change so as to strengthen them as the network is settling. It also exhibits some random behaviour so that instead of always moving 'uphill' in goodness, when the temperature is high it will sometimes move downhill; this allows it get out of local goodness peaks and tend instead to get 'caught' in the global maximum.

The Boltzmann Machine is similar to the Schema Model but has the following characteristics:

- Its units are binary and take on only their minimum and maximum values.
- The update rule specifies only a probability that a unit will take on one or other of the values.
- The behaviour of the system depends on the global parameter, *temperature*, which starts out high and decreases during the settling phase.

The update rule is probabilistic and is given by what is termed the *logistic* function:

$$probability(a_i(t) = 1) = \frac{1}{1 + e^{-net_i/T}} \quad - Eq. 4.11$$

where  $T$  is the temperature.

### 4.4.2. Necker Cube (Boltzmann Machine)

The results of a test on the Necker cube to compare PDP with NNW when using the Boltzmann model, with 100 runs, each of 200 cycles, are given in Table 4.4 – using an annealing schedule over 20 cycles.

Necker Cube Interpretation	PDP Results			NNW Results		
	Goodness/ <i>istr</i>	Occurrences	Total	Goodness/ <i>istr</i>	Occurrences	Total
Left, down	16	38	84 valid	16	39	76 valid
Right, up	16	46		16	37	
Outside edges	12	2	16 invalid			24 invalid
Inside edges	12	7		12	22	
Top left, bottom right				12	1	
Top right, bottom left	12	7		12	1	
$\frac{3}{4}$ right, $\frac{1}{4}$ left						
$\frac{3}{4}$ left, $\frac{1}{4}$ right						
TOTAL		100	100		100	100

Table 4.4 – Comparison of Necker Cube results, annealing over 20 cycles

When compared, NNW produced a set of results that are very similar to the PDP results. As before, there does appear to be a slight trend for PDP to reach global maxima more often than NNW, i.e. for NNW to get caught in local maxima more often than PDP (again, possible reasons for this are discussed in *Section 4.10. Discussion of Deviations*).

Compared to the Schema Model tests, the results are generally better than the original results, but not as good as the results with the lower *istr*. The tests were therefore repeated with a much more gentle annealing schedule (over 400 cycles rather than 20), running to 500 cycles. The results are given in Table 4.5.

Necker Cube Interpretation	PDP Results			NNW Results		
	Goodness/ <i>istr</i>	Occurrences	Total	Goodness/ <i>istr</i>	Occurrences	Total
Left, down	16	52	99 valid	16	51	99 valid
Right, up	16	47		16	48	
Outside edges			1 invalid			1 invalid
Inside edges						
Top left, bottom right	12	1		12	1	
Top right, bottom left						
$\frac{3}{4}$ right, $\frac{1}{4}$ left						
$\frac{3}{4}$ left, $\frac{1}{4}$ right						
TOTAL		100	100		100	100

Table 4.5 – Comparison of Necker Cube results, annealing over 400 cycles

It can be seen now that not only do both systems settle into local maxima less often, and NNW results are now nearly as good as the PDP results, but the results are even better than for the Schema Model with low input strength in the same number of cycles. This clearly demonstrates the utility of the Boltzmann model.

## 4.5. Constraint Satisfaction Network, Harmony Model

### 4.5.1. Harmony Model Theory

In the Harmony Model network (or 'Harmonium') developed by Paul Smolensky [Smolensky 1986] the principles are similar to the Boltzmann Machine but instead of an interconnected set of homogeneous units, there are two distinct layers of units. These are a lower level of representational feature units (corresponding to a featural description of a situation) and an upper layer of knowledge atoms (corresponding to pieces of knowledge about what configurations of features go together).

The Harmony Model has the following characteristics:

- The feature units take on activation values of  $\pm 1$  (representing that they are either present or absent).
- The knowledge atoms take on values of 0 or 1.

- All connections are between features and knowledge atoms (and are  $\pm 1$ ).
- Each knowledge atom has a strength (a new parameter, *sigma*) corresponding to the degree that the knowledge atom implies that the features to which it is connected are present in the input.

The goodness ("harmony") function is similar to that used before:

$$harmony = \sum_i \sigma_i a_i h_i \quad - Eq. 4.12$$

where  $i$  ranges over the knowledge atoms.

In this function  $h_i$  is a measure of the degree to which the current set of feature values is consistent with knowledge atom  $i$ , and is given by:

$$h_i = \frac{\sum_j r_j k_{ij}}{n_i} - \kappa \quad - Eq. 4.13$$

where  $j$  ranges over the features,  $r_j$  is the activation of representational feature  $j$ , and  $n_i$  is the number of non-zero connections to atom  $i$ .

The variable  $k_{ij}$  is given by:

$$\begin{aligned} k_{ij} &= 1 && \text{if positive connection} \\ k_{ij} &= -1 && \text{if negative connection} \\ k_{ij} &= 0 && \text{if no connection} \end{aligned}$$

That is, the total harmony is given by the sum of contributions of each of the knowledge atoms. If a knowledge atom is not activated ( $a_i=0$ ) there is no contribution. If it is active ( $a_i=1$ ) then it contributes an amount proportional to the product of its importance,  $\sigma_i$ , and a term representing the consistency of that atom with the current pattern of activation among the representational features. This consistency term,  $h_i$ , is the proportion of relevant features that are consistent, minus the proportion that are inconsistent, less a constant  $\kappa$  (a new parameter in NNW, *kappa*). When  $\kappa$  is near zero, turning on atom  $i$  will contribute a positive amount to the overall harmony of the system whenever the number of consistent features exceeds the number of inconsistent features (in this case the goodness function is the same as for the Schema and Boltzmann models). When  $\kappa$  is near 1, the general case in the Harmony Model, then it will contribute to the overall harmony only when all, or nearly all, of its features match the template for the atom.

## 4.5.2. Electricity Problem Solving

Once the Harmony Model had been added to NNW it was tested using the electricity problem proposed by Smolensky [Smolensky 1986; McClelland 1988]. This considers a simple electrical circuit containing a voltage source ( $V_T$ ) and two resistors ( $R_1$  and  $R_2$ ), and aims to determine the behaviour of the different elements when the others are changed. For example, if one of the resistances increases (and the other resistance and total voltage are unchanged) what happens to the voltage across each resistor ( $V_1$ ,  $V_2$ ) and to the current ( $I$ )? To solve this and similar problems the laws of electricity (Ohm's Law and Kirchoff's Law) are encoded as knowledge atoms – for example, there will be ones that encode the fact that  $V_T = V_1 + V_2$ , and so on.

	I	R1	R2	RT	V1	V2	VT			
Inputs	00	10	11	00	00	00	10		cycleno	0
	cu	cu	cu	cu	cu	cu	cu		temp	1.0000
Features	00	10	11	00	00	00	10		harmony	0.0000
knowledge atom activations										
	u	u	u	u	u	s	s	s	d	d
	u	s	d	d	d	u	s	d	u	u
	u	u	u	s	d	u	s	d	u	s
$V_1 + V_2 = V_T$	0	0	0	0	0	0	0	0	0	0
$R_1 + R_2 = R_T$	0	0	0	0	0	0	0	0	0	0
$I * R_1 = V_1$	0	0	0	0	0	0	0	0	0	0
$I * R_2 = V_2$	0	0	0	0	0	0	0	0	0	0
$I * R_T = V_T$	0	0	0	0	0	0	0	0	0	0

Figure 4.13 – CS Network, Harmony Model, initial state of electricity problem

This situation before starting to cycle with the given problem is shown in Figure 4.13. At this stage the interesting lines are marked "Inputs" and "Features". In these, in each pair of digits the first digit (under 'c') represents whether the feature changes (0 means not known, 1 means it changes, -1 means it does not change) while the second digit (under 'u') represents whether the feature changes up or down (0 means not known, 1 means it increases, -1 means it decreases); the second digit is irrelevant if the first digit is a -1. The Inputs line represents the problem – in the example given it means that  $R_1$  is fixed (first digit is -1, second digit is ignored),  $R_2$  is changing (first digit is 1) and increasing (second digit is 1) and  $V_T$  is fixed (first digit is -1, second digit is ignored). The aim when the network is run is for the unknown features to be filled in correctly (each 0 in the Features line).

	I	R1	R2	RT	V1	V2	VT		cycleno	200
Inputs	00	10	11	00	00	00	10		temp	0.0500
	cu	cu	cu	cu	cu	cu	cu		harmony	1.1667
Features	11	11	11	11	11	11	11			
knowledge atom activations										
	u	u	u	u	u	s	s	s	d	d
	u	s	d	d	d	u	s	d	u	u
	u	u	u	s	d	u	s	d	u	s
V1 + V2 = VT	0	0	0	0	0	0	0	0	1	0
R1 + R2 = RT	0	0	0	0	0	1	0	0	0	0
I * R1 = V1	0	0	0	0	0	0	0	0	0	1
I * R2 = V2	0	0	0	0	0	0	0	0	1	0
I * RT = VT	0	0	0	0	0	0	0	0	1	0

Figure 4.14 – CS Network, Harmony Model, final state of electricity problem

This situation after 200 cycles is shown in Figure 4.14 and the network has successfully determined the behaviour of the other features. The Features line can be interpreted as follows (ignoring the elements already specified in the Inputs line): the current  $I$  will change and decrease, the total resistance  $R_T$  will change and increase,  $V_1$  will change and decrease, and  $V_2$  will change and increase. After 200 cycles the features have settled apart from the up/down digit of those that are fixed; these continue to flick between 1 and -1 but are ignored anyway.

It is of interest to note that the different features usually settle sequentially and in the same order. Specifically, they settle as follows:  $V_2$  goes to 11 (voltage across  $R_2$  will change and increase) after about 50 cycles,  $I$  goes to 1-1 (current will change and decrease) after about 100 cycles,  $R_T$  goes to 11 (total resistance will change and increase) after about 150 cycles, and finally  $V_1$  goes to 1-1 (voltage across  $R_1$  will change and decrease).

Underneath the Inputs and Features are the representations of the knowledge atoms in an array where the rows indicate electrical relationships and the columns represent the new relationships determined by the network. For example, in Figure 4.14, for the first row ( $V_1 + V_2 = V_T$ ) the network has activated the d-u-s relationship, i.e. that when the first term ( $V_1$ ) goes down ('d') and the second term ( $V_2$ ) goes up ('u') the third term ( $V_T$ ) remains the same ('s'). Even after 200 cycles the knowledge atom relationships occasionally flick and continue to do so indefinitely. However, the relationships being temporarily activated are also valid, though clearly not as highly activated as the stable ones. In the example given, the next column to the left is also often activated (d-u-u) meaning that when  $V_1$  goes

down and  $V_2$  goes up then  $V_T$  goes up, which can also be true (obviously depending on the relative magnitudes of the changes in  $V_1$  and  $V_2$ ).

The results from NNW are essentially identical to the PDP results. They settle different features at different number of cycles, but the variation between the two systems is only of the order of magnitude of the variation between different runs of each system.

## 4.6. Pattern Associator Network, Hebb Learning Rule

### 4.6.1. PA Theory

The Pattern Associator is a device that learns associations between input patterns and output patterns. It is also interesting because what it learns about one pattern it tends to generalise to other similar patterns. Pattern Associators can learn to act as content-addressable memories; they generalise the responses they make to novel inputs that are similar to the inputs they have been trained on; they learn to extract the prototype of a set of repeated experiences in ways that are very similar to the concept-learning characteristics seen in human cognitive processes; and they degrade gracefully with damage and noise.

Inside the network there are two sets of units: input units and output units. There is also a matrix representing the connections from the inputs to the outputs. The general arrangement of a PA network is shown in Figure 4.15. Two main learning rules are used, the 'Hebb' rule (see Section 4.6.2. *The Hebb Rule*) developed by W. James in 1890 and again by D. Hebb in 1949, and the error-correcting or 'Delta' rule (see Section 4.7.1. *The Delta Rule*) studied by Widrow and Hoff, and Rosenblatt (see Section 3.2. *Neural Network History and Terminology*).

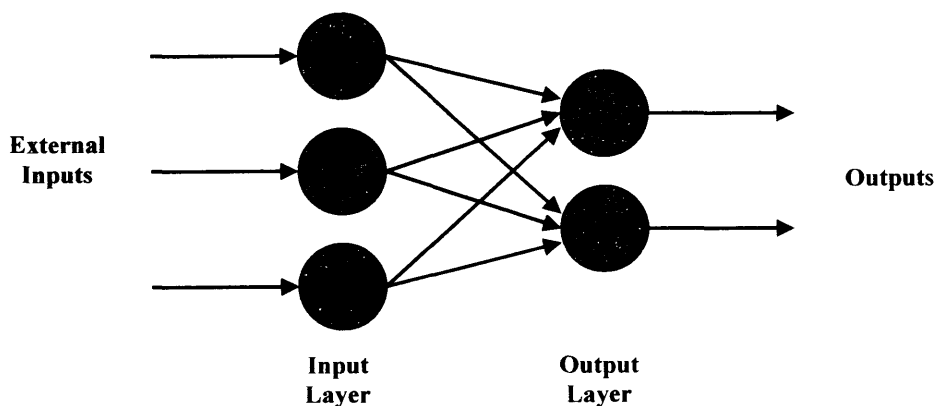


Figure 4.15 – Diagrammatic representation of a PA Network



In the PA network, pattern pairs are presented consisting of an input pattern and a target pattern. A training *epoch* consists of one learning trial on each pattern pair. On each trial, the input is presented, the corresponding output is computed, and the weights are updated. Patterns may be presented in fixed sequential order or in permuted order within each epoch (in NNW this is selected on the `Options` menu).

Four activation rules are available: Linear, Linear Threshold, Stochastic and Continuous Sigmoid. In Linear activation the output of a unit is simply equal to the net input. In Linear Threshold the output is set to 1 if its net input is positive, otherwise it is set to 0 (this form was used in the Perceptron). In Stochastic, the default, the output is set to 1 with a probability given by the logistic function (see *Section 4.4.1. Boltzmann Machine Theory*). In Continuous Sigmoid, each of the output units takes on an activation given by the logistic function.

After processing each pattern, several measures of the output that is produced, and its relation to the target, are computed. The patterns are essentially vectors, and the measures are the normalised dot product (*ndp*), the normalised vector length (*nvl* – the magnitude of the output normalised by the number of elements) and the vector correlation (*vcor* – which measures the similarity of the vectors independent of their length). Further measures are the pattern sum of squares (*pss* – the sum over all output units of the squared error) and the total sum of squares (*tss* – the sum of *pss* values for each pattern in the training set). The various values of these measures can be seen in the output displays illustrating the following tests but their consideration is not essential to the understanding of the processes involved and they will not be discussed in further detail (for a formal analysis see [Jordan 1986]).

#### 4.6.2. The Hebb Rule

Hebb proposed that when two cells fire at the same time, the strength of the connection between them should be increased. More formally, this rule can be expressed as follows:

$$\Delta w_{ij} = \varepsilon a_i a_j \quad \text{- Eq. 4.14}$$

where  $\varepsilon$  is referred to as the *learning rate parameter*.

Activations of the input units are 'clamped' (see *Section 4.3.1. CS Theory*) based on an externally supplied input pattern, and activations of the output pattern are clamped to the values given by some externally supplied target pattern. Learning

then occurs by adjusting the strengths of the connections using the Hebb rule formulated as:

$$\Delta w_{ij} = \epsilon o_i i_j \quad - \text{Eq. 4.15}$$

where  $o_i$  is the activation of output unit  $i$ , and  $i_j$  is the activation of input unit  $j$ .

#### 4.6.3. PA Implementation

A network definition for PA is loaded into NNW as for previous types, but then a pattern file (.pat) is loaded in which holds definitions of pattern pairs (i.e. an input pattern and a target pattern) with pattern labels. The network is trained (i.e. it is put into learning mode) using `Run:Go/Train`. Training will stop before all requested epochs are complete if the total sum of squares, *tss* (see Section 4.6.1. PA Theory), is less than an error criterion value, *ecrit*.

For testing (i.e. learning mode is off) all patterns can be applied using `Run:Test All`. An individual pattern pair can be selected with `Patterns:Select Pattern`, or an input or target pattern can be selected with `Patterns:Select Input` or `Patterns:Select Target`, or a new pattern and/or pair can be created with `Patterns:Enter Pattern` – in all cases the selected pattern/pair can then be tested with `Run:Test`.

There are three new parameters in PA: the *learning rate*, which scales the size of the changes to the weights (and is equivalent to  $\epsilon$ ); *noise*, which determines the amount of random variability added to elements of input and target patterns; and *temperature*, used as the denominator of the logistic function to scale net inputs in the stochastic mode.

#### 4.6.4. Generalisation and Similarity

Once NNW had been extended to model Pattern Associator networks it was tested on a problem proposed by the PDP group [McClelland 1988] which aims to demonstrate how its output after training on a given pattern is affected by the similarity of the input pattern to the original trained pattern. The network definition files specify a linear Hebb rule PA with eight input units and eight output units starting with zero initial weights. A further file, the pattern file (.pat) defines a pattern pair for training with, consisting of two sequences of eight numbers (each either +1 or -1) – the first eight being the input pattern and the second eight being the target pattern. Once the PA has been trained on the pattern pair it can then be

tested to see if it has learned the pattern and can reproduce it, and also tested to see if it can reproduce other similar patterns. The network state before training is shown in Figure 4.16.

```

                                pname ipattern tpattern
                                a      11111111 11111111

epochn      0
cpname
ndp      0.0000
nvl      0.0000
vcor      0.0000
pss      0.0000
tss      0.0000

                                out   tar
weights      0  0  0  0  0  0  0  0      0    0
              0  0  0  0  0  0  0  0      0    0
              0  0  0  0  0  0  0  0      0    0
              0  0  0  0  0  0  0  0      0    0
              0  0  0  0  0  0  0  0      0    0
              0  0  0  0  0  0  0  0      0    0
              0  0  0  0  0  0  0  0      0    0
              0  0  0  0  0  0  0  0      0    0
input        0  0  0  0  0  0  0  0

```

Figure 4.16 – PA Network, Hebb Learning, network state before training

The state shown in Figure 4.17 is after training on the pattern pair (which is shown in the upper right corner; it has pattern name 'a', input pattern 1-11-11-11-1 and target pattern 11-1-111-1-1). The new weights calculated by the network are shown (their absolute magnitude is 0.125, i.e.  $1/\text{no. units}$ ) and it can be shown that these network weights will successfully convert the input pattern (shown as the `input` row underneath the weight matrix) to an output pattern (shown as the `out` column to the right of the weight matrix) that is identical to the target pattern (shown as the `tar` column to the right of the output column).

```

                                pname ipattern tpattern
                                a      11111111 11111111

epochn      1
cpname      a
ndp      0.0000
nvl      0.0000
vcor      0.0000
pss      8.0000
tss      8.0000

                                out   tar
weights      13 13 13 13 13 13 13 13     100  100
              13 13 13 13 13 13 13 13     100  100
              13 13 13 13 13 13 13 13     100  100
              13 13 13 13 13 13 13 13     100  100
              13 13 13 13 13 13 13 13     100  100
              13 13 13 13 13 13 13 13     100  100
              13 13 13 13 13 13 13 13     100  100
              13 13 13 13 13 13 13 13     100  100
input        100100100100100100100100

```

Figure 4.17 – PA Network, Hebb Learning, network state after training

#### Chapter 4: Testing and Verification of Neural Network Software

It can also be tested with a pattern that it has not seen before. Entering a new input pattern 1-11-11111, labelled 'b', using the same target pattern, and then testing the network produces the results in Figure 4.18. Now the output pattern has matched the target pattern but at only half the activation (0.5 in each position instead of 1).

										<p>pname ipattern tpattern</p> <p>a 11111111 11111111</p> <p>b 11111111 11111111</p>		
<p>epochn 1</p> <p>cpname b</p> <p>ndp 0.5000</p> <p>nvl 0.5000</p> <p>vcor 1.0000</p> <p>pss 2.0000</p> <p>tss 10.0000</p>												
										out	tar	
weights	13	13	13	13	13	13	13	13	13	50	100	
	13	13	13	13	13	13	13	13	13	50	100	
	13	13	13	13	13	13	13	13	13	50	100	
	13	13	13	13	13	13	13	13	13	50	100	
	13	13	13	13	13	13	13	13	13	50	100	
	13	13	13	13	13	13	13	13	13	50	100	
	13	13	13	13	13	13	13	13	13	50	100	
	13	13	13	13	13	13	13	13	13	50	100	
	13	13	13	13	13	13	13	13	13	50	100	
input	100100100100100100100100											

Figure 4.18 – PA Network, Hebb Learning, network state after testing with a new pattern

A limitation to the Hebb learning rule becomes clear if the sets of patterns are orthogonal, or not orthogonal but linearly independent. In both cases, after one epoch of training, a trained input pattern is perfectly reproduced at the output; however, after further training the output pattern has too high an activation (and for the orthogonal case it is exactly scaled by the number of epochs of training). Figure 4.19 shows the results of testing the first pattern (from the list of orthogonal patterns at the upper right) after three epochs of training. A means of solving this problem is discussed in the next section; however all these NNW results are identical to the PDP results.

										pname	ipattern	tpattern
										a	11111111	11111111
										b	11111111	11111111
										c	11111111	11111111

epochn	3										
cpname	a										
ndp	3.0000										
nvl	3.0000										
vcor	1.0000										
pss	32.0000										
tss	32.0000										

										out	tar
weights	113	38	38	38113	38	38	38			300	100
	38	38	38113	38	38	38113				300	100
	38113	38	38	38113	38	38				300	100
	38	38113	38	38	38113	38				300	100
	38113	38	38	38113	38	38				300	100
	38	38113	38	38	38113	38				300	100
	113	38	38	38113	38	38	38			300	100
	38	38	38113	38	38	38113				300	100

input	100100100100100100100100										
-------	--------------------------	--	--	--	--	--	--	--	--	--	--

Figure 4.19 – PA Network, Hebb Learning, network state after three epochs of training

## 4.7. Pattern Associator Network, Delta Learning Rule

### 4.7.1. The Delta Rule

With the Delta rule, the idea is that the difference between the desired target activation and the obtained activation can be used to drive learning, i.e. by adjusting the strengths of the connections so that they will tend to reduce the difference or error measure. It can be written:

$$\Delta w_{ij} = \epsilon e_i a_j \quad - \text{Eq. 4.16}$$

where  $e_i$ , the error for unit  $i$ , is given by:

$$e_i = t_i - a_i \quad - \text{Eq. 4.17}$$

the difference between the teaching input to unit  $i$  and its obtained activation.

In NNW the selection of Hebb or Delta rule is made on the Options menu.

### 4.7.2. Solving for Orthogonality and Linear Independence

When the Delta Rule is used the weights are changed in such a way that the output activations do not just increase with the amount of training. When testing with the orthogonal patterns from the previous section the output pattern is exactly correct after just one epoch, and *after more epochs the weights do not change further*, so the output pattern remains correct. When testing with the linearly independent patterns the first output pattern is only approximately correct after just one epoch, but *after more epochs the weights change so that the output pattern converges on the target*

pattern. Figure 4.20 shows the results of testing the first pattern (from the list of linearly independent patterns at the upper right) after just three epochs of training.

	pname	ipattern	tpattern
	a	11111111	11111111
	b	11111111	11111111
	c	11111111	11111111

epochn	3		out	tar
cpname	a			
ndp	0.9844			
nvl	0.9849			
vcor	0.9995			
pss	0.0098			
tss	0.0098			

	out	tar
weights		
29 4 4 29 29 4 12 13	102	100
12 13 13 12 12 13 36 61	95	100
4 29 29 4 4 29 13 12	102	100
13 12 12 13 13 12 61 36	95	100
4 29 29 4 4 29 13 12	102	100
13 12 12 13 13 12 61 36	95	100
29 4 4 29 29 4 12 13	102	100
12 13 13 12 12 13 36 61	95	100

input	100100100100100100100
-------	-----------------------

Figure 4.20 – PA Network, Delta Learning, network state after 3 epochs of training

Further NNW tests were conducted on PA networks which, when using the Delta rule, demonstrated they had the ability to cope well with noise. For example, repeating the previous test on the orthogonal pattern but introducing a high level of noise (randomly distributed between -0.5 and 0.5) to each element in the input and target patterns still produced good results (Figure 4.21 shows results for the first pattern after the network has been trained for 100 epochs).

	pname	ipattern	tpattern
	a	11111111	11111111
	b	11111111	11111111
	c	11111111	11111111

epochn	100		out	tar
cpname	a			
ndp	1.0410			
nvl	1.0203			
vcor	0.9624			
pss	0.6637			
tss	0.6637			

	out	tar
weights		
38 8 13 10 39 10 13 9	103	94
13 15 15 41 14 17 14 37	120	77
14 40 13 13 14 37 10 14	86	106
13 14 37 13 8 10 34 11	88	81
10 39 13 15 12 40 13 18	79	134
13 13 40 14 10 11 37 12	94	120
34 13 11 9 37 14 9 8	105	82
14 10 15 39 18 11 11 38	130	135

input	129 59 79123108107113100
-------	--------------------------

Figure 4.21 – PA Network, Delta Learning, network state after 100 epochs of training, despite noise

These tests demonstrated the effectiveness of the Delta Learning Rule which is now the primary learning rule used in neural networks. It is an important element of the most significant type of learning neural network, the Back Propagation network described in the next section.

In all the PA tests the NNW results were essentially identical to the PDP results and so the software was developed further to include back propagation networks.

## 4.8. Back Propagation Network

### 4.8.1. BP Theory

The Pattern Associator network described above has been known since the late 1950s when variants of the Delta Rule were first proposed. In one version, in which the output units were purely linear, it was known as the Least Mean Square (LMS) associator; this used the delta rule for adjusting connection strengths using a gradient descent method. In the most well known version, in which output units were linear threshold units, it was known as the Perceptron. Many important theorems were proved about both of these versions, the most significant being the Perceptron Convergence Theorem. This demonstrated the remarkable truth that the Perceptron learning procedure is *guaranteed to find a set of weights that can correctly classify input vectors if such a set of weights exists*.

Unfortunately, such a mapping does not always exist. In their famous book, *Perceptrons*, Minsky and Papert demonstrated in 1969 the limitations of the Perceptron (see *Section 3.2. Neural Network History and Terminology*), and specifically showed that it can only solve functions that are linearly separable. A function that appears to be simple but is not linearly separable and is therefore not solvable by a Perceptron is the Exclusive-Or (XOR) problem. In the XOR problem the inputs 00, 01, 10 and 11 should produce the outputs 0, 1, 1, 0 (i.e. activated if either input is activated but not if both are activated). This problem is not linearly separable because, put simply, if the inputs are considered as the co-ordinates (0, 0), (0, 1), (1, 0) and (1, 1) and their outputs are plotted at these co-ordinates (i.e. 0 at 0,0; 1 at 0,1; 1 at 1,0 and 0 at 1,1) then it is not possible to find a line that can be drawn to separate the 0 outputs from the 1 outputs. Therefore it had been demonstrated that even a problem as simple as the XOR one could never be solved by a Perceptron or other PA network.



One possible way forward was that it was known that any linearly separable problem could be solved by moving up to the next dimension. In the case of the XOR function a distinction could be made between the first 0 output and the last by 'ANDing' the first two dimensions, so that the inputs were 000, 010, 100 and 111; when these are plotted it is a simple matter to find a *plane* that separates the 0 and 1 outputs. The equivalent procedure with a neural network is to add an extra layer of units. Unfortunately, these units neither receive inputs directly nor are given direct feedback; they are known as hidden units and the problem is knowing how to teach them. The original Perceptron learning procedure could not be applied to more than one layer, and Minsky and Papert believed that no such general procedure could be found. As a result of their work, research into neural networks went into a decline that lasted for more than a decade.

The history of the developments that began in the early 1980s and led to the back propagation method of learning has been given above (see *Section 3.2. Neural Network History and Terminology*). Back Propagation networks combine hidden units with the BP method of learning and their general arrangement is shown in Figure 4.22.

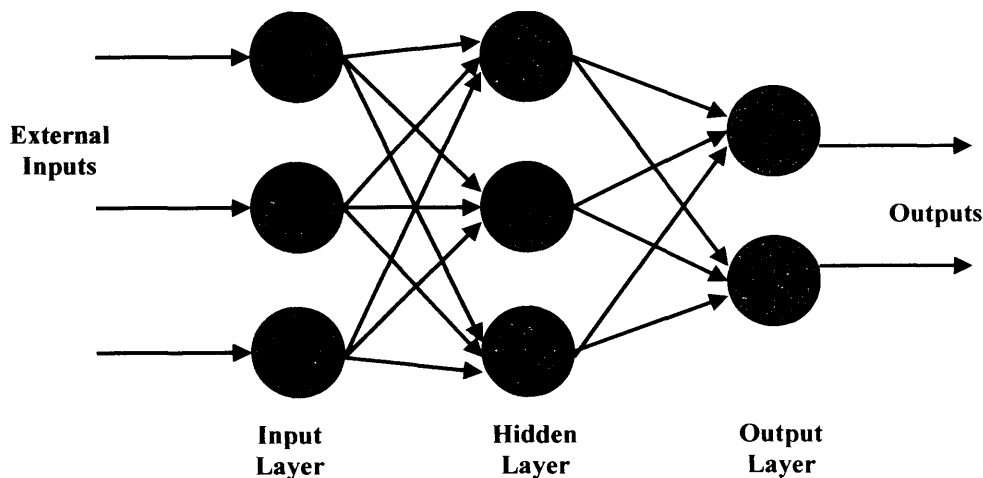


Figure 4.22 – Diagrammatic representation of a BP Network

The basic idea of the back propagation method of learning is to combine a non-linear Perceptron-like system capable of making decisions with the objective error function and gradient descent of the LMS associator. In such a system the total LMS error (i.e. summed squared error) is given by:

$$E = \sum_p E_p = \sum_p \sum_i (t_{pi} - o_{pi})^2 \quad - \text{Eq. 4.18}$$

where  $p$  ranges over the set of input patterns,  $i$  ranges over the set of output units, and  $E_p$  represents the error on pattern  $p$ .

The object is to find a set of weights that minimises this function. The idea of gradient descent is to make a change in the weight proportional to the negative of the derivative of the error, as measured on the current pattern, with respect to each weight (negative because if the weight is above the minimum value the slope is positive and the weight needs to be decreased, and *vice versa*), i.e. a learning rule of the form:

$$\Delta w_{ij} = -k \frac{\partial E_p}{\partial w_{ij}} \quad - \text{Eq. 4.19}$$

where  $k$  is the constant of proportionality,  $t_{pi}$  is the desired target for output unit  $i$ , and  $o_{pi}$  is the actual output of the output unit  $i$ , when the pattern  $p$  has been presented.

Carrying out the derivative of the error measure just described gives:

$$\Delta w_{ij} = \varepsilon \delta_{pi} i_{pj} \quad - \text{Eq. 4.20}$$

where  $\varepsilon = 2k$  and  $\delta_{pi} = t_{pi} - o_{pi}$  is the difference between the target for unit  $i$  on pattern  $p$  and the actual output produced.

This is exactly the Delta Rule previously described. If weights are changed according to this rule, each weight is moved towards its own minimum and the system moves downhill until it reaches its minimum error value. When all of the weights have reached their minimum points, the system has reached equilibrium. If the system is able to solve the problem entirely, it will reach zero error and the weights will no longer be modified. If the system is unable to solve the problem entirely, it will have found a set of weights that produces the minimum error.

Essentially  $\delta_{pi}$  represents the effect of a change in the net input to unit  $j$  on the output of unit  $i$  in pattern  $p$ . In Back Propagation, the determination of  $\delta$  is a recursive process that starts with the output units for which the rule is<sup>1</sup>:

$$\delta_{pi} = (t_{pi} - a_{pi}) f'_i(net_{pi}) \quad (\text{for output units}) \quad - \text{Eq. 4.21}$$

where:

$$net_{pi} = \sum_j w_{ij} a_{pj} + bias_i \quad - \text{Eq. 4.22}$$

and  $f'_i(net_{pi})$  is the derivative of the activation function with respect to the change in the net input to the unit. The  $\delta$  term for hidden units for which there is no specified target is determined recursively in terms of the  $\delta$  terms of the units to which it directly connects and the weights of those connections:

$$\delta_{pi} = f'_i(net_{pi}) \sum_k \delta_{pk} w_{ki} \quad (\text{for hidden units}) \quad - \text{Eq. 4.23}$$

The application of the BP rule, therefore, involves two phases: during the first phase the input is presented and propagated forward through the network to compute the output value  $a_{pj}$  for each unit. This output is then compared to the target, resulting in a  $\delta$  term for each output unit. The second phase involves a backward pass through the network during which the  $\delta$  term is computed for each unit in the network – this allows the recursive computation of  $\delta$  indicated above. Once these phases are complete, for each weight is calculated the *weight error derivative* – the product of the  $\delta$  term associated with the unit it projects to times the activation of the unit it projects from. The weight error derivatives can then be used to calculate actual weight changes pattern-by-pattern, or accumulated over all patterns.

The BP rule only works, as has been shown, if there is a derivative of the activation function,  $f'_i(net_i)$ . The PA network used three types of activation function: linear, linear threshold and the logistic function (the latter being used in both the stochastic and continuous sigmoid rules). The linear system achieves no advantage from hidden units, the linear threshold function is discontinuous, and so the logistic function is used for BP. The derivative of this function with respect to its total input  $net_{pi}$  is given by:

---

<sup>1</sup> For a BP network the output of a unit is equal to its activation, so  $o_{pi} = a_{pi}$ .

$$\frac{da_{pi}}{dnet_{pi}} = a_{pi}(1 - a_{pi}) \quad - \text{Eq. 4.24}$$

so that the error signals are given by:

$$\delta_{pi} = (t_{pi} - a_{pi})a_{pi}(1 - a_{pi}) \quad (\text{for output units}) \quad - \text{Eq. 4.25}$$

$$\delta_{pi} = a_{pi}(1 - a_{pi}) \sum_k \delta_{pk} w_{jk} \quad (\text{for hidden units}) \quad - \text{Eq. 4.26}$$

#### 4.8.2. Solving the XOR Problem

NNW was adapted to do Back Propagation networks and tested on the XOR problem. Using appropriate network definition files and an initial set of weights, a network is created with two inputs units, two hidden units and one output unit. The state of the network is shown in Figure 4.23 after testing on each pattern but before any training has taken place.

epoch	0	tss	1.0554	pname	ipatterns	tpatterns
		gcor	0.0000	p00	0 0	0
cpname	p11	pss	0.3857	p01	0 1	1
				p10	1 0	1
				p11	1 1	0
sender acts:	100	100	65	40	bia net act	tar del
weights:	43	45			28	60 65 9
	4	4			40	40 40 3
			27	8	28	49 62 0 146

Figure 4.23 – BP Network, XOR Problem, before training

The four XOR patterns (inputs and output) are shown at top right, and labelled p00 to p11. The current pattern is shown top left, below the current epoch number, and the main statistics are shown in the centre, i.e. *tss* and *pss* (see Section 4.6.1. *PA Theory*). A new statistic is the gradient correlation (*gcor*) which is the vector correlation of the current weight error derivatives with the previous ones and indicates whether the gradient is staying relatively stable or shifting from epoch to epoch (e.g. a negative value indicates the gradient is changing direction). Its value can be considered to be following the gradient and so the mode for turning on this calculation is called *follow* (selected with `Options:Follow is on` in NNW).

The 'sender' activations are shown beneath, these are the outputs from units that are inputs to other units (i.e. the two input and the two hidden units). They form the

headings of columns of the (sparse) weight matrix – each column shows the weights for that sender unit while the rows indicate where the weight is going to, so the first input has the weights 43 and -4 to the two hidden units, the second input has the weights 45 and 4 to the two hidden units, and the two hidden units have the weights 27 and 8 to the output unit. The columns to the right provide insights into the internal values within the network (the biases, net inputs, activations and delta values for the 'receiving' units) plus show the target value for the output unit (0 for pattern p11).

The network is then trained on the four patterns, changing the weights as necessary to achieve the required target for each input pair presented. After about 60 epochs the weights begin to build up. After about 200 epochs one of the hidden units starts to act like an OR unit; its output is about the same for all input patterns in which one or more input units is on. After 300 epochs the *tss* value is below the required threshold; the system has solved the XOR problem. The results of testing on the four patterns 00, 01, 10 and 11 are shown in Figure 4.24; the input pair has been highlighted, as well as the output activation ('act') and target ('tar'), in each case.

tss	0.0101	pname	ipatterns	tpatterns	tss	0.0187	pname	ipatterns	tpatterns
gcor	0.00006	p00	0 0	0	gcor	0.0000	p00	0 0	0
pss	0.0101	p01	0 1	1	pss	0.0085	p01	0 1	1
		p10	1 0	1			p10	1 0	1
		p11	1 1	0			p11	1 1	0
<div> <div>0 0</div> <div>9 1</div> <div>bia net act tar del</div> <div>582 582</div> <div>340 340</div> <div>673 742</div> <div>236 236</div> <div>521 521</div> <div>297 243</div> <div>9</div> <div>1</div> <div>8 0</div> <div>3</div> <div>0</div> <div>6</div> </div>					<div> <div>0 100</div> <div>97 14</div> <div>bia net act tar del</div> <div>582 582</div> <div>340 340</div> <div>673 742</div> <div>236 347</div> <div>521 181</div> <div>297 250</div> <div>97</div> <div>14</div> <div>92 100</div> <div>1</div> <div>5</div> <div>5</div> </div>				
<div> <div>100 0</div> <div>97 14</div> <div>bia net act tar del</div> <div>582 582</div> <div>340 340</div> <div>673 742</div> <div>236 346</div> <div>521 181</div> <div>297 250</div> <div>97</div> <div>14</div> <div>92 100</div> <div>1</div> <div>5</div> <div>5</div> </div>					<div> <div>100 100</div> <div>100 83</div> <div>bia net act tar del</div> <div>582 582</div> <div>340 340</div> <div>673 742</div> <div>236 928</div> <div>521 159</div> <div>297 241</div> <div>100</div> <div>83</div> <div>8 0</div> <div>0</div> <div>7</div> <div>6</div> </div>				
tss	0.0272	pname	ipatterns	tpatterns	tss	0.0379	pname	ipatterns	tpatterns
gcor	0.0000	p00	0 0	0	gcor	0.0000	p00	0 0	0
pss	0.0085	p01	0 1	1	pss	0.0107	p01	0 1	1
		p10	1 0	1			p10	1 0	1
		p11	1 1	0			p11	1 1	0

Figure 4.24 – BP Network, XOR Problem, testing results after training with 300 epochs

The results achieved with NNW were essentially the same as the PDP results except that NNW takes 300 cycles to complete where PDP takes 289; possible reasons for this are discussed in *Section 4.10. Discussion of Deviations*.

#### 4.9. NNW Features

The NNW interface application provides comprehensive facilities for loading in network definition files; defining and creating networks directly through menu and dialog box options; processing defined networks in order to evaluate their result; and the reporting of results. Its arrangement of windows was covered in *Section 3.7. NNW in Use* and its menu commands and other features are detailed in Appendix C.

#### 4.10. Discussion of Deviations

NNW can currently import most network definition files conforming to the PDP specification [McClelland 1988]. Initial results indicate that on PDP example networks NNW returns the same results as PDP software with a few exceptions. These deviations fall into three categories:

1. NNW produced substantially different behaviour to PDP in the IAC spontaneous generalisation tests (see *Section 4.2.7. Spontaneous Generalisation*), specifically after appearing to settle into a valid stable state the PDP system started to change again and move towards a new state that appeared to be invalid.
2. NNW is slightly more susceptible to getting caught in local minima, where PDP seems to reach the global maximum more easily (see *Sections 4.3.4. Necker Cube (Schema Model)* and *4.4.2. Necker Cube (Boltzmann Machine)*).
3. NNW sometimes takes slightly longer (i.e. more cycles) to settle to a solution compared to PDP (see *Section 4.8.2. Solving the XOR Problem*).

It was considered that this behaviour could be caused at a low level in the mathematical implementation by the transfer function diverging in proportion to the difference between very similar numbers. The PDP software is implemented using the *float* representation of floating point numbers in the C language [McClelland 1988 p322], where NNW uses the *double* representation (i.e. double the length of float). The Microsoft C compiler uses the IEEE *float* format (and PDP probably does too) which is four bytes long (1 bit for sign, 8 bits for exponent and 23 bits for mantissa) whereas its *double* is eight bytes long (1 bit for sign, 11 bits for exponent and 52 bits for mantissa) –

a substantially greater resolution. The PDP implementation may therefore be much more susceptible to small errors in the stored value of numbers<sup>1</sup>.

This can work against the PDP system in, for example, the spontaneous generalisation example because the network is trying to hold stable with three units in perfectly balanced equilibrium (because the gang units are equally distributed between three occupations). It is suggested that due to the small errors resulting from the use of *float* the activations for the three units are not exactly the same and, because of the IAC tendency for the "rich to get richer" one of them eventually wins out.

However, it appears that these errors can work *for* the PDP system in, for example, the constraint satisfaction and back propagation examples. It is suggested that in a CS network the errors may act like a kind of random noise that helps the system sometimes climb out of local minima, rather like the simulated annealing is explicitly trying to do on a much larger scale. Also, it is suggested that in the BP network example, the errors may help the system get to a solution slightly faster by reducing the occurrence of situations where similar activation values are competing and tending to slow down the settling. These areas could be worth investigating in future work.

As an experiment, the main `DOUBLEVAR` type in the `NNW NEURON` data structure (see Section 4.3.2. *CS Implementation*) was changed to use *float* in place of *double*. The test with the most visibly deviating behaviour, the IAC spontaneous generalisation test, was repeated and the results are given in Figure 4.25. While the NNW results for the average activation of the `Gang` are now nearly identical to the PDP results, the results for the `Pusher` are very different, dropping very quickly where the PDP results rose quickly – though the change takes place at about the same number of cycles.

This result seems to confirm that the behaviour observed is due to a lack of accuracy when using the *float* representation of floating point numbers (it should be noted, however, that the test did not make NNW completely equivalent in data storage to PDP, since all the intermediate and local variables in NNW continued to be held using the *double* representation). The result also implies that once significant error has been introduced into the system it is no longer stable and may diverge unpredictably (and if the system is monitored over a long period, even to 1000 cycles or more, it can be

---

<sup>1</sup> Note that it is probably the storage, not the calculation, of the values that introduces the error since most C compilers conduct floating point calculations at full precision then convert the result to *double* or *float*.



observed that the activations still change occasionally, i.e. the system never seems to settle to a fully stable state). Furthermore, it is suggested that the activations for `Gang` are behaving correctly, but that their reducing values can be considered to indicate that the system no longer has 'confidence' in the various activations in the network being consistent with each other.

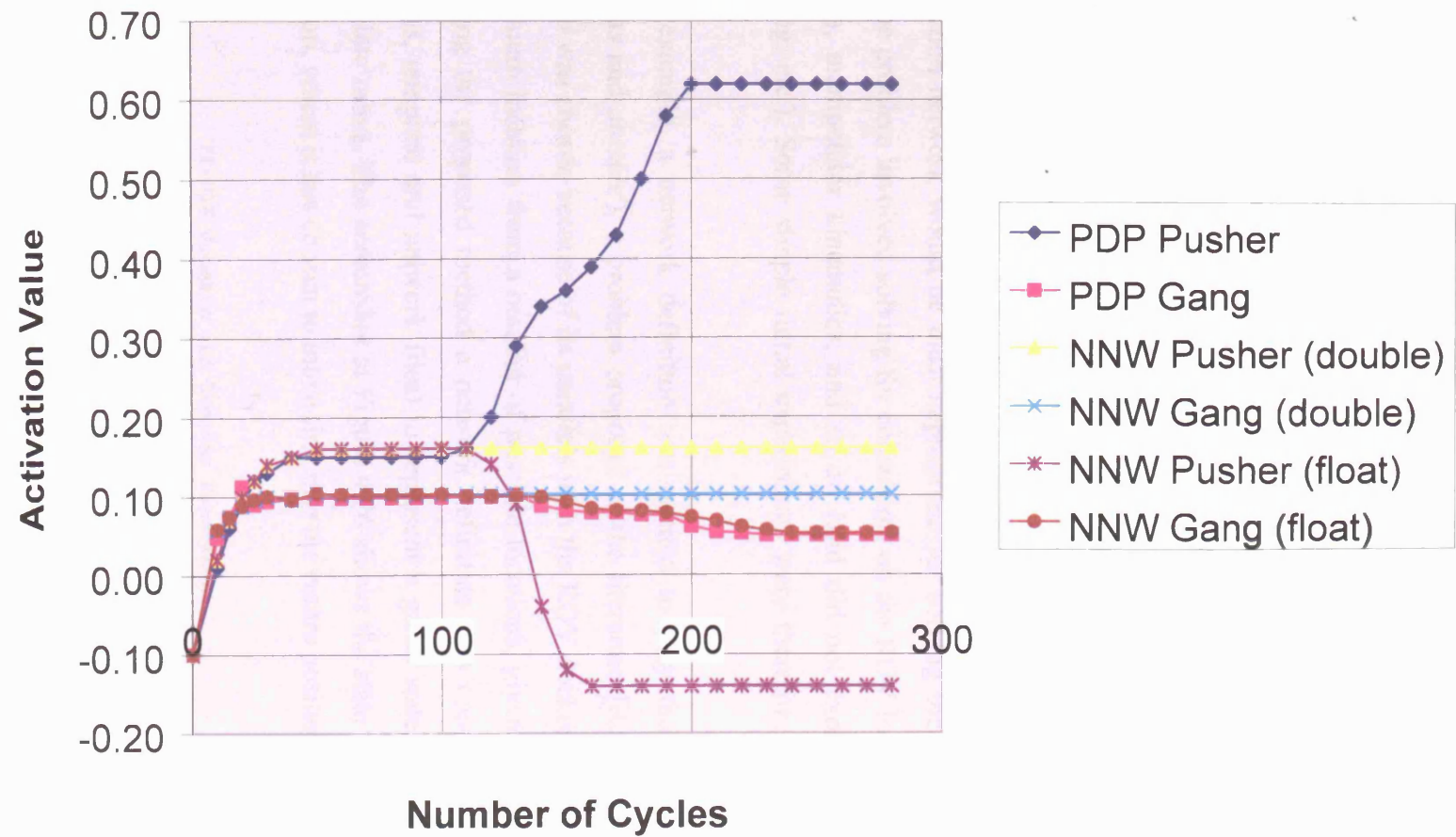


Figure 4.25 – Comparison of activation values changing over time for PDP versus NNW using *double* and *float* floating point representations

#### 4.11. Early Experimentation

At this stage in the development of NNW, the interesting anomalies discussed in the previous section notwithstanding, it was clear that the software was working correctly. In parallel it had become clear that of the different areas that neural networks could usefully be applied to supplement a supervisory manipulator controller (as proposed in *Section 3.1. Introduction*) the most promising area was that of optimising a docking location – particularly as other tasks, such as solving manipulator forward and inverse kinematics, were routinely solvable by a manipulator controller such as ARM. This was certainly true for typical offshore manipulator configurations which were generally designed in such a way as to avoid problems encountered in more general designs, such as singularity issues, by employing restricted joint ranges.

An analysis of the different neural network types indicated that the Constraint Satisfaction network would be most appropriate for solving the docking optimisation since the problem involved solving for constraints on the ROV location, attachment leg location, manipulator kinematics, and so on (and did not explicitly require pattern matching, etc.). Some simple initial experiments were therefore conducted using this type.

As an example, a network definition was created to play the game of 'tic-tac-toe' ('noughts and crosses'), a problem proposed in the literature [Rumelhart 1986b]. This problem was chosen because of its parallels with the ROV docking problem – selecting an optimum location from a number of possible locations, given specified constraints. Following the proposed method a network definition was created (with appropriate strengths, template and network files) to represent a game state, and choose the next appropriate move. The screenshot at Figure 4.26 shows the state of the system after an initial run, where it has chosen to move first into the centre position.

```

TL-NNW Noughts and Crosses Experiment
=====

Response 000 Friendly 000 Opponent 000      Cycleno      50
units    0*0 posn    000 posn    000      updateno     42
          000          000          000      cuname      Tle1
          goodnes 9.7024
          tempera 2.0000

Empty    *** Friendly 000 Opponent 000 Friendly 070 Opponent 080
Lines    * * Doublet 0 0 Doublet 0 0 Single 7 7 Single 7 8
          ***          000          000      080      080

```

Figure 4.26 – NNW noughts and crosses network

Although the results could not be directly verified against published data, the system could demonstrably play the game. It was therefore decided at this point to conclude development and testing of the NNW software and turn instead to looking in detail at the docking optimisation problem.

#### **4.12. Thesis CD-ROM**

NNW is a large and complex application not suitable for including as an Appendix. The full source code for it (some 300 files) is therefore included on the attached CD-ROM; this source includes all project and build files necessary to compile it directly in Microsoft Visual C++ version 6. Installable/executable versions of NNW, plus data files, are also supplied on the CD ROM – for more details see the Contents list in Appendix G.

---

## CHAPTER 5: MANUAL PLANNING OF ROV DOCKING

---

### 5.1. Introduction

In order for an ROV to be able to conduct manipulative and other intervention tasks it needs to hold position at the worksite. This can either be done by the ROV itself in the water by control of its thrusters, a method known as dynamic positioning (DP) or 'station-keeping', or else the ROV can physically dock with the structure. The first half of the chapter will detail the main methods for docking, and show why the most appropriate technique for nodal weld inspection uses attachment systems such as sticky feet. This is because they provide rigidity for conducting manipulator tasks and can attach the ROV anywhere required around the node.

This approach introduces its own problems – in particular it becomes very difficult to determine where the ROV system should attach in order to get the best access to the weld. The second half of the chapter will look at how the ARM Software has been used to conduct manual planning of docking positions (and the next chapters will introduce automated planning methods).

#### 5.1.1. Working Without Docking – Dynamic Positioning

Even some of the earliest ROV systems such as Challenger AROWS had dynamic positioning, altitude and heading control, and autodepth control [Harman 1988; Russell 1990], and Sonsub claimed "Challenger's dynamic positioning capability makes it a stable work platform for precision cleaning and inspection operations when fitted with the JCV work package" [Sonsub IRST]. However, it has not been demonstrated that a typical DP system as found on most ROVs (usually based on inertial navigation) is capable of holding the vehicle sufficiently still to undertake useful work, as clearly indicated by the fact that the AROWS JCV has attachment legs to hold it still at the worksite while cleaning (see *Section 1.6. AROWS*).

External DP systems, however, have shown more promise, particularly where it is possible to arrange a network of acoustic transponders around the worksite. A prototype high resolution and high update rate acoustic DP system for ROVs was developed and demonstrated in trials on a Perry Slingsby MRV [Somers 1992]. It successfully achieved station-keeping to within 10cm RMS, and path following to within 20cm. A variation of this system, known as the Sonic High Accuracy Ranging and Positioning System (SHARPS), has been used by Imetrix Corporation for automated dam inspections with the claimed ability to position a modified ROV beside a submerged structure to  $\pm 2\text{cm}$  [Bowen 1995]. Imetrix has also demonstrated the use of a similar, modified version for nuclear vessel inspection and has claimed "Manipulation tasks are also made much easier as the need for constant joystick movements to hold the ROV steady is eliminated" [Fletcher 1995]. An ROV specially developed by Imetrix for automated control, the Talon, has a control system that has been demonstrated to automate ROV movements to better than  $\pm 10\text{cm}$  [Fletcher 1997].

A more generally applicable system is the CyberStation controller which provides an ROV station-keeping facility using a support vessel's Hydroacoustic Position Reference (HPR) system. This has been successfully demonstrated on a HiROV 3000 ROV and can also be integrated with a 3D graphical interface for visualization [Johansen 2000; Johansen 2001]. However, this system is only capable of keeping the ROV position constant to  $\pm 1\text{m}$  [Hallset 2000].

Some success has been achieved using the ROV's own cameras and a vision system linked to the ROV controller to provide station keeping. This technique has been demonstrated in simulation and in an experimental setup in a test tank using a Cartesian robot to emulate two DOF of an ROV [Lots 2000], and in preliminary pool and sea trials on an underwater vehicle [Van Der Zwaan 2001]. As far as is known, however, a commercial system is not yet available.

One of the problems of DP is the limited power and response of the ROV thrusters. Another is the low resolution of sensing systems for determining the ROV position and orientation – an alternative method proposed was "to fix the position of the ROV relative to the subsea structure by deploying 'sensor arms' from the vehicle to the structure. These arms will sense the movement of the vehicle and provide the corrective signal to move the vehicle back to its original position" [Vinsen 1988]. This method has seen some experimental development in Brazil, first on the TATUI

experimental ROV [Hsu 1994] and later on a Benthos Mark II ROV when it successfully followed a defined trajectory with an error of less than 5cm [Hsu 1999]. As far as is known, however, such a system has never been used commercially, perhaps because, since the sensor arm needs to be fixed to the structure by an electromagnet or sticky foot, the arm could be used with little modification to provide simpler direct stabilisation instead.

Even when ROV DP systems good enough for accurate and consistent ROV positioning do become generally available it is still unlikely that the ROV will form a sufficiently stable base for general manipulative work because of the movements of the manipulator and its interactions with the environment. Nonetheless, theoretical research has already been conducted into compensating for effects on the ROV from moving the manipulator [Dunnigan 1993; Koval 1994; McLain 1995]. Other work has looked at improving ROV stability through better hydrodynamic design [Baker 1990].

An alternative technique used in 1993 had the ROV holding an approximate location with its thrusters, including thrusting against the workpiece, but independently fixing an ACFM array weld inspection probe in place on the workpiece using its own suction skirt. A flexible link was used between the array and the manipulator to allow the ROV some movement without disturbing the probe during data collection [Raine 1996b; Pennison 1997].

#### **5.1.2. Docking Using Pre-Defined Attachment Points**

Physically docking with a structure takes two forms, either attaching to a prefixed docking point, or using some attachment system connected to a position of opportunity (see the next section). The most common type of fixed docking point uses a pair of tapered cones attached to the front of the ROV which engage with tapered receptacles on the structure; this is usually accompanied by a hydraulic latch to hold the ROV in place for the duration of the task. This method is very effective for pre-planned tasks, such as valve operation by ROV, where the structure has been designed and built with ROV intervention in mind. Unfortunately, particularly in the early days, every manufacturer used different, incompatible docking systems. A typical example used by Oceaneering ROVs on the Norsk Hydro Oseberg subsea system is shown at Figure 5.1. In this case, once docked, the ROV could deploy a hydraulic torque tool from the front of its toolskid, or stab a control surface-deployed umbilical into receptacles above the docking point [Renard 1988]. Very

similar systems have been used by Subsea Offshore Limited [Mair 1990] and other ROV contractors.

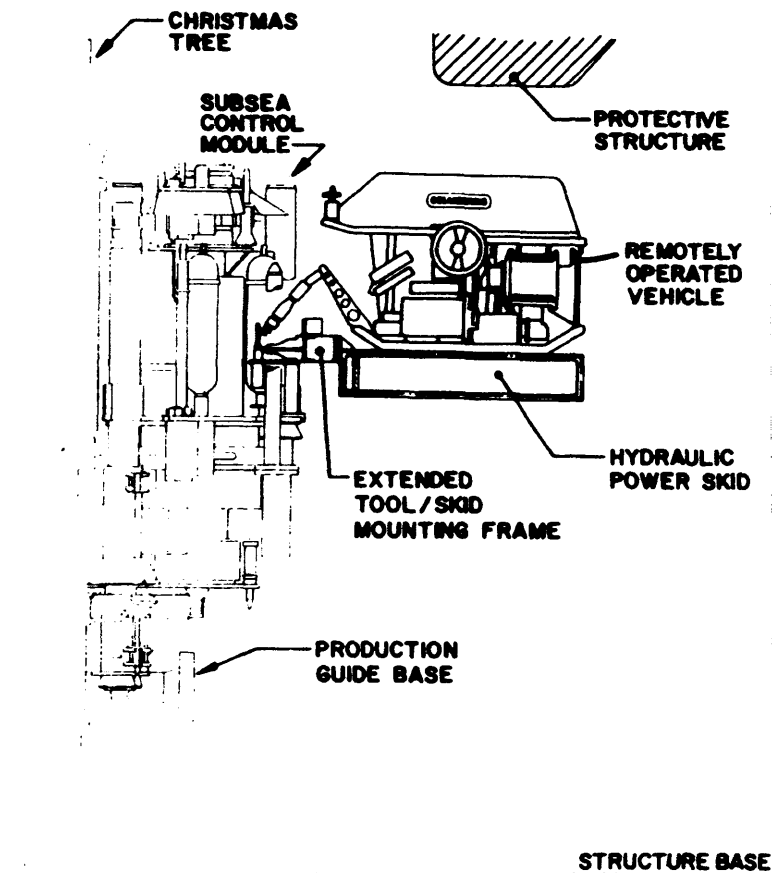


Figure 5.1 – Docking onto a proprietary template [from Renard 1988]

The benefits of standardising docking systems are obvious and in the early days such standardisation was proposed, e.g. "Contractors should continue to develop their own attachment devices but these should be designed to suit industry standard docking cones or studs attached to the subsea structures" [Vinsen 1988]; this was aimed not just at seabed systems but at nodes on jacket structures to aid nodal cleaning and inspection – see Figure 5.2. Over the next decade or so a degree of standardisation was achieved for seabed systems but docking points, standard or otherwise, have never been successfully used on jackets – this is most likely the case simply because by this time the majority of jackets were already in place, whereas, with new developments happening increasingly in deeper water, many new systems were being designed and installed on the seabed.



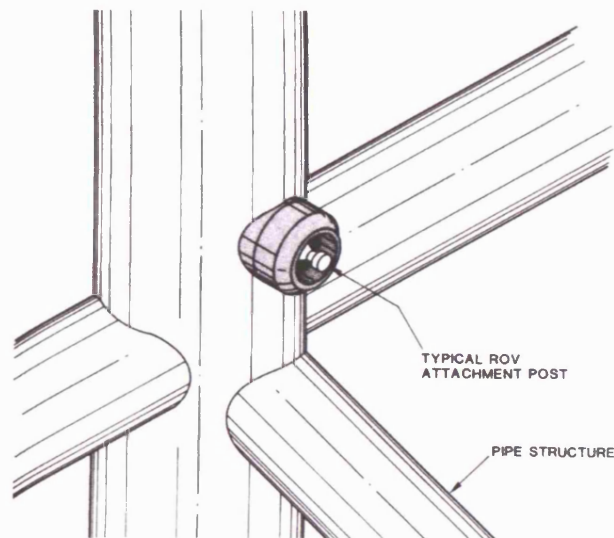


Figure 5.2 – Docking onto standard template [after Vinsen 1988]

The French Cybernétix company is currently developing the SWIMMER (Subsea Works Inspection and Maintenance with Minimum Environment ROV), a hybrid ROV/AUV (see Figure 5.3). It consists of an AUV that shuttles a conventional ROV from the surface to a subsea pre-installed docking station which is connected to a production umbilical [Chardard 2002; Ingebretsen 2002]. The vehicle swims to its underwater docking station in long range auto-navigation mode, using onboard sensors and acoustic positioning, supervised through a low speed acoustic modem.



Figure 5.3 – SWIMMER AUV (orange buoyancy) with ROV (yellow buoyancy) [from Chardard 2002]

On reaching the docking station location, the system switches to a local auto-navigation system based on 3D sonar recognition of the station. Final approach is followed by mechanical guiding of the vehicle into the station until it is physically

connected, at which point electrical and hydraulic connections are made. At this point, the onboard ROV can be operated as a conventional vehicle via a TMS on the shuttle, and using the production umbilical to communicate with the surface. A similar system, the AUto-ROV, has been proposed by Fugro-UDI [Garmulewicz 2000] but is believed to still be a paper design. Such systems certainly have great potential, and Shell has predicted that they are feasible in the short term, low-risk, and on a new field could give operational savings of \$0.5m - \$12m over five years [Van Der Veen 2000].

Cybernetix has also started early development of ALIVE (Autonomous Light Intervention VEhicle), an AUV with work (i.e. manipulative) capabilities [Chardard 2002]. It will be equipped with a telemanipulation unit controlled through an acoustic modem. It will dock itself onto an underwater structure – one that is known in detail, but which does not need to have any dedicated docking system like SWIMMER (the exact details have not yet been published).

#### **5.1.3. Docking Using Attachment Systems**

Most docking for nodal weld inspection is done using attachment legs, as described for the AROWS, REMO, ATES and ARM systems. The idea of using electromagnets has been proposed a number of times but they have not found widespread use, probably because of the problems of the uneven attachment surface (due to marine growth), and potential interference with any electromagnetic inspection techniques being used (e.g. MPI, ACFM, eddy current). A recent ROV system introduced a large docking claw underneath the toolskid for clamping onto horizontal braces; this system is the subject of later chapters.

The attachment legs used on the ARM System are fairly typical – see Figure 5.4. They have a telescopic extension from 1.3m to 1.6m, a ball-jointed, soft rubber sticky foot for attaching on to structures, and shoulder yaw and pitch joints each capable of rotating the leg through  $\pm 90^\circ$ . The leg actuators take their power from the ROV hydraulics, and the foot attaches through water being drawn out of the foot by a pump inside the toolskid.

In an ideal system, there are three attachment legs in use on an ROV system so that they form a rigid tripod arrangement that prevents the ROV from moving significantly while work is being conducted. The ARM System was designed from the outset with this philosophy in mind. Other IRM systems, such as AROWS,

REMO and ATES, are typically only equipped with one or two attachment legs and so there may be residual ROV motion induced from movement of the surrounding water (currents and swell) and from interactions of the manipulator tooling with the workpiece.



Figure 5.4 – ARM attachment leg

The amount of ROV motion is difficult to quantify (though some limited work has been done on this issue [Tisdall 1997]) and obviously depends on environmental factors and the ROV location (e.g. how close it is to the surface). It can be reduced by various techniques, such as the ATES TV Trackmeter (see *Section 2.5. ATES*), using attachment legs in conjunction with dynamic positioning (see above) or by the simple expedient of thrusting the ROV against a nearby brace or other appurtenance. The philosophy considered in the remainder of this thesis when planning how to dock on is simply to maximise the number of attachment legs in position, i.e. three if possible, two otherwise, or one if that is all that can be achieved.

## **5.2. Background to Access Simulation**

Any offshore underwater activity is very expensive because of the high backup costs, e.g. a single ROV in the water conducting a task may require a crew of seven, plus the whole system typically requires a large, expensive vessel, with its own crew of a dozen or more. For many years it was appreciated that significant costs could be saved by advance planning, particularly considering the basic issues such as whether an ROV could gain access to the required work area - “the planning of Inspection, Maintenance and Repair onto permanent subsea installed production systems are of importance in order to obtain a satisfactory production availability” [Skyberg 1988]. Before the availability of the ARM Software for conducting access simulations, however, the options for checking access were limited.

One engineer described the methods used in his project in some detail [Renard 1988], one where the structure was still being designed. These were typical of the time and interesting to look at in detail. The stages were as follows:

1. Small cardboard models – “While waiting for the future sophisticated ROV simulators to come, and before complete studies and CAD drawings could be produced, preliminary designs of the prototype structures were simply transformed into 3 dimensional cardboard models to visualize the geometry, accesses and potential obstructions for an ROV. The first simulated ROV to ‘fly’ around the structure... was a matchbox.”
2. More detailed plywood and plastic models
3. Half size cardboard models
4. Full size ROV mock-ups made of scaffolding tubulars and joints – “... were suspended to the hook of a crane and ‘flown’ in and out of the structure openings to each work location”
5. Shallow water testing using a prototype structure

Even at this time, computers were starting to be used to simulate the motions of ROVs [Primrose 1988; Broome 1988] and it was appreciated that computers could be involved in access checking. This was particularly true once a project was at the stage where the structure had been fully modelled in CAD - “for access verification at an early stage of the subsea production system development Computer Aided Design has proved to be a satisfactory tool” [Skyberg 1988]. Essentially a model of the ROV and a model of the structure are both held in the CAD software and the user is able to move the ROV

model around the structure and check access, clearances, etc. More sophisticated approaches were soon proposed: "Based on numerical models, motions of the vehicle, manipulator or tools at the work site can be animated. By introducing the surrounding structures at the work site, both accessibility and obstacle avoidance can be studied" [Sortland 1990].

Nonetheless, it is of interest to note that at the start of the ARM Project, before the ARM Software had been developed, one of the first things done by Slingsby was to make cardboard models of structures and manipulators in order to check manipulator access capabilities and hence help design the manipulator configuration. Perhaps more surprising, and despite the rapid advances in computer simulation, is the fact that even today a number of ROV operating companies check ROV access to subsea structures by flying an ROV off a crane around the structure before it is deployed into the water.

Notwithstanding the above, it is now possible to test a complete intervention task in simulation, not just in terms of access checking but also considering the handling of the vehicle in the ambient water conditions (current, swell, etc.) [Larkum 2000; Larkum 2002]; "3D model testing and simulator training comes highly recommended in the process of verifying access and purpose testing the application in question" [Ingebretsen 2002].

### **5.3. ARM Access Simulation**

As the development of the ARM System progressed it became clear that the original high demand for weld inspection that it had been designed for had receded. This was largely due to the development of new structural integrity simulations that allowed platform operators to demonstrate, through finite element analyses and similar techniques, that a particular structure had sufficient structural integrity to achieve certification, without requiring large amounts of inspection.

Platform operators began to develop new inspection programmes, and obtained dispensations from their Certifying Authorities, in order that they could maximise the period between weld inspections [Raine 1996a; Raine 1997; Pennison 1997]. Two main philosophies predominated. In the first philosophy, weld inspection on a small number of node welds was replaced by Flooded Member Detection (FMD) on a much larger scale. This could be carried out with the use of an ROV and a radiation FMD system which did not require cleaning of the member or accurate placing of the FMD tool. Only if flooding was detected would detailed weld inspection be carried out with the



necessary cleaning, and detailed application of a weld inspection tool (preferably by ROV). The other philosophy was to carry out all but weld inspection over a four year cycle and then deploy divers in the final year of a five year cycle. If the structures were located in mixed depths of water (suitable for air diving and saturation diving) then a saturation diving team would be deployed.

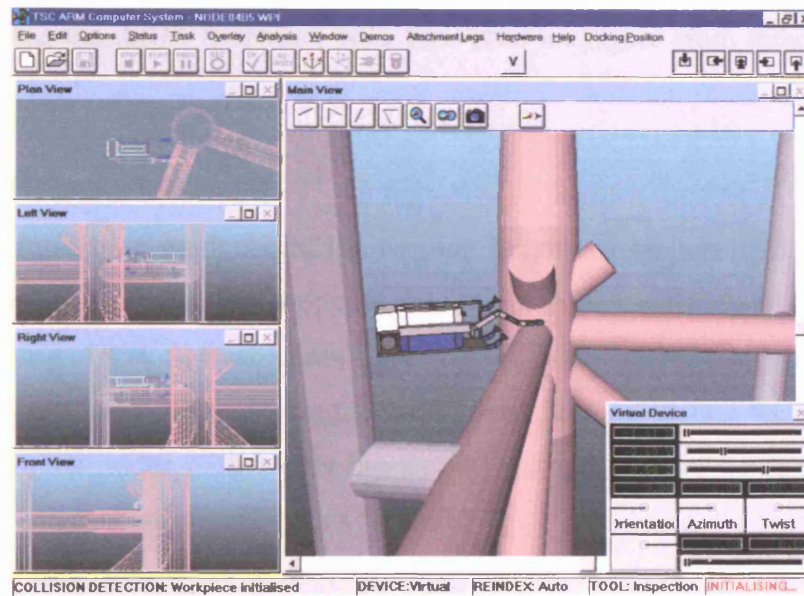


Figure 5.5 – ARM Software being used for Access Simulation

Inspection continued, but at such low volumes that it could be done by means other than automated ROV deployment, e.g. by divers when on site for other tasks (hence at low extra cost) or by conventional ROVs (since the low work rate was not significant for a small amount of inspection). Nonetheless, even as demand for the ARM hardware dried up, the ARM Software was increasingly in demand as a simulation and task planning tool (see Figure 5.5): "Using the ARM simulation package it is possible to decide the suitability of a particular ROV manipulator combination to inspect nodal welds on a particular platform" [Pennison 1997].

#### 5.4. Development of ARM Docking Planning

Most of the planning work in the literature (as described above) was simply access checking, i.e. to see if the ROV could physically fit in the workpiece environment. However, as ARM developed it became more sophisticated and, from 1995, it was able to model attachment legs in detail, considering their kinematics, and determine whether a particular leg could attach at a particular position on the workpiece as specified by the operator. During 1996 and 1997 this was extended so that the software could calculate

whether a particular leg could attach given a specified ROV location (initially just considering the closest tubular, but later considering all tubulars in turn to see if a successful attachment could be made). From this point, ARM was able not just to conduct an access check, but to plan a docking location for the ROV system – the type of docking planning that is the subject of this thesis. The remainder of this chapter will describe the kind of access checks and docking planning tasks that were solved manually using the ARM Software, with four brief examples from major commercial jobs.

#### **5.4.1. Texaco Node Visualisation**

The first modelling work conducted was for Texaco in August 1995, looking at the ACFM inspection of nodal welds on the Tartan Alpha platform in the North Sea. The existing ARM Software was used for the node modelling, but a number of enhancements were made during the work (particularly allowing for the building of more complex nodes with longer braces and fixtures such as caissons and risers). Example screenshots are at Figures 5.6 and 5.7.

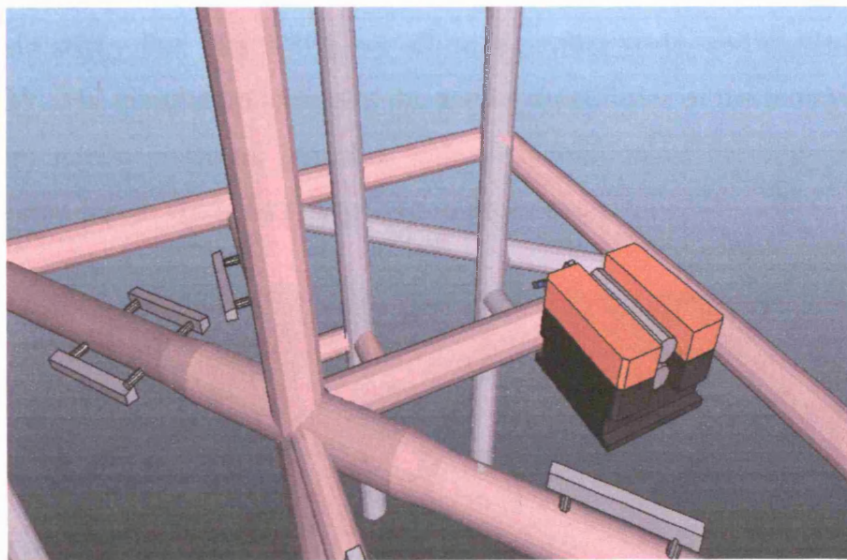


Figure 5.6 – Tartan Alpha: the riser of interest and its mounting brace are directly in front of the ROV

The Tartan Alpha work remained largely an access simulation job – access for the ROV to the vicinity of the work areas was considered, and the results were primarily the 3D graphical displays of the nodes which were used by Texaco to help them visualise and plan the ROV tasks.

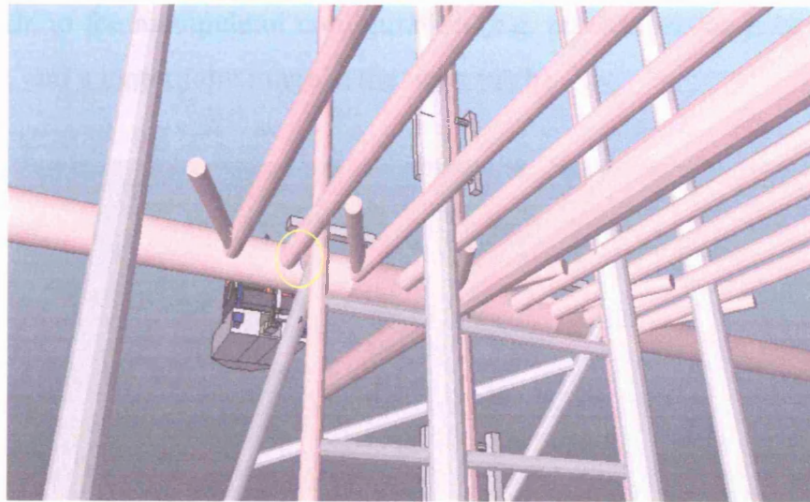


Figure 5.7 – Access checking on an almost hidden weld (highlighted ahead of the ROV)

#### 5.4.2. RACAL Manipulator Evaluation

The second job was the simulation of a proposed small inspection ROV toolskid and manipulator (see Figure 5.8) for RACAL. This was really the first manual docking planning work since it fully considered the attachment of the ROV system to the workpiece (as did all later work). The toolskid had a rotating/extending manipulator boom and sticky feet like ARM, but all on a smaller scale, and carried on a small Seal ROV. The simulation looked at the access capabilities of the toolskid on the Elf Claymore Alpha platform, but also with consideration of the way in which the manipulator design could be changed to improve access.

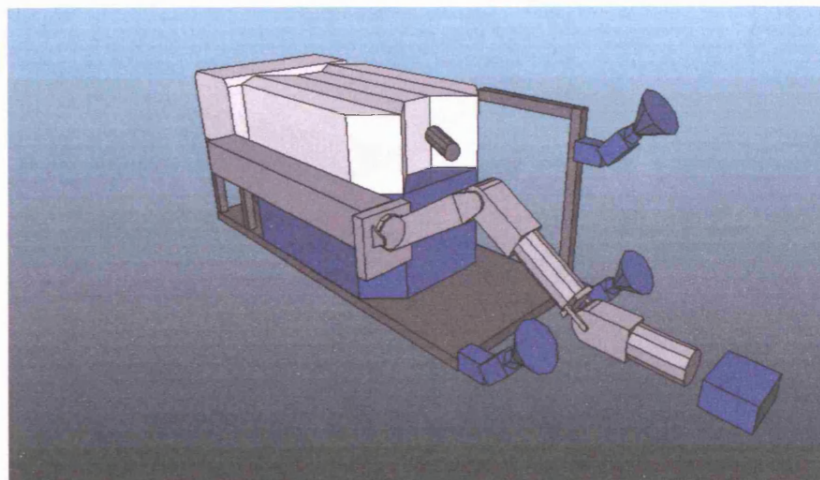


Figure 5.8 – Proposed RACAL toolskid and manipulator design

The system is shown docked on a node in Figure 5.9. The simulation results showed that the system could access the nodes considered so long as a number of changes



were made to the manipulator configuration (e.g. an extra rotate joint at the end of the wrist, and a larger joint range at the wrist pitch).

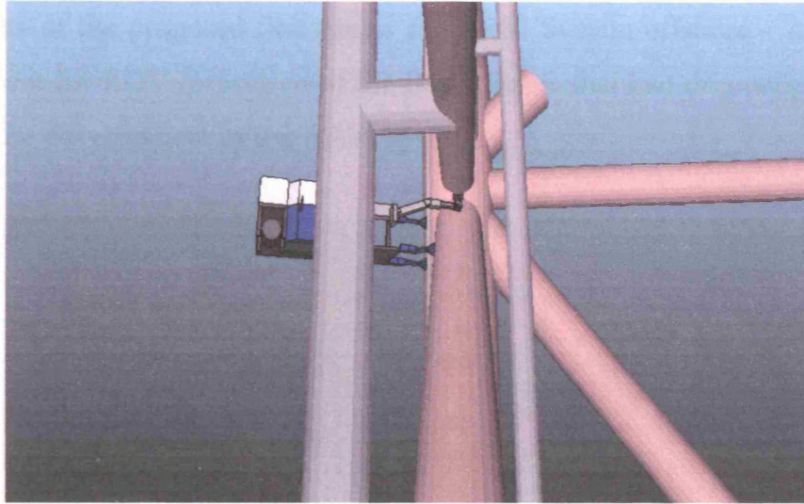


Figure 5.9 – RACAL system inspecting the underside (6 o'clock) on a nodal brace

The system was redesigned, largely in line with the recommendations from the simulation work. It was built in a very short space of time by Trittech International Limited (see Figure 5.10), in conjunction with Hydro-Lek Limited, and went offshore in 1996 [Raine 1997]. It conducted the inspection programme for Elf with some, limited success [Pennison 1997] – the problems were largely due to equipment failure rather than faults in the toolskid or manipulator configurations. Nonetheless, it will be seen in Chapter 8 that Elf looked again at system requirements for this inspection work.

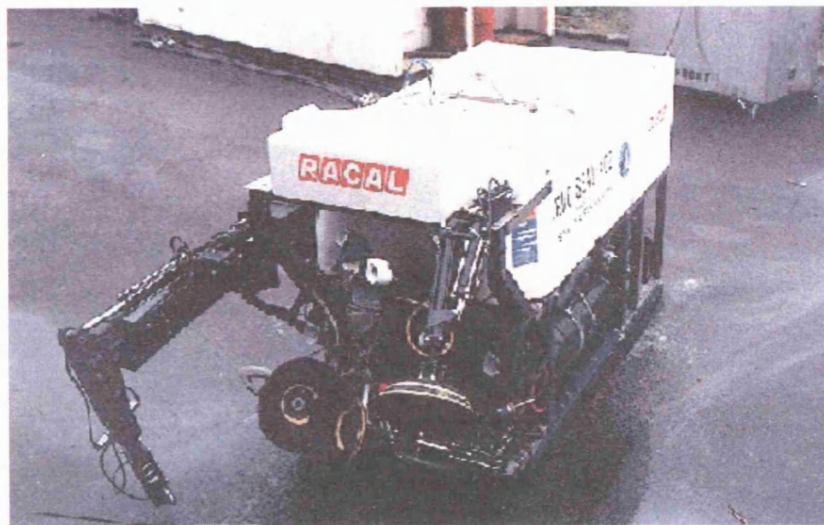


Figure 5.10 – RACAL system manufactured by Trittech [courtesy TSC Ltd]

#### **5.4.3. Mobil Docking Planning**

The third job involved access simulation for the ARM System on to five nodes on Mobil's Beryl Bravo platform. This was an important job as it was to be conducted in advance of the proposed first use of the ARM System offshore – it was Mobil's requirement for ROV inspection of the Beryl Bravo that had originally led to Mobil funding the development of the ARM System.

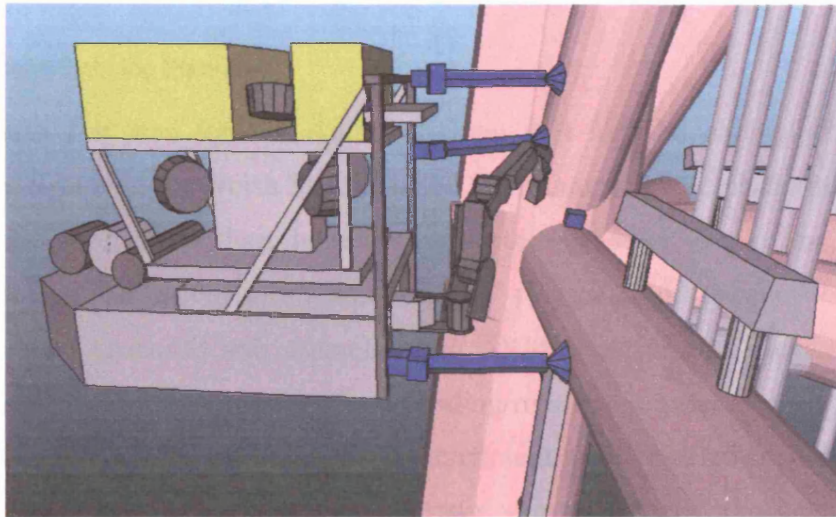


Figure 5.11 – The ARM System on node 3A2, inspecting the 12 o'clock position

An interim simulation had been conducted in 1994 but the conclusions were tentative due to the incomplete nature of the ARM Software at the time. Furthermore, following the ARM 3 trials in April 1996 (see Chapter 2) a number of significant improvements had been made to the Software.

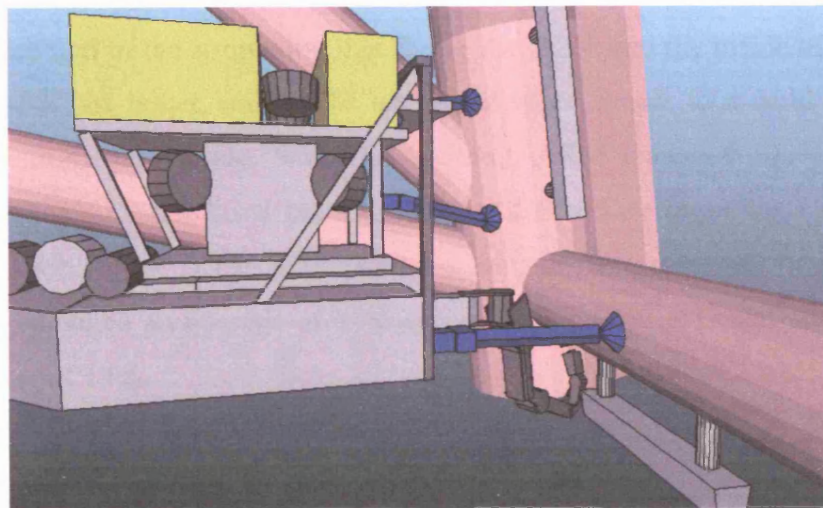


Figure 5.12 – The ARM System on the inside of 6A3, inspecting the 6 o'clock position

The new simulation results showed that the ARM System was capable of inspecting most of the welds considered, and certainly as much as could be inspected by diver. Nonetheless, due to the general changes in inspection philosophy, as discussed above, combined with particular internal political and financial control changes within Mobil, the Beryl Bravo work was cancelled and the ARM System never did go offshore. The ARM System is shown docked onto two of the Beryl Bravo nodes in Figures 5.11 and 5.12.

#### **5.4.4. Amoco Docking Planning**

The fourth job was for the Amoco Leman and Indefatigable platforms in the Southern Sector of the North Sea for DSND Oceantech. These platforms are smaller and in shallower water than those further north considered above, and so the work involved considering whether a smaller ROV system could conduct the work. The small system proposed was essentially an ARM System with smaller dimensions (i.e. same toolskid design with extending/rotating manipulator boom plus a 'goalpost' at the front mounting three attachment arms) carried by a Seal ROV (as per the RACAL job above). An important aim was also to compare the results achieved by:

1. a fixed manipulator, i.e. the same as a standard ROV manipulator arrangement
2. using the same manipulator mounted on the proposed toolskid (to see the benefit of the extending/rotating mount)
3. the complete (but much larger) ARM System, as a reference

It was assumed in the simulation that flying the ROV into the inside of the structure should be a last resort, only to be attempted when access to a weld could not be achieved from the outside. With this in mind, ARM managed 100% access from outside (making use of two probe offsets, and therefore requiring a total of three dives) – see Figure 5.13. Also from the outside, the manipulator on the small toolskid managed an average of 48% access, while the fixed manipulator managed an average of 14%.

If, however, flying inside were regarded as acceptable then the ARM system could do without the probe offset and could achieve 100% access to each weld in one dive to each node. Therefore, if these nodes are typical of ones that could be found within the same platform then ARM could access 100% of the four welds in one and the



same dive. Making use of the inside, the manipulator on the small toolskid managed an average of 91% access (see Figure 5.14), while the fixed manipulator managed an average of 45%.

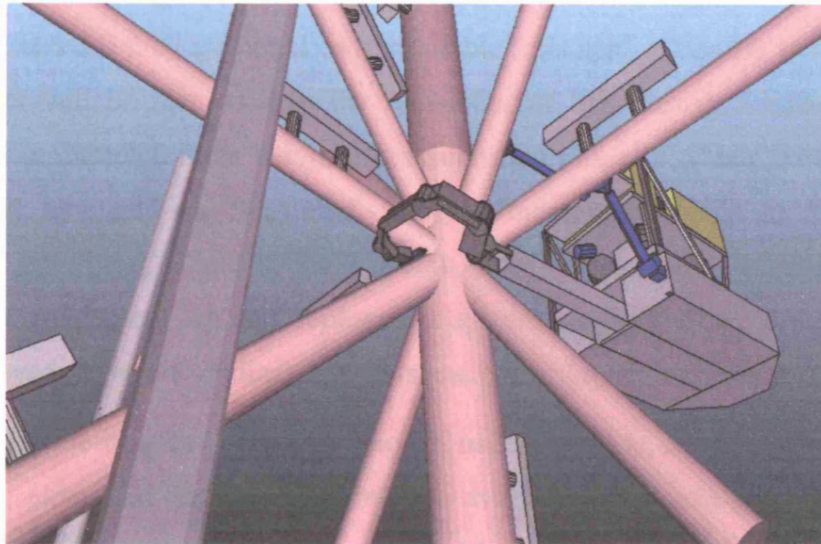


Figure 5.13 – ARM Inspecting an internal nodal weld from outside the platform

These results clearly vindicated the original ARM design decisions with regard to the necessity of having a rotating and/or extending boom for the manipulator. However, on these smaller platforms at least, and if flying inside was acceptable, good access could still be achieved by a smaller ROV system so long as it had its own rotating/extending boom.

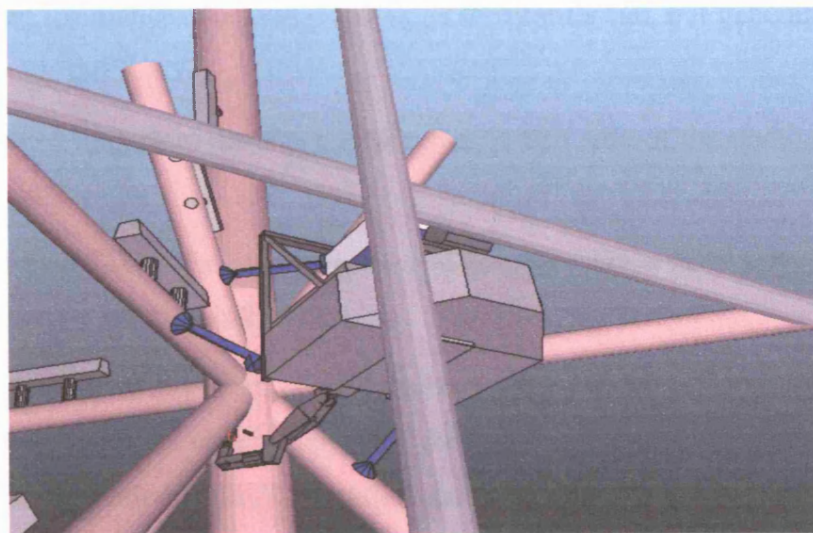


Figure 5.14 – Seal ROV inspecting a weld from the edge of the conductor guide frame

## **5.5. Simulation Process**

The process of conducting manual access checks for ROV/manipulator combinations follows a standard procedure that will be described here. It should be noted that before any simulation can begin, models of the workpiece, ROV, toolskid and manipulator need to be created if they are not already available (this applies equally to the automated methods that will be discussed in the remainder of this thesis). Similarly, after the simulation is completed there is typically a reporting phase in which the access results are compiled, tabulated and illustrated – where a manipulator could not access a weld, the reasons are given; where it could access, information is given on the best ROV position, the attachment leg and deployment system configurations, and the degree of access achieved (in terms of clock positions and percentage of total weld reached).

The process of creating the workpiece model, usually a node, involves interpretation of the platform plans and any other information available (such as photographic stills or video). The model is then built in the ARM software using a combination of primitive shapes, primarily cylinders and cuboids, to represent the chord, braces, pipes (e.g. risers and caissons), and miscellaneous brackets often found on jacket nodes. In addition, certain predefined components are available in the ARM software that can be added directly, as required, such as anodes and name plates. Also, ARM has an extensive database of ROVs, toolskids and manipulators but when a new one is required it can be constructed from primitives, much like workpieces, but the description is written directly into a text file (i.e. there is no graphical interface for building it).

The procedure for manual docking planning is fairly informal, but generally takes place as follows:

1. Choose an appropriate start position (i.e. which side of the node) and approach direction.
2. Place the ROV at the start position and then move it as necessary to prevent any collisions with the workpiece.
3. Check to see if any or all of the attachment legs can attach at this ROV position. If not enough can attach (generally the requirement is all of them) then change the ROV position and/or orientation and try again.
4. Choose suitable start and end positions on the weld for the inspection task. The manipulator is then deployed and the probe stepped around the weld in simulation (the ARM Software creates a model of the weld metal that is

superimposed on the workpiece and used to determine the precise position and angle required of the probe. Since the software includes full forward and inverse kinematic models of each manipulator this provides a very accurate measurement of possible weld access, taking into account collision detection of the manipulator against itself, the ROV and toolskid, and the workpiece components and fixtures). If the simulated inspection fails, try closer and smaller segments of weld until the inspection has succeeded or all segments have been tried. This simulated inspection is a very time consuming process since it is conducted in real-time, and therefore can take some minutes for each partial check.<sup>1</sup>

5. If the complete weld cannot be inspected with the manipulator in its default position, as is invariably the case, estimate a better deployment system setting (e.g. extending and/or rotating the boom or whatever is appropriate) and repeat from 4.
6. If the weld has not yet been fully inspected, having tried a number of likely deployment system settings, then adjust the ROV position as necessary and repeat from 2.
7. If the weld has not yet been fully inspected, then consider a different approach position and direction (e.g. from the other side of the node), i.e. repeat from 1.
8. If a sufficient number of permutations have been tried (this is entirely down to the discretion, experience and patience of the operator) try other means of improving access, e.g. offset the manipulator from the deployment system, offset the probe from the manipulator wrist, change the orientation of the probe on the manipulator, change to a different manipulator (on a two manipulator system), etc. – and repeat from 1 for each option.
9. If it is required to access different parts of the weld (the default is the chord toe, the edge touching the chord, but the brace toe and weld cap often also need to be inspected) then select the new weld path, and repeat all from 1 (though the ROV configuration chosen this time will obviously be informed to some extent by successful access achieved for any previous weld paths).

---

<sup>1</sup> After the automated docking work was completed, the kinematic path check it used was made available for manual use so that this inspection simulation could be replaced with the faster kinematic check.

It is very obvious that the whole process of manual docking planning is a long and laborious one without any indication of whether an initial location (necessarily chosen somewhat arbitrarily) was a good position or not until an extensive access simulation was conducted. It could take an experienced operator some days to plan the docking on a single complex node, and clearly indicated the requirement for an automated docking planning system such as those described in detail in the remainder of this thesis.

Although in some ways the proposed solution will inevitably be closely tied to the ARM Software to avoid unnecessary duplication of work (since this is already used to define the problem in all its complexity, e.g. workpiece details, manipulator and attachment leg kinematics, etc.), it is clearly a requirement that the resulting system be applicable to planning the docking not just of the ARM System (i.e. the Slingsby hardware) but of any of the many ROV/manipulator combination systems considered in this and earlier chapters.

---

## CHAPTER 6:

# DOCKING PLANNING USING NEURAL NETWORKS

---

### 6.1. Introduction

As has been described above (see Chapters 3 and 4), a general purpose neural network software application was created as part of this thesis work. This chapter will describe the use of the neural network software to help plan docking locations in an automated way.

The plan was as follows:

1. Create simple, test docking problem definition files by hand
2. Read these into the neural network software to solve
3. Verify the results were valid
4. If this method proved feasible, adapt the ARM software to create docking problem definition files using its existing simulation facilities
5. Read these into the neural network software to solve
6. Verify the results were valid and, where feasible, compare with the results achieved by a human operator

The remainder of this chapter will describe the testing of manual docking definitions (i.e. 1-3 above) and the next chapter will describe the development and testing of docking definitions created through ARM processing (i.e. 4-6 above).

### 6.2. First Manual Scenario (Coincident Attachment Legs), Schema Model

The first scenario to be defined and tried in the neural network software was a very basic one:

1. Assume a simple node: a vertical chord (radius 1m) with a single horizontal intersecting brace (radius 0.5m) arranged so that the brace direction is perpendicular to the ROV primary axis.



2. Assume that the required access point is at the centre of the brace where it intersects the chord (to represent approximately typical access to the whole weld around the brace where it intersects the chord).
3. Consider possible ('candidate') ROV docking positions as being on a coarse grid of points 1m apart, centred on the required access point, extending from -1 to 1 in x, y and z (see Figure 6.1).
4. Discard any points inside the chord or brace.
5. Assume that the manipulator origin (the shoulder mounting plate) is located at the ROV origin (and that it is not attached to a rotating or extending boom, or similar system).
6. Discard any points that are too close or too far from the required access point for practical manipulator interaction, i.e. if not within working manipulator reach – see below.
7. 'Score' remaining points in terms of how much working reach the manipulator retained, the criterion being how close it was to the middle of its reach range (considered to be 0.5m to 2.0m for the Slingsby ARM inspection manipulator under consideration) – see below.

These items produced a result with a large number of valid ROV positions, and so further criteria were added in order to distinguish between a number of valid ROV positions:

8. Assume that the ROV has one attachment leg, *and that it is connected to the ROV at its origin* (this will be changed in the next scenario). Discard any points that are too close or too far from the chord or brace surfaces for the attachment leg to attach.
9. 'Score' remaining points (for details see below) in terms of how well the attachment leg is fixed, the criteria being how close it is to the middle of its extension range (considered to be 1.3m to 1.6m for the Slingsby ARM attachment legs under consideration at this stage) when reaching the surface of either the chord or brace.

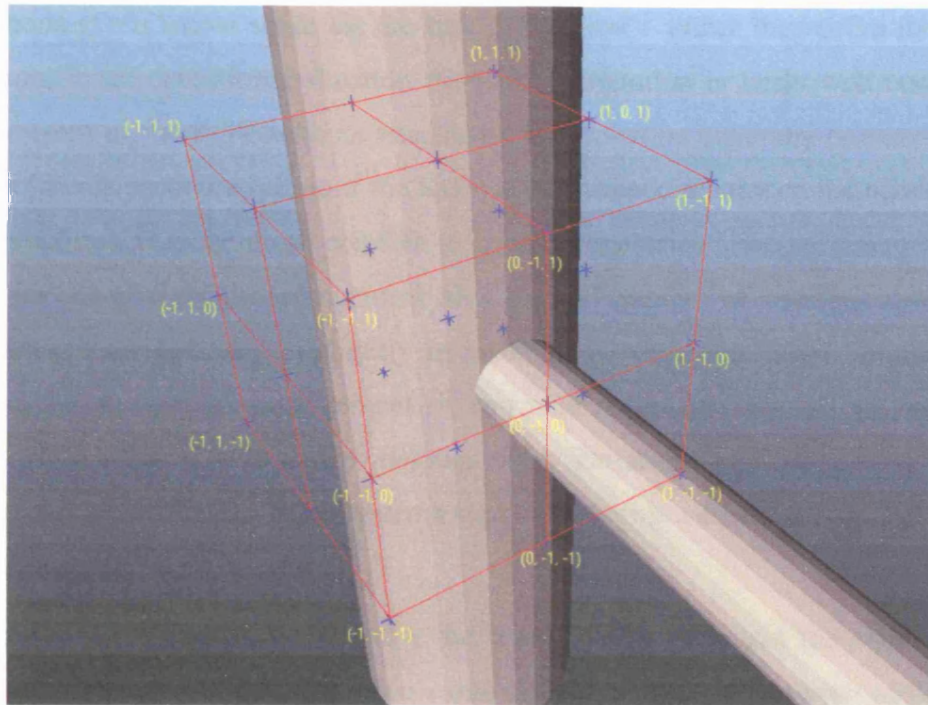


Figure 6.1 – Illustration of the grid of candidate positions used

The type of scenario envisaged was entered into the ARM software purely as a means of illustration, and this is shown at Figure 6.2.

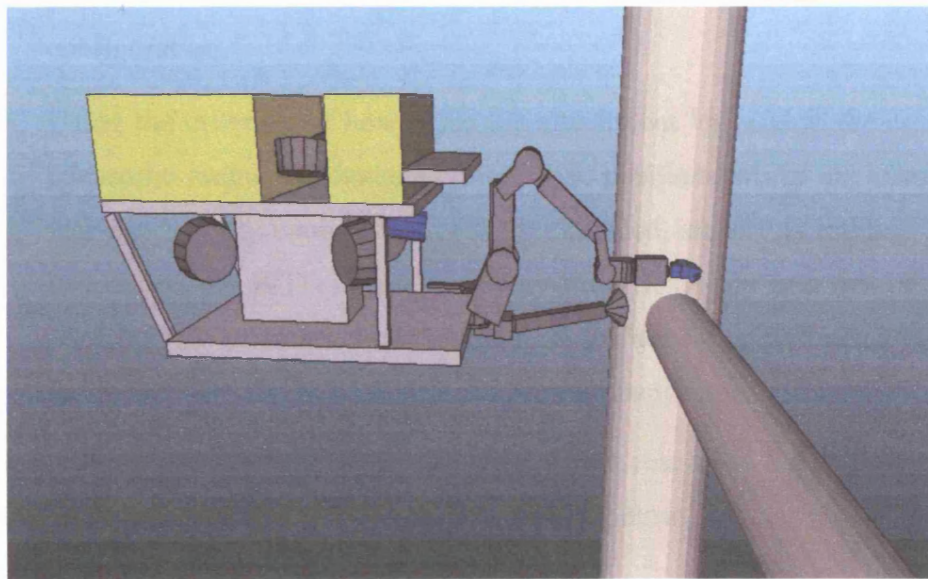


Figure 6.2 – Illustration of the First Scenario (coincident single leg and manipulator)

Note that it was assumed (see item 1.) that the ROV system was 'straight on' to the node (either from the 'left' in Figure 6.1, as illustrated in Figure 6.2, or 180 degrees around from the 'right'). This was a simplification that made the solution of the problem considerably more straightforward than if the orientation was also considered. None of

the methods given below solve for the best orientation – rather they solve for a given orientation. In an operational situation the ROV orientation is fairly well constrained, since the front of the ROV with its attachment legs must be generally oriented towards the node (this is particularly true if the ROV is constrained to stay on the outside of the jacket structure). It is therefore possible to consider the orientation sufficiently well by solving for the straight on arrangement plus a small number of arrangements that are nearly straight on but vary in slightly in heading. For example, once the system can solve for the straight on arrangement, it can also solve for, say, 10 degrees left, 5 degrees left, 5 degrees right and 10 degrees right. Variations in pitch and roll were not considered since workclass ROV systems tend to be highly constrained in these by their buoyancy arrangement.

The purpose of calculating a score for the manipulator reach and the attachment leg extension was to provide a quantitative value to help distinguish between positions that were otherwise equivalent. The reasoning was as follows:

- Using the criterion of how close the manipulator was to the middle of its working range was designed to penalise positions where the manipulator was nearly fully stowed or fully extended, since this implied that once in location the manipulator would have very little flexibility in terms of changing configuration.
- Using the criterion of how close the attachment leg was to the centre of its telescopic range was designed to penalise positions where the telescopic leg was nearly fully compressed or fully extended, since this implied that once in location the ROV would have very little flexibility in terms of changing position (the orientation of the attached surface is ignored since the suction feet generally have very flexible ball-joint couplings).

These are just approximations to the actual manipulator reach and leg attachment ability of a system, and will be refined in the work described below.

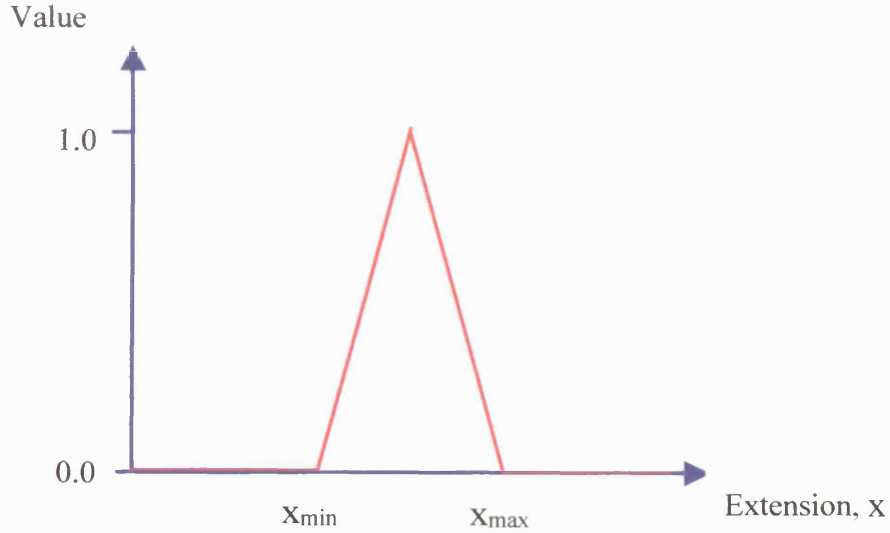


Figure 6.3 – Determining a score value from manipulator or attachment leg extension

The actual score value was calculated very simply as the absolute ratio between 'the difference of the extension value from the average/midpoint value' and 'half the working extension range' – this was then subtracted from one, so that a high value represented a good extension position, i.e. near its midrange. This is illustrated in Figure 6.3 and can be represented as follows:

$$Value = 0 \quad 0 < x < x_{min}, x_{max} < x \quad - Eq. 6.1$$

$$Value = 1 - \left| \frac{x - \frac{1}{2}(x_{max} + x_{min})}{\frac{1}{2}(x_{max} - x_{min})} \right| = 1 - \left| \frac{2x - x_{max} - x_{min}}{x_{max} - x_{min}} \right| \quad - Eq. 6.2$$

$$= 1 - \left| 1 + \frac{2(x - x_{max})}{x_{max} - x_{min}} \right| \quad x_{min} < x < x_{max} \quad - Eq. 6.3$$

Using this scheme a table of candidate positions was created, and then each position checked against the listed criteria and either rejected, or given an appropriate score – see Table 6.1. Once a criterion was found that led to a position being rejected, the other criteria were not calculated for that candidate – with the exception of the attachment leg distance, since it was valid to attach either to the chord or the brace.

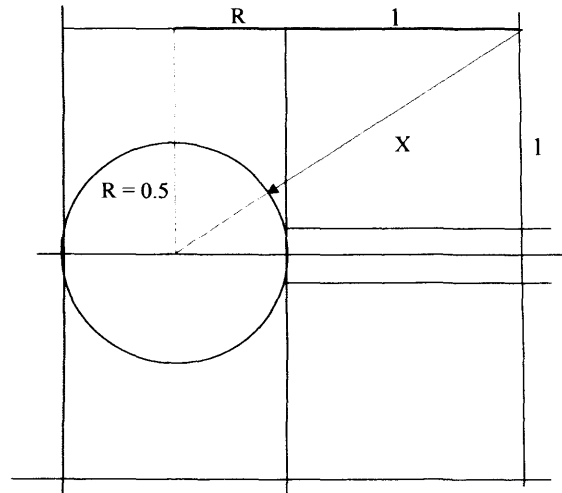
Candidate	Position (X, Y, Z)	Outside Chord?	Outside Brace?	Manipulator Reach Value <sup>2</sup>	Leg Distance to Chord Value <sup>3</sup> (radius R = 0.5m)	Leg Distance to Brace Value <sup>3</sup> (radius r = 0.25m)
1	0, 0, 0	✗				
2	0, 0, 1	✗				
3	0, 1, 0	✗ <sup>1</sup>				
4	0, 1, 1	✗				
5	1, 0, 0	✓	✓		0	0
6	1, 0, 1	✓	✓	0.78 ( $x=\sqrt{2}$ )	0	0.24 ( $x=\sqrt{2}-r$ =1.16)
7	1, 1, 0	✓	✓		0	0
8	1, 1, 1	✓	✓		0	✗
9	-1, 0, 0	✓	✓		0	0
10	-1, 0, 1	✓	✓	0.78	0	0.24
11	-1, 1, 0	✓	✓		0	✗
12	-1, 1, 1	✓	✓		0	✗
13	0, -1, 0	✓	✗			
14	0, -1, 1	✓	✓		0	0
15	1, -1, 0	✓	✓	0.78	0.8 ( $x=1.3$ ) <sup>4</sup>	
16	1, -1, 1	✓	✓	0.36 ( $x=\sqrt{3}$ )	0.8	
17	0, 0, -1	✗				
18	0, 1, -1	✗				
19	1, 0, -1	✓	✓	0.78	0	0.24
20	1, 1, -1	✓	✓		0	✗
21	1, -1, -1	✓	✓	0.36	0.8	
22	0, -1, -1	✓	✓		0	0
23	-1, 0, -1	✓	✓	0.78	0	0.24
24	-1, 1, -1	✓	✓		0	✗
25	-1, -1, 0	✓	✓	0.78	0.8	
26	-1, -1, 1	✓	✓	0.36	0.8	
27	-1, -1, -1	✓	✓	0.36	0.8	

Table 6.1 – Scoring docking positions manually, first scenario

Key: ✓ = valid position, ✕ = invalid or inaccessible position

Notes:

- <sup>1</sup> Points on the chord surface are considered to be inside the chord
- <sup>2</sup> For manipulator reach,  $x_{\min} = 0.5\text{m}$ ,  $x_{\max} = 2.0\text{m}$
- <sup>3</sup> For attachment leg distance,  $x_{\min} = 1.1\text{m}$ ,  $x_{\max} = 1.6\text{m}$  (the minimum extension of the Slingsby legs is 1.3m, but by inclining the leg at  $30^\circ$  – the maximum rotation of the ball joint at the foot – they could be used to attach to a surface as close as  $1.3\cos 30 = 1.1\text{m}$  away).
- <sup>4</sup>  $x = \sqrt{((1+R)^2 + 1^2)} - R = 1.3$   
(see attached sketch for reasoning)



From this manual scoring of docking positions there remained out of the original 27 possible locations, 10 candidate positions that satisfied the different criteria (marked with yellow shading in Table 6.1). The next stage was to create a neural network definition that included these 10 positions along with their scores for manipulator reach and attachment leg extension in order for the neural network software to be able to make the best selection. Of course, at this stage, the example was artificially simple so that it could be created manually in a relatively straightforward way – but if the system worked then this would indicate that the whole approach could be automated and hence applied to situations of increasing complexity.

A network definition was created with appropriate files as follows:

- `Rov.str` (network strengths): This gave initial values for all main parameters (based on the values found successful for the tic-tac-toe problem initially – see *Section 4.11. Early Experimentation*), the names of other files required, and a list of the names of candidate positions. See Table 6.2. The names were created directly from their co-ordinates, e.g. (1.0, 0.0, 1.0) became '+10+1', (-1.0, 0, 1.0) became '-10+1' etc.
- `Rov.tem` (screen layout) and `Rov.loo` (variables layout): These defined some initial parameters and indicated where unit values were to be displayed on the screen. See Appendix D.

- `Rov.net` (network definition): This defined a symmetrical network with 12 units, 12 inputs and 12 biases, and with 12 updates per cycle – these were for the 10 candidate positions, plus the two criteria values (manipulator reach and attachment leg extension). See Appendix D.
- `Rov.wts` (unit weights): This defined the weights applied to each unit in the form of a 12 x 12 array. See Table 6.3.

```
set dlevel 1
get network rov.net
get weights rov.wts
set mode clamp 1
set param estr 1.0
set param istr 0.2
set ncyc 50
get unames +10+1 -10+1 +1-10 +1-1+1 +10-1 +1-1-1 -10-1 -1-10 -1-1+1
-1-1-1 Acs SF
```

Table 6.2 – Network strengths and unit names in `Rov.str` file

For this First Manual Scenario the Schema Model, the simplest type of constraint satisfaction network, was used. As has been seen, for a CS network (see *Section 4.3.1. CS Theory*):

$$goodness_i = \sum_j w_{ij} a_j + input_i a_i + bias_i a_i = net_i a_i \quad - Eq. 6.4$$

$$\text{where } net_i = \sum_j w_{ij} a_j + input_i + bias_i \quad - Eq. 6.5$$

= net input into a unit

That is, goodness is maximised if each activation is increased when the net input is positive (and vice versa). Therefore the activation for a particular grid location will increase from the units it is linked to multiplied by the weight of the link. There are no dynamic inputs in this system (the requirements do not change during the processing) and so activation of all units will initially be just their bias. Then during processing, the activation of a grid choice  $a_{x,y,z}$  will depend on its initial bias plus any links from connected units (a simpler case than the general arrangement shown in Figure 4.10).

It is worth looking at the unit weights chosen in the `Rov.wts` file in more detail. For each of the possible candidate positions there is a negative (-1.0) value from each of the other candidates so that they are mutually exclusive, i.e. as the strength for one position increases so it tends to push down the strength of any other selection – this is done so as

to produce a best ('winner takes all') solution. For each position there is then a manipulator reach value and an attachment leg extension value taken directly from the manual docking table above. Finally, there is a small bias (0.1), highlighted in yellow, applied to each position so as to 'kick off' the selection, i.e. each position has a starting value as the competition begins so that the system is essentially unstable initially and will have to start responding to the various unit activations.

0.00	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.78	0.24
-1.0	0.00	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.78	0.24
-1.0	-1.0	0.00	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.78	0.80
-1.0	-1.0	-1.0	0.00	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.36	0.80
-1.0	-1.0	-1.0	-1.0	0.00	-1.0	-1.0	-1.0	-1.0	-1.0	0.78	0.24
-1.0	-1.0	-1.0	-1.0	-1.0	0.00	-1.0	-1.0	-1.0	-1.0	0.36	0.80
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.00	-1.0	-1.0	-1.0	0.78	0.24
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.00	-1.0	-1.0	0.78	0.80
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.00	-1.0	0.36	0.80
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.00	0.36	0.80
0.78	0.78	0.78	0.36	0.78	0.36	0.78	0.78	0.36	0.36	0.00	0.00
0.24	0.24	0.80	0.80	0.24	0.80	0.24	0.80	0.80	0.80	0.00	0.00
0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.00	0.00

Table 6.3 – Network weights in `ROV.wts` file

This network definition was then run in NNW to see if it could make a selection of the best position. In most runs it made a selection fairly quickly, within about 50 cycles. In Figure 6.4 it has run 50 cycles and positions 3 and 8 already have high activation values ('\*' = 1.0, and '9' = 0.9, respectively) and the overall goodness figure is 2.1562; position 10 is also slightly activated (0.2). In Figure 6.5 it has run 100 cycles and positions 3 and 8 both now have an activation of 1.0, and the goodness is 2.3598; position 10 is no longer activated at all. In Figure 6.6 it has run 150 cycles – the activation levels are fixed at 1.0 and the goodness has reached a plateau at 2.3600. Further cycles produce no change in the system, it has fully settled. When conducting a large number of runs it can be seen that very occasionally other positions are selected, but positions 3 and 8 are selected in the vast majority of runs.



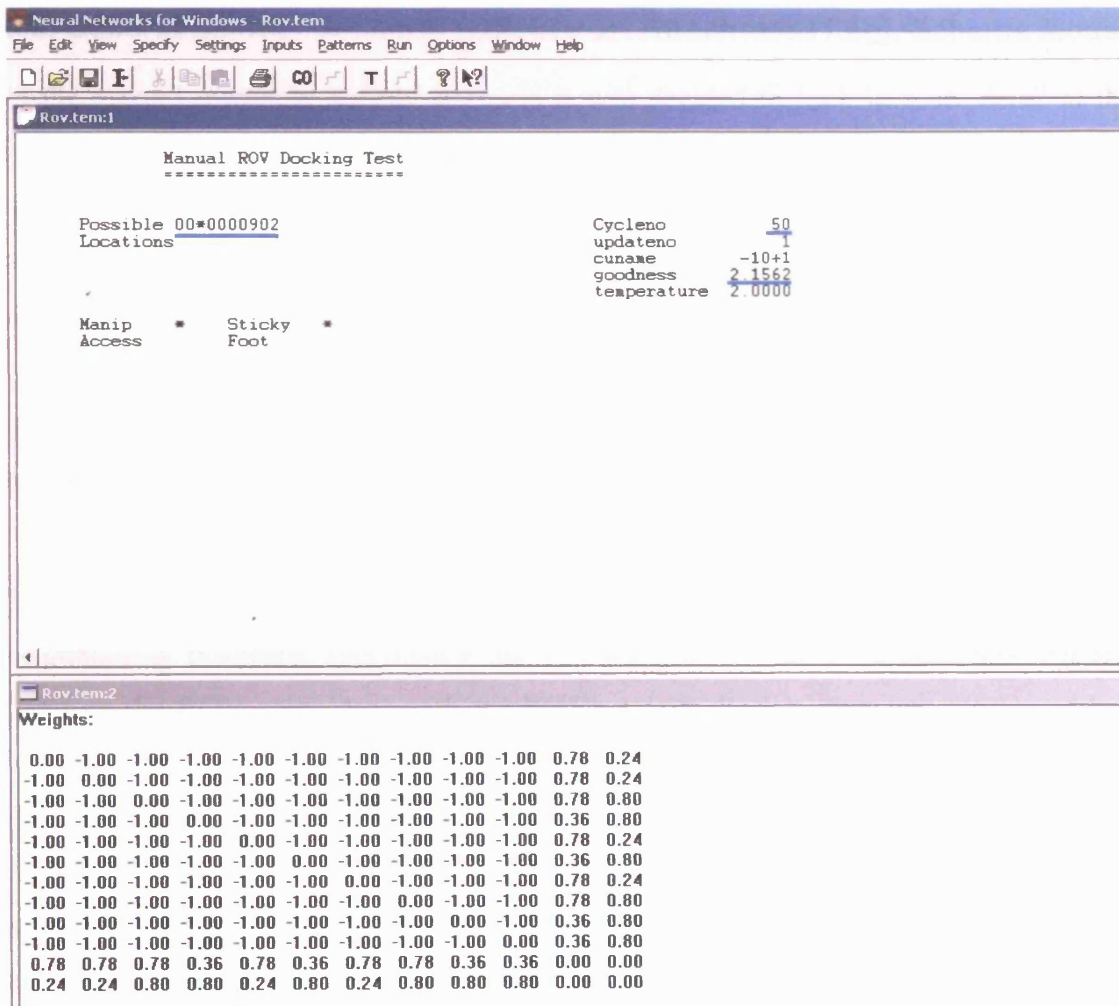


Figure 6.4 – Results of ROV Docking Test after 50 cycles

The NNW software was clearly giving the correct result, since positions 3 and 8 had both the highest manipulator reach value and the highest attachment leg value. It was decided, therefore, to increase the complexity of the scenario.

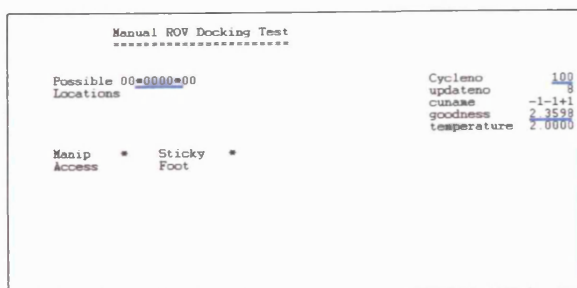


Figure 6.5 – Results after 100 cycles

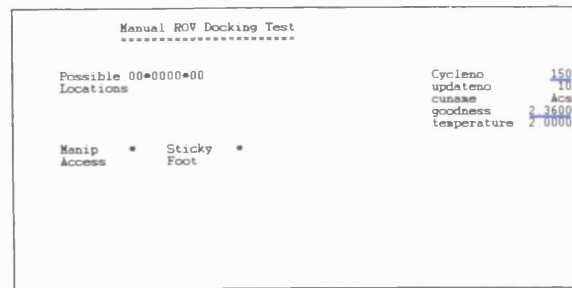


Figure 6.6 – Results after 150 cycles

### **6.3. Second Manual Scenario (Offset Port Attachment Leg), Schema Model**

In the second manually defined scenario it was decided to look in more detail at the requirements for attaching the sticky feet and so produce a more sophisticated scenario that better reflected real life complexities. Specifically, it was clear that the first scenario was too simple in that it was possible to look at a particular position and determine easily whether it was a good position by calculating values for its manipulator reach, attachment leg extension etc. Since there was no parallel computation required (choosing one position had no effect on the choice of neighbouring positions) it was felt that the neural network was not able to show any advantage over conventional linear processing systems.

Therefore a scenario was defined in which there was a connection implied between neighbouring positions, specifically that any attachment legs were not connected at a point on the ROV coincident with the ROV's origin. Rather, that any attachment legs were offset horizontally or vertically from the ROV origin. This meant that when considering a particular candidate position, it might have a particular value for manipulator reach for an ROV at that position (as the manipulator origin was still assumed to be coincident with the ROV origin). However, in addition it might have, for example, an attachment leg value that applied to an ROV at a position on its left (for a starboard attachment leg) or a value that applied to an ROV at a position on its right (for a port attachment leg) and so on.

This may be better explained with an example (and for simplicity the attachment legs are assumed to be offset by 1m in any direction, i.e. to match the coarseness of the grid being considered): the position  $(-1, 0, 0)$  may have a certain manipulator reach value for an ROV at that position but in addition it may have an extension leg value for a starboard attachment leg from an ROV docked at  $(-1, 1, 0)$  – a position 1m to the left, and a further extension leg value for a port attachment leg from an ROV docked at  $(-1, -1, 0)$  – a position 1m to the right, and so on. Although in this example the attachment leg offsets are artificially constrained to 1m, the method could be extended (i.e. if automated) so that any offset could be fairly well considered given a sufficiently fine grid.

In order to make manual definition of this scenario feasible the following method was used:

1. Assume the same simple node, access point and 1m grid, discard any points inside the chord or brace, and score the manipulator access the same as before.
2. Assume that the ROV has one attachment leg, *and that it is offset 1m to the port side of the ROV*. See Figure 6.7. Using this method, unlike the previous one, any number of attachment legs could ultimately be considered.
3. Assume that the manipulator is at the ROV origin *and that the ROV is facing the access point with its axis normal to the chord and brace axes*, i.e. that for any ROV position the port attachment leg is at  $y+1$  for  $x < 0$  and at  $y-1$  for  $x > 0$  (there are no valid positions with  $x=0$ ). Then score manipulator reach as before.
4. Score the attachment leg extension as follows: 0 if outside the attachment leg range, 0.3 if the attachment leg can attach *regardless of extension*.
5. Put the attachment leg value at its position in the grid not at the position of the ROV, which is 1m to the right.

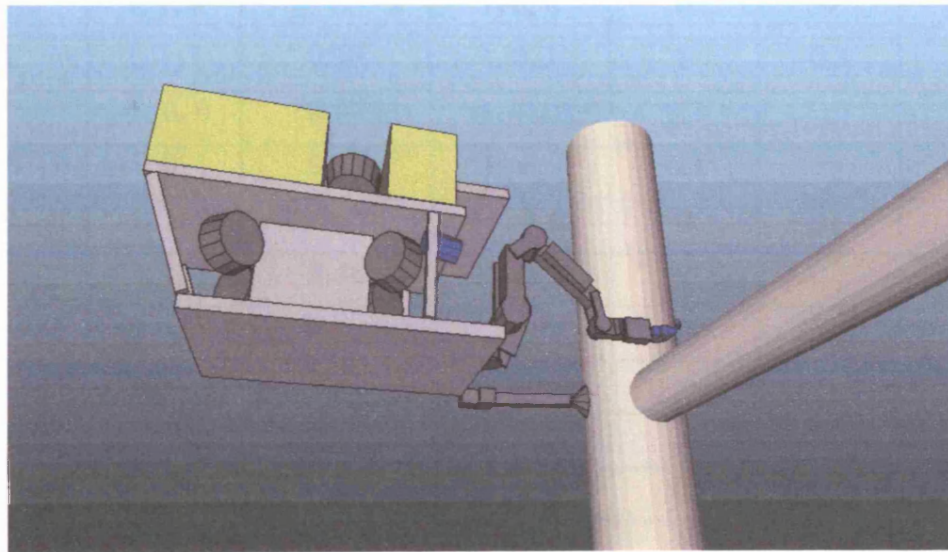


Figure 6.7 – Illustration of the Second Scenario (offset port attachment leg)

This change in the scoring of the attachment leg extension value was introduced since it was now possible to take account of multiple attachment legs. The value 0.3 was chosen so that each leg that could attach added a significant amount to the value (1 leg = 0.3, 2 legs = 0.6, 3 legs = 0.9) up to 3 legs, then, since the value could not exceed 1.0, the fourth or further legs would add little to the value. This was to reflect the physical situation where more legs produced a stronger attachment until a tripod was made with three legs, beyond which a fourth or further legs give very little further improvement in

attachment rigidity. Furthermore, it was felt that this better represented the criteria involved in choosing a docking position, i.e. it was more important that the ROV could attach (and have a significant manipulator reach in that position) than whether or not there was significant residual movement in the attachment legs.

The method was then applied to the candidate positions considered for the first method, excluding only those points already known to be inside the chord or brace. It included points excluded previously since in this scenario it is only necessary this time either to have an acceptable manipulator reach value or for the neighbouring position on the left to have an acceptable extension leg value, not necessarily both at the same location.

Candidate Number	ROV Position (X, Y, Z)	Manipulator Reach Value	Port Attachment Leg Position	Port Leg can Attach to Chord?	Port Leg can Attach to Brace?	Valid Position Number
5	1, 0, 0	0.67 (x=1.0)	1, -1, 0	0.3		1
6	1, 0, 1	0.78	1, -1, 1	0.3		2
7	1, 1, 0		1, 0, 0	0	0	
8	1, 1, 1	0.36	1, 0, 1	0	0.3	3
9	-1, 0, 0	0.67	-1, -1, 0	0.3		4
10	-1, 0, 1	0.78	-1, -1, 1	0.3		5
11	-1, 1, 0		-1, 0, 0	0	0	
12	-1, 1, 1	0.36	-1, 0, 1	0	0.3	6
14	0, -1, 1		✖			
15	1, -1, 0	0.78	✖			
16	1, -1, 1	0.36	✖			
19	1, 0, -1	0.78	1, -1, -1	0.3		7
20	1, 1, -1	0.36	1, 0, -1	0	0.3	8
21	1, -1, -1	0.36	✖			
22	0, -1, -1		✖			
23	-1, 0, -1	0.78	-1, -1, -1	0.3		9
24	-1, 1, -1	0.36	-1, 0, -1	0	0.3	10
25	-1, -1, 0	0.78	✖			
26	-1, -1, 1	0.36	✖			
27	-1, -1, -1	0.36	✖			

Table 6.4 – Scoring docking positions manually, second scenario

From this manual scoring of docking positions there remained out of the 20 locations considered, 10 candidate positions that satisfied the different criteria (marked with yellow shading in Table 6.4). The next stage was to create a neural network definition that included these 10 positions in two distinct ways, firstly as a set of possible ROV docking positions and secondly as a second independent set of possible attachment leg positions – so now a 20 unit network was required.

A network definition was created with appropriate files as follows:

- `Rov2.str` (network strengths), `Rov2.tem` (screen layout) and `Rov2.l00` (variables layout): Essentially the same as for the first Scenario but with names for 20 units – these were for the candidate positions in each of the two sets (10 for ROV positions, prefixed 'RV', and 10 for attachment leg positions, prefixed 'AL'). See Appendix D.
- `Rov2.net` (network definition): This defined a symmetrical network with 20 units, 20 inputs and 20 biases, and with 20 updates per cycle. See Appendix D.
- `Rov2.wts` (unit weights): This defined the weights applied to each unit in the form of a 20 x 20 array. See Table 6.5.

As before it is worth looking at the network weights as defined in the `Rov2.wts` file in more detail (see Table 6.5). As before, for each of the possible candidate positions there is a negative (-1.0) value from each of the other candidates so that they are mutually exclusive, except that this now applies not just to the ROV positions but to the leg positions as well, i.e. the system will tend to choose one and only one ROV position and one and only one leg position. This time instead of having a manipulator reach value for each position and a fixed starting bias, the manipulator value is used as the starting bias for the ROV positions (highlighted in blue), and a fixed bias is used for the leg positions (highlighted in yellow), i.e. the ROV position will be chosen partly depending on its manipulator value, whereas the leg positions all start on an equal basis.

0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	+00
-1.0	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	0.	0.3	0.	0.	0.	0.	0.	0.	0.	+0+
-1.0	-1.0	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	0.3	0.	0.	0.	0.	0.	0.	0.	0.	+++
-1.0	-1.0	-1.0	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	-00
-1.0	-1.0	-1.0	-1.0	0.	-1.0	-1.0	-1.0	-1.0	-1.0	0.	0.	0.	0.	0.	0.3	0.	0.	0.	0.	-0+
-1.0	-1.0	-1.0	-1.0	-1.0	0.	-1.0	-1.0	-1.0	-1.0	0.	0.	0.	0.	0.3	0.	0.	0.	0.	0.	-++
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	-1.0	-1.0	-1.0	0.	0.	0.	0.	0.	0.	0.3	0.	0.	0.	+0-
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	-1.0	-1.0	0.	0.	0.	0.	0.	0.	0.3	0.	0.	0.	++-
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	-1.0	0.	0.	0.	0.	0.	0.	0.	0.	0.3	0.	-0-
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.3	0.	-+-
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	+00
0.	0.	0.3	0.	0.	0.	0.	0.	0.	0.	-1.0	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	+0+
0.	0.3	0.	0.	0.	0.	0.	0.	0.	0.	-1.0	-1.0	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	+++
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	-1.0	-1.0	-1.0	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-00
0.	0.	0.	0.	0.	0.3	0.	0.	0.	0.	-1.0	-1.0	-1.0	-1.0	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-0+
0.	0.	0.	0.	0.3	0.	0.	0.	0.	0.	-1.0	-1.0	-1.0	-1.0	-1.0	0.	-1.0	-1.0	-1.0	-1.0	-++
0.	0.	0.	0.	0.	0.	0.	0.3	0.	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	-1.0	-1.0	-1.0	+0-
0.	0.	0.	0.	0.	0.	0.3	0.	0.	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	-1.0	-1.0	++-
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.3	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	-1.0	-0-
0.	0.	0.	0.	0.	0.	0.	0.	0.3	0.	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.	-+-
0.67	0.78	0.36	0.67	0.78	0.36	0.78	0.36	0.78	0.36	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	
+00	+0+	+++	-00	-0+	-++	+0-	++-	-0-	-+-	+00	+0+	+++	-00	-0+	-++	+0-	++-	-0-	-+-	

Port Foot. Key: + for +1, - for -1

Table 6.5 – Network weights in Rov2.wts file

### *Chapter 6: Docking Planning Using Neural Networks*

Running this network 50 times gave the results shown in Table 6.6 (where the First Position and Second Position columns use the Valid Position Numbers from Table 6.4). Each run consisted of 500 cycles, though in the majority of cases the results were settled on within about 50 cycles.

Run	First Position	Second Position	Goodness
1	2	3	1.18
2	2	3	1.18
3	9	10	1.18
4	2	3	1.18
5	2	3	1.18
6	1	5	0.77
7	5	6	1.18
8	5	6	1.18
9	2	3	1.18
10	2	3	1.18
11	5	6	1.18
12	2	3	1.18
13	7	8	1.18
14	2	3	1.18
15	2	3	1.18
16	2	3	1.18
17	9	10	1.18
18	7	8	1.18
19	2	3	1.18
20	5	6	1.18
21	2	3	1.18
22	7	8	1.18
23	2	3	1.18
24	2	3	1.18
25	2	3	1.18
26	2	3	1.18
27	2	3	1.18
28	2	3	1.18
29	7	8	1.18



Run	First Position	Second Position	Goodness
30	5	6	1.18
31	2	3	1.18
32	2	3	1.18
33	5	6	1.18
34	9	10	1.18
35	2	3	1.18
36	5	6	1.18
37	2	3	1.18
38	7	8	1.18
39	5	6	1.18
40	2	3	1.18
41	2	3	1.18
42	2	3	1.18
43	2	3	1.18
44	2	3	1.18
45	2	3	1.18
46	1	7	0.77
47	2	3	1.18
48	2	3	1.18
49	5	6	1.18
50	2	3	1.18

Table 6.6 – Results from 50 runs of the second scenario network definition

Figure 6.8 shows the system after 30 cycles as it starts to settle on a First Position (ROV Location) of 9 and a Second Position (Sticky Foot Location) of 10, when it has a goodness of 0.8206.



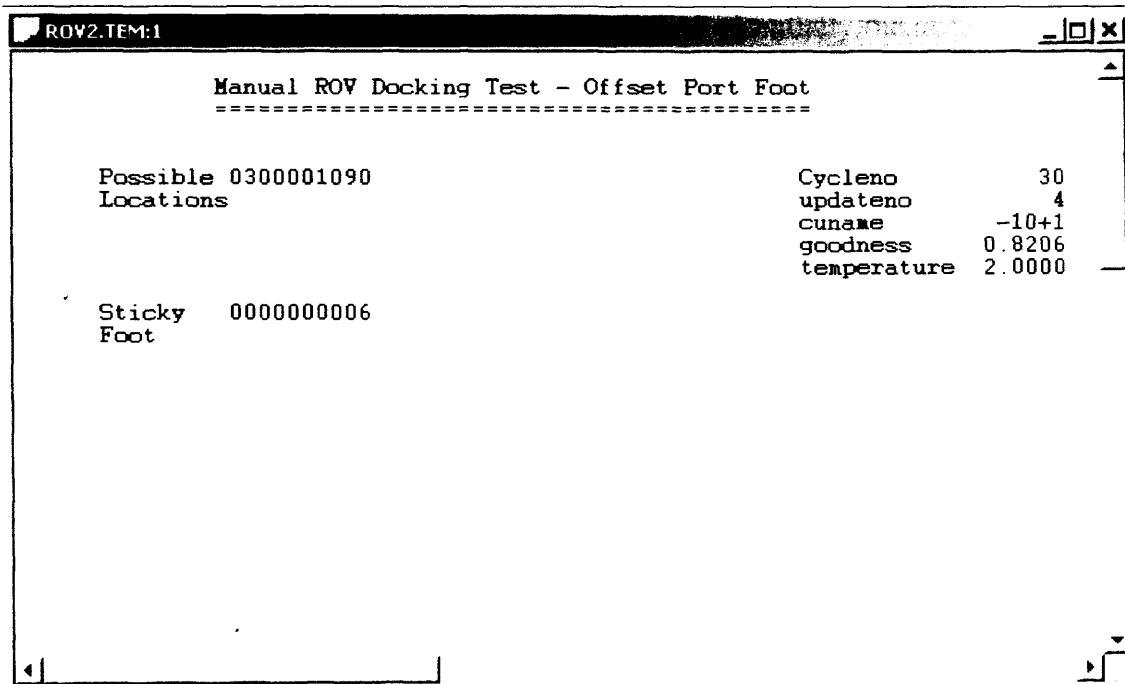


Figure 6.8 – Results of second Manual ROV Docking Test after 30 cycles

It is clear from these results that positions nearly all come in pairs (position 2 with position 3, 5 with 6, etc.) and, in fact, it turns out that the system is in each case correctly choosing a valid ROV position along with a valid attachment leg position 1m to the left. Note, however, that *nothing should be read into the ordering of the positions*. The system definition was completely symmetrical and so it is effectively choosing the best pairs of points - so in fact it is as likely to produce the pairs 3 and 2, 6 and 5, and it chooses them the way it does simply because the first element in each pair has the higher bias.

First Position	Second Position	Goodness	No. Of Occurrences
1 = 1, 0, 0	5 = -1, 0, 1	0.77	1
1 = 1, 0, 0	7 = 1, 0, -1	0.77	2
1 = 1, 0, 0	9 = -1, 0, -1	0.77	1
2 = 1, 0, 1 (Port Leg)	3 = 1, 1, 1 (ROV)	1.18	59
5 = -1, 0, 1 (ROV)	6 = -1, 1, 1 (Port Leg)	1.18	17
7 = 1, 0, -1 (Port Leg)	8 = 1, 1, -1 (ROV)	1.18	11
9 = -1, 0, -1 (ROV)	10 = -1, 1, -1 (Port Leg)	1.18	9
TOTAL			100

Table 6.7 – Summary of results from 100 runs using the Schema Model

A second set of 50 runs of the system was made and essentially the same results were found, although this time a new combination of positions occurred once (1 and 9). The combined results from the 100 runs are summarised in Table 6.7 (the appropriate position label, 'ROV' or 'Port Leg', has been applied retrospectively simply to make interpretation easier). It is clear that in the vast majority of cases the network successfully chooses appropriate neighbouring pairs of positions, one for the ROV and one for the port leg. However, it is interesting to note that on three occasions it chose completely inappropriate pairs of positions (1 and 5, 1 and 7, 1 and 9 – the two from the first 50 runs are highlighted in Table 6.6). These represent outcomes where the system has settled into local maxima rather than the global maximum as indicated by the goodness value. These selections are indicated as inappropriate solutions by the low goodness value; however, they remain very stable (to check this, one was run to 50,000 cycles and did not change, with the goodness constant at 0.77).

In order to examine this process in more detail, a selection of 10 runs was repeated, with the goodness value after each 10 cycles being recorded (and also after 1, 2 and 5 cycles as it was found that the goodness value changed very quickly right at the start). The goodness values are plotted against the number of cycles in Figure 6.9. The plots are labelled to indicate the number pair that they seemed to be starting towards, and then the number pair they actually finished on (e.g. "4,8 then 7,8")

The plots clearly fall into three distinct categories. Firstly, the majority of runs increased fairly smoothly in goodness and reached the maximum value after about 50 cycles. Secondly, a smaller number increased in goodness gradually as though about to converge on a low goodness value and then suddenly start increasing again towards the final maximum value. Thirdly, a very small number (just one of which has been plotted) increase in goodness gradually and do in fact converge on a low goodness value. The first group are, of course, those that successfully choose a valid position pair and settle on the global maximum goodness value. The third group are those that choose an invalid pair and settle on a local maximum goodness value. The second group are perhaps the most interesting as they choose an invalid pair temporarily, but then successfully 'change their minds' and choose a valid pair, at which point their goodness value starts to shoot up again.

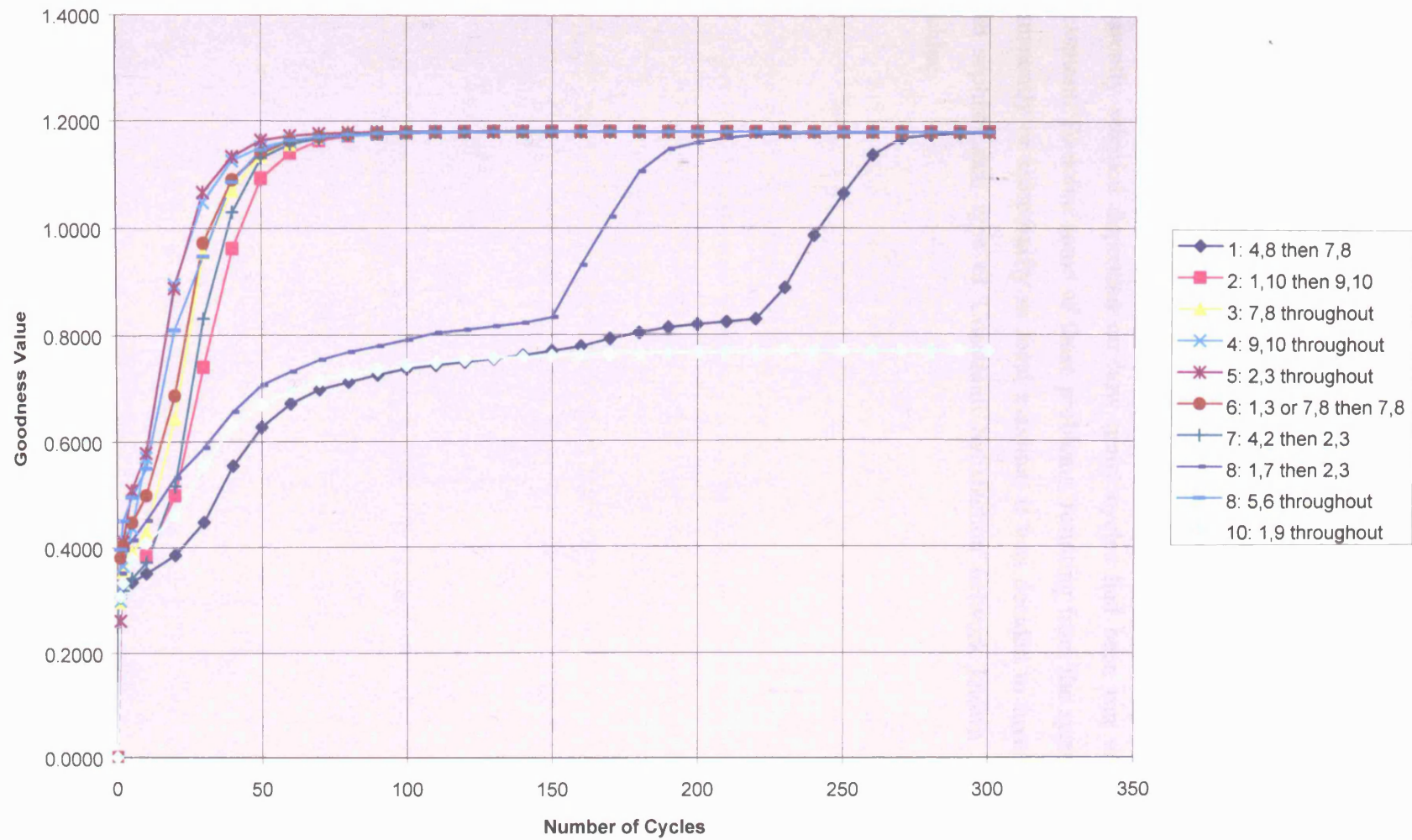


Figure 6.9 – A Plot of Goodness Against Number of Cycles for the Second Manual Scenario, Schema Model

One point of interest that is not immediately obvious is that the first group actually includes some (the outlying plots, 2 and 7) that appeared initially to be settling on invalid pairs just like the other groups but 'changed their minds' early enough that they are virtually indistinguishable from those that settled on a correct pair from the start.

It was clear that using the Schema Model the network had successfully chosen valid pairs in the majority of cases; however, it occasionally got completely stuck in a local maxima and chose an invalid pair and, nearly as problematic, it sometimes chose a valid pair only after having settled on a different pair for some time – and which pair was apparently selected depended on how many cycles had been run when making the assessment. To solve some of these problems, resulting from the system getting stuck permanently or temporarily in local maxima, it was decided to investigate using the more sophisticated type of Constraint Satisfaction network known as a Boltzmann Machine.

#### 6.4. Second Manual Scenario (Offset Port Attachment Leg), Boltzmann Machine

The previous scenario was rerun using the Boltzmann Machine version of constraint satisfaction network (see *Section 4.4.1. Boltzmann Machine Theory*) because of its ability to get out of local minima (as demonstrated in *Section 4.4.2. Necker Cube (Boltzmann Machine)*). A new network definition was therefore created, with appropriate files as follows:

- `Rov2.str` (network strengths): See Table 6.8. The main changes were the incorporation of a flag to indicate that the Boltzmann Machine model was to be used (highlighted in blue), plus the definition of an annealing schedule that started with a temperature of 2.0 and decreased down to 0.05 at cycle 250 (highlighted in yellow).
- `Rov2.tem` (screen layout), `Rov2.loo` (variables layout) and `Rov2.net` (network definition): unchanged.
- `Rov2.wts` (unit weights): Because in the Boltzmann machine the units are binary (the system only settles if the weights are integers) so all the weight values were multiplied by a factor of 10, then rounded to the nearest integer. See Appendix D.

```
set dlevel 1
get network rov2.net
get weights rov2.wts
set mode clamp 0
set mode boltz 1
set param estr .4
set param istr .4
set ncyc 250
get annealing 2 250 .05 end
get unames RV+100 RV+10+1 RV+1+1+1 RV-100 RV-10+1 RV-1+1+1 RV+10-1
RV+1+1-1 RV-10-1 RV-1+1-1 AL+100 AL+10+1 AL+1+1+1 AL-100 AL-10+1
AL-1+1+1 AL+10-1 AL+1+1-1 AL-10-1 AL-1+1-1
```

Table 6.8 – `Rov.str` file with Boltzmann mode on and annealing schedule defined

Running this network 100 times gave the results shown in Table 6.9. Each run consisted of exactly 250 cycles, as this was the end of the annealing schedule and the selection was essentially fixed at this point. It was also estimated that 250 cycles would be sufficient since all changes had largely finished before this in the Schema Model runs.

First Position	Second Position	Goodness	No. Of Occurrences
1 = 1, 0, 0	4 = -1, 0, 0	3.60	3
2 = 1, 0, 1	2 = 1, 0, 1	3.60	1
2 = 1, 0, 1	7 = 1, 0, -1	3.60	1
2 = 1, 0, 1	9 = -1, 0, -1	3.60	1
7 = 1, 0, -1	5 = -1, 0, 1	3.60	1
9 = -1, 0, -1	2 = 1, 0, 1	3.60	1
9 = -1, 0, -1	4 = -1, 0, 0	3.60	1
2 = 1, 0, 1 (Port Leg)	3 = 1, 1, 1 (ROV)	4.80	29
5 = -1, 0, 1 (ROV)	6 = -1, 1, 1 (Port Leg)	4.80	22
7 = 1, 0, -1 (Port Leg)	8 = 1, 1, -1 (ROV)	4.80	27
9 = -1, 0, -1 (ROV)	10 = -1, 1, -1 (Port Leg)	4.80	13
TOTAL			100

Table 6.9 – Summary of results from 100 runs using the Schema Model

In fact it was found that there were now more invalid pairs being selected than before. Five runs were repeated and examined in more detail, considering the goodness value against cycle number as before, and plotted in Figure 6.10. The first 3 plots are valid pairs and achieve the maximum global goodness value of 4.8 but the last 2 plots are invalid pairs and only achieve a goodness value of 3.6. Unlike with the Schema Model runs, the particular positions chosen change virtually every cycle so there is no concept of the system settling towards a particular pair then perhaps changing and settling towards a different pair. This is because the selections are binary so a particular position is chosen completely then in the next cycle it is rejected and a different one chosen, and it is essentially the probability of the system staying with a particular selection that is changing in the Boltzmann Machine. The plots are therefore simply labelled with the positions they finally selected.

Clearly since there are many invalid pairs being selected, annealing over 250 cycles had not been sufficient to prevent the system settling into local maxima. The process was therefore repeated with an annealing schedule extended to 500 cycles ("get annealing 2 500 .05"), and the runs extended to 500 cycles. The results are summarised in Table 6.10.



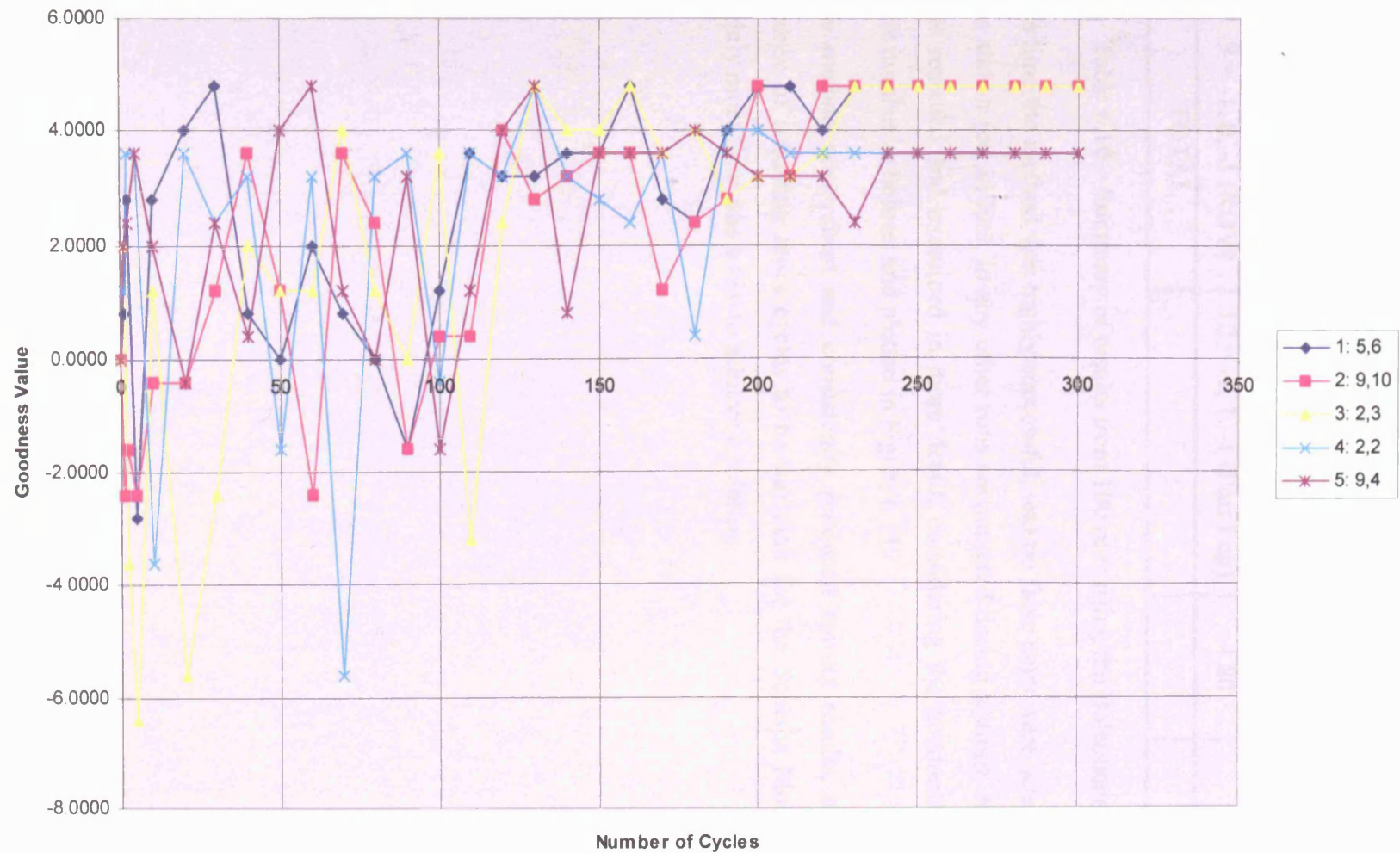


Figure 6.10 – A Plot of Goodness Against Number of Cycles for the Second Manual Scenario, Boltzmann Model, Annealing over 250 cycles

First Position	Second Position	Goodness	No. Of Occurrences
2 = 1, 0, 1 (Port Leg)	3 = 1, 1, 1 (ROV)	4.80	13
5 = -1, 0, 1 (ROV)	6 = -1, 1, 1 (Port Leg)	4.80	41
7 = 1, 0, -1 (Port Leg)	8 = 1, 1, -1 (ROV)	4.80	18
9 = -1, 0, -1 (ROV)	10 = -1, 1, -1 (Port Leg)	4.80	28
TOTAL			100

Table 6.10 – Summary of results from 100 runs using the Boltzmann Machine

This time the method was highly successful, and no false pairs were selected in the 100 runs shown (or, in fact, in any other runs encountered during testing). Again, five runs were repeated and examined in more detail, considering the goodness value against cycle number as before, and plotted in Figure 6.11.

This method was robust and consistently produced correct results, although at the expense of requiring more cycles to be run than for the Schema Model, and hence slightly more time was taken to achieve a solution.



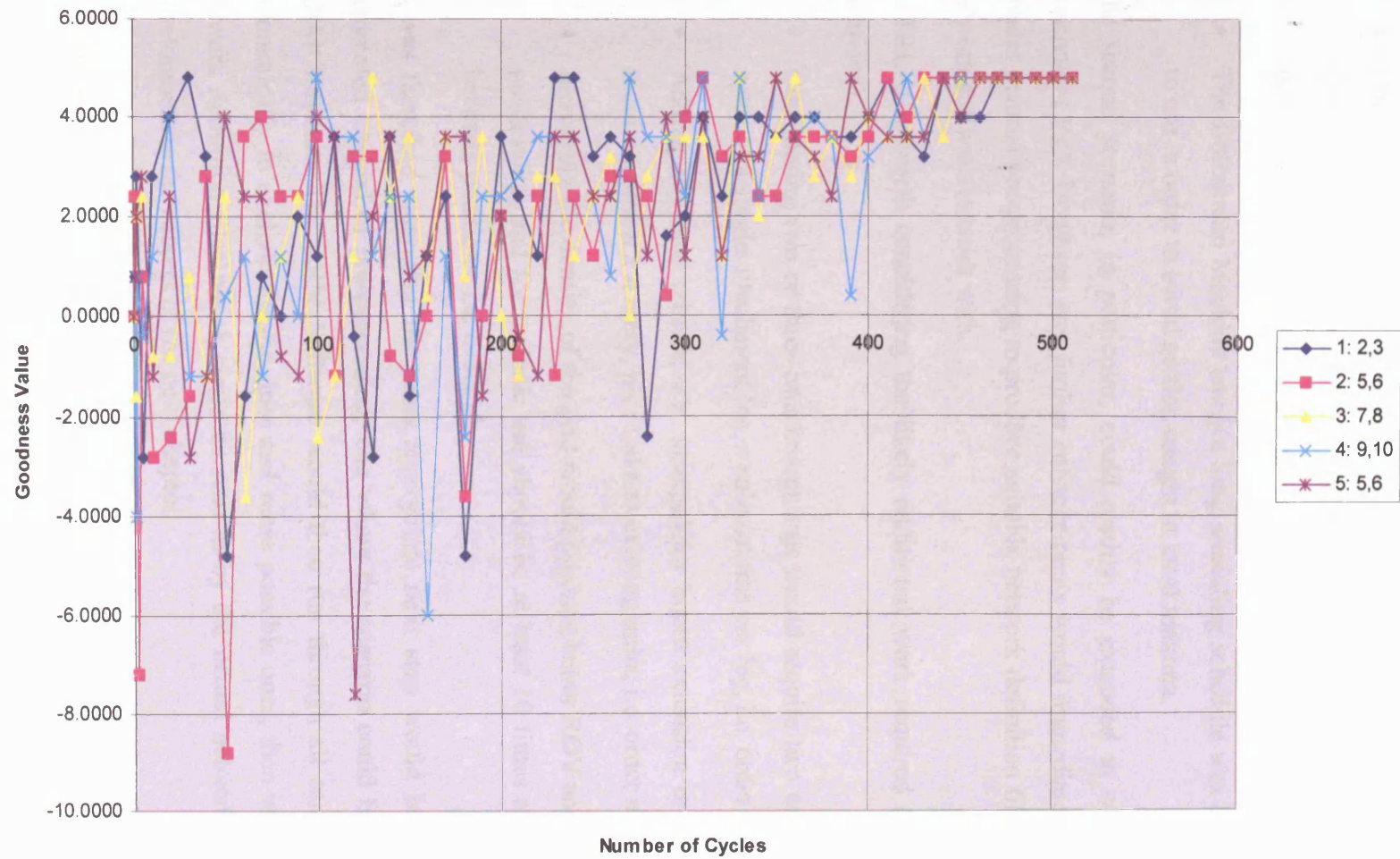


Figure 6.11 – A Plot of Goodness Against Number of Cycles for the Second Manual Scenario, Boltzmann Model, Annealing over 500 cycles

## 6.5. Conclusion

The main results from this section of the work were:

- Both of the manually defined scenarios tested had successfully demonstrated that the neural network system was capable of choosing appropriate docking positions given a set of possible candidates and appropriate selection criteria.
- The Boltzmann Machine using a long annealing schedule was the best network to use in order to avoid getting caught in local maxima.

The second scenario, in particular, could readily be extended to consider multiple attachment legs. However, any further enhancements would immediately require much greater manual pre-processing to produce suitable network definition files for the neural network software to work with.

In fact, it is worth considering the likely additional work required (for  $n$  candidate positions):

- Considering two or three attachment legs would require two or three times the work of a single attachment leg,  $n$  calculations per leg, i.e. order  $n$  per leg.
- Adding any new criteria, e.g. manipulator boom extension or rotation, would require a new  $n \times n$  array, plus links to existing units, i.e. order  $n^2$ .
- Increasing the density of the grid would produce better ROV and attachment leg positioning, and for realistic use should be at least 10 times as fine, say 0.1m between positions, i.e. order  $n^3$ .

It was therefore clear that the most appropriate next step would be to develop an automated method of pre-processing, one where the scenario could be defined in the ARM Software. The ARM Software would then run through all candidate positions automatically to exclude invalid ones and score possible ones, then write these out to network definition files which could be read in by the neural network software. This development is the subject of the next chapter.

---

## CHAPTER 7:

# DOCKING PLANNING USING NUMERICAL PROCESSING

---

### 7.1. Introduction

The next stage was to add new code to the ARM Software to pre-process a system definition (ROV, toolskid, manipulator and workpiece) to produce the network definition files for the NNW software. The aim initially was to conduct the equivalent of the manual processing described in the previous chapter, then to extend the processing to deal with more complex scenarios. The procedure would have the following stages:

1. Create a large number of possible candidate ROV locations.
2. For each candidate, check if the position is in collision (e.g. inside chord or braces) and eliminate it if it is.
3. Calculate a manipulator reach value for each position.
4. Calculate attachment leg values for each position.
5. Create and write to the appropriate files a network definition encompassing this information, i.e. \*.str, \*.tem, \*.loo, \*.wts, \*.net files, ready for the neural network software to read in and solve.

The new code in ARM consisted of a new "Docking" library (`docking.lib`) which, unlike the majority of ARM code, was written in C++ rather than C. This consisted of two new classes, `CDocking` and `CCandidate`, plus a set of C global functions that formed an interface between the C calling in ARM, and the C++ `CDocking` class. The docking files are listed at Appendix E.

## 7.2. First Automated Scenario (Coincident Attachment Legs)

The definition of the first manual scenario was used as a 'recipe' for the work that needed to be done in order to be able to generate suitable neural network files automatically. The nine specifications (summarised from *Section 6.2. First Manual Scenario (Coincident Attachment Legs), Schema Model*) were as follows:

1. Assume a simple node, a vertical chord (radius 1m) with a single horizontal intersecting brace (radius 0.5m) arranged so that the brace direction is perpendicular to the ROV axis.
2. Assume required access point is at the centre of the brace where it intersects the chord (to represent approximately typical access to the whole weld around the brace where it intersects the chord).
3. Consider possible ROV docking positions as being on a coarse grid of points 1m apart, centred on the required access point, extending from -1 to 1 in x, y and z.
4. Discard any points inside the chord or brace.
5. Assume that the manipulator is located at the ROV origin (and that it is not attached to a rotating or extending boom, or similar system).
6. Discard any points that are too close or too far from the required access point for practical manipulator interaction, i.e. if not within working manipulator reach.
7. 'Score' remaining points in terms of how much working reach the manipulator retained, the criterion being how close it was to the middle of its reach range (considered to be 0.5m to 2.0m for the Slingsby ARM inspection manipulator under consideration).
8. Assume that the ROV has one attachment leg, and that it is connected to the ROV at its origin. Discard any points that are too close or too far from the chord or brace surfaces for the attachment leg to attach.
9. 'Score' remaining points in terms of how well the attachment leg was fixed, the criteria being how close it is to the middle of its extension range (considered to be 1.3m to 1.6m for the Slingsby ARM attachment legs under consideration).

This was considered to be a list of requirements that led to the following tasks being conducted (one task to satisfy each requirement):

1. A simple node with a vertical chord (radius 1m) with a single horizontal intersecting brace (radius 0.5m) was created in ARM as a workpiece file.
2. The existing ARM software was temporarily modified to use the centre of the brace where it intersects the chord as the target point for the manipulator access check.
3. The new docking code generated a large number of candidate positions on a pre-defined grid; this will be described in more detail below.
4. The candidate positions were checked by existing ARM code for collisions with the workpiece and rejected if they were inside the chord or brace.
5. In ARM, the ROV origin was set to be the base of the manipulator, and for the initial work it was assumed there was no extend/rotate boom (though this was included in later work, as will be detailed below).
6. The new code calculated the manipulator reach value for each candidate position and rejected any position outside the given range.
7. Remaining positions had their reach value calculated automatically using the same function as described for manual operation.
8. The new code calculated the distance required for the attachment leg to reach either the chord or brace surface and rejected any positions outside the given range.
9. The new code calculated the attachment leg value for each remaining candidate position.

On command from the user the Docking library creates a list of candidate docking positions extending from  $-1\text{m}$  to  $+1\text{m}$  in all three directions, with a grid spacing of  $1.0\text{m}$ , centred on the target point (the chord surface at the intersection of the brace) and then draws them onto the ARM display; the 27 positions are shown in Figure 7.1. It then removes any positions in collision with the workpiece (i.e. each point is checked to see if it is contact with or inside the chord or brace); the remaining 22 positions are shown in Figure 7.2.

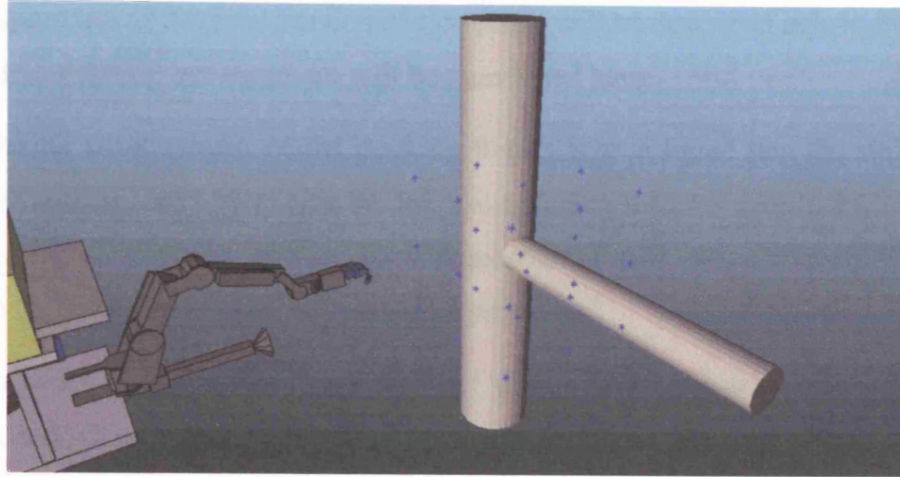


Figure 7.1 – Creation of candidate positions (-1 to +1 in X, Y and Z, 1m grid spacing)

The architecture of the Docking library was very straightforward. The `CCandidate` class represented a possible candidate position; it contained a standard ARM `CVector` class that held the position as X, Y and Z parameters, plus it contained other parameters such as the reach values and a validity flag, plus various functions for accessing its parameters (see Appendix E).

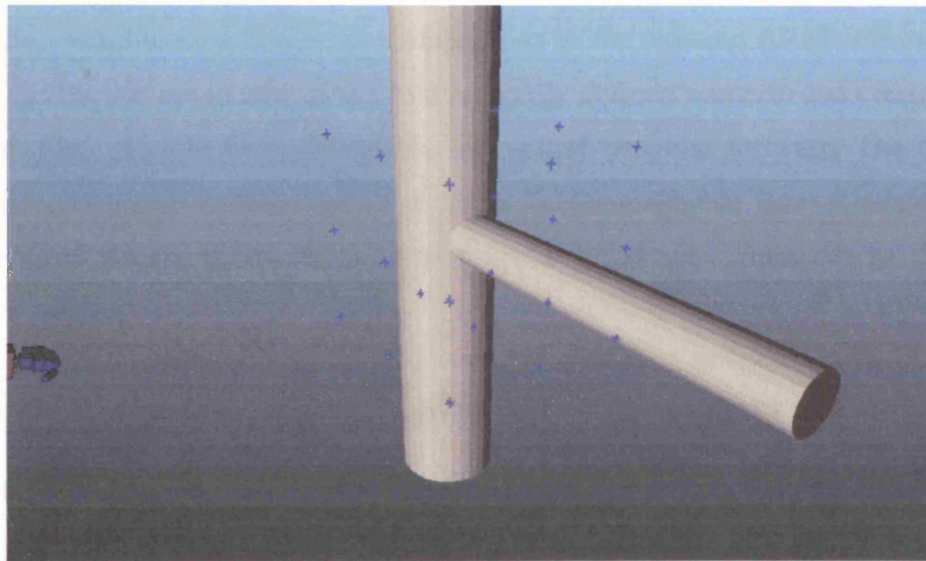


Figure 7.2 – Elimination of candidate positions in collision with workpiece

The `CDocking` class did most of the work – creating a list of `CCandidate` positions as required, cycling through them and setting their reach values and other parameters if they remained valid positions or else setting their validity flag false, then on completion of the calculation cycles pruning away all invalid positions (see Appendix E). The

detailed operation of the final version of `CDocking` will be described below, but some of the milestones during development will be mentioned here:

- During development of the docking library it was found that the software was not able to correctly handle the large amounts of memory required for very large lists of candidate positions. For this reason the code was converted from 16 bit to 32 bit operation and this cured the problem.
- In order to evaluate the operation of the process a visualization function was added whereby the candidate positions were drawn onto the ARM display; then, as the process developed, it was possible for the user to observe as positions were pruned down to the remaining valid candidates. Furthermore, a facility was added to step through the process to see at what point particular positions were rejected.
- During initial testing it was found that there were candidate positions remaining that had been rejected in the manual processing. It was found that it was incorrectly allowing the attachment leg to stick on a brace even where the brace was actually inaccessible because the chord was in the way – this was corrected.

Eventually, with the new library and adaptations to the existing ARM software, it was possible for the system to take in the first manually defined scenario and create network definition files suitable for reading into the neural network software. The `CDocking` code produced network definitions that were indistinguishable from the ones created manually and which, therefore, produced the same results when run in the neural network software.

### **7.3. Second Automated Scenario (Offset Port Attachment Leg)**

Adapting the `CDocking` code to consider the manipulator/ROV position and attachment leg position separately – with the attachment leg 1m to the left of the ROV position – appeared to be very straightforward since the values were simply calculated for, and stored in, different `CCandidate` positions. With the necessary adjustments to writing the network definition files, the system would be able to generate both manually tested scenarios automatically and so get the same results when run in the NNW software.

However, rather than just replicate the manual work, the automated system was significantly more powerful since everything was parameterised and changeable in the software. For example, one of the fundamental limitations of the manual method was



the amount of work required in defining a network with more than about ten candidate positions – hence the large 1m grid spacing. Conversely, the automated system worked just as easily with a finer grid, say 0.5m spacing (8 times more points) or even 10cm spacing (1000 times more points). Similarly, the system worked just as easily with a larger grid, say 3m x 3m x 3m or even 4m x 4m x 4m. Figure 7.3 shows the system working with a grid 3m in each direction and a spacing of 0.25m (a total of  $13^3 = 2197$  positions).

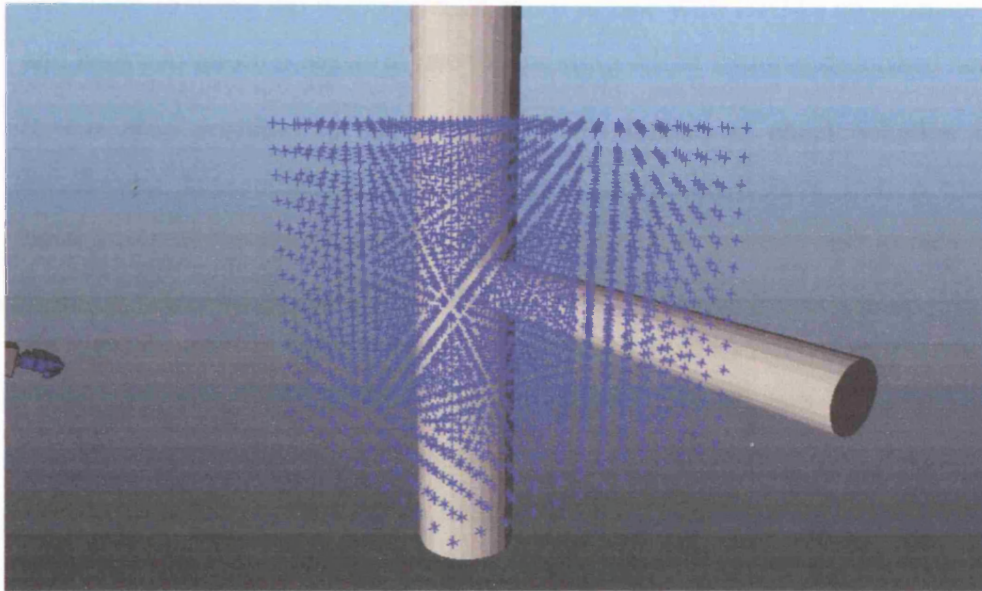


Figure 7.3 – Increased grid density from automated method

Unfortunately, this brought its own problems. With, say, 10 times as many more points the automated system was still very fast at checking them all (taking less than a second). However, when defining the neural network files, it had to create network (\*.net) and weight (\*.wts) files which contained an array of any remaining candidate positions and their interconnections. Therefore 10 times as many candidates required 100 times as much disk space. Furthermore, the NNW software conducts its calculations per connection, and so it now had to do 100 times as many calculations.

It was clear, therefore, that it would be better to use more sophisticated pruning in the pre-processing phase, rather than try to eliminate the worst candidate positions in the neural network phase.



#### **7.4. Third Automated Scenario (ARM Collision Detection and Attachment Leg Features)**

It was decided to significantly increase the sophistication of the scenarios being considered, since the work was now automated, and so consider the effects of other elements. Three main changes were incorporated initially:

1. The origin of the grid of candidate positions was moved from the centre of the brace where it intersects the chord to the centre of the complete workpiece. This was done to bring the docking code more in line with ARM's co-ordinate frame, and hence to make it easier to use the existing ARM features described here.
2. It was now possible to use ARM's inbuilt feature to check whether defined attachment legs could attach in a given scenario (taking into account full kinematics of the leg, including its various links, joints and ball-jointed 'ankle'), and this replaced the simple check for sticky feet reach used up to this point<sup>1</sup>. The attachment leg functionality was also extended in two ways. Firstly, it was enhanced so that it could check for attachment onto any tubular in the node, not just the nearest tubular as it had done previously (sometimes the nearest tubular is too close). Secondly, it was modified so that it could be called programmatically from the Docking library rather than only on command from the user. This use of automated checking for sticky feet attachment meant that it was now fully feasible to consider the attachment of one, two, or three feet at little cost.
3. It was decided to consider collisions between the ROV and the workpiece. This required extending the ARM collision detection library since, in the existing ARM code, the ROV position was considered to be fixed throughout an ARM session, and so the collision detection only worked between the manipulator and other elements (e.g. ROV and workpiece). Also, the change of position grid origin described above made it much easier to interpret the results of the collision detection checks (since the ROV positions were now relative to the workpiece origin, rather than to a weld origin which varied with the selection of the weld, and with the specific chord diameter).

---

<sup>1</sup> In the Docking library code given in Appendix C, the function `CheckStickyFeetPositions` is the replacement for `CheckStickyFeetReach` in `docking.cpp`.

Within the docking code, the attachment legs are checked as follows (see `CheckStickyFeetPositions` in Appendix E). For each candidate position, and for each available attachment leg (ARM has 3, most other systems have 2), a check is conducted to see if the leg can attach to the nearest tubular, and if not then to each of the tubulars in turn. With the visualisation and reporting feature turned on (using the `Show Graphical Working` menu item) it is possible to see the system go through the processing graphically, and also in that case the system displays messages when a valid arrangement is found giving first the ROV position and then the sticky foot position on the tubular. Figure 7.4 shows the system checking one of the ROV positions (-1, -2, 0.75) and finding it can attach two sticky feet.

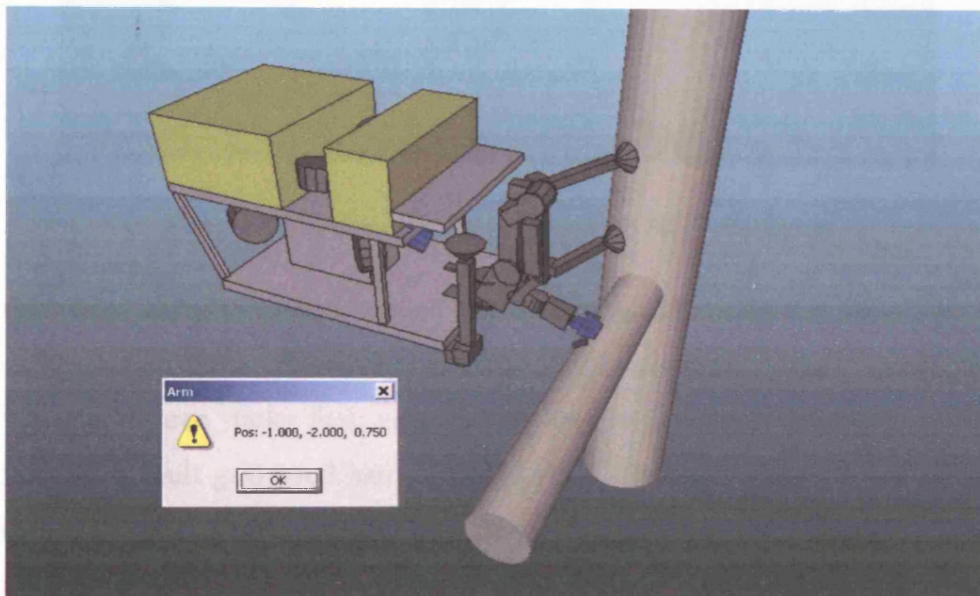


Figure 7.4 – Checking a position for sticky feet attachment

It was clear that invalid candidate positions could be rejected by appropriate use of ARM's collision detection facilities. Specifically, although all scenarios up to this point (manual and automated) rejected any ROV position that was in contact with or inside the workpiece, this did not fully take account of the bulk of the ROV, particularly its width and height. With the new changes, the docking library was able to check all of the candidate positions by moving the ROV, in turn, to each position – and then rejecting a position if any part of the full ROV model was in contact with the workpiece. For this process to be appropriate, the orientation of the ROV was crucial and so from this point on the grid of candidate positions was restricted by default to just one side of the workpiece at one time. Of course, this restriction does not preclude a second check being conducted with the ROV on the other side, if required. With `Show Graphical`

Working on, Figure 7.5 illustrates the system checking one of the positions and finding it causes a collision between the ROV frame and the vertical chord.

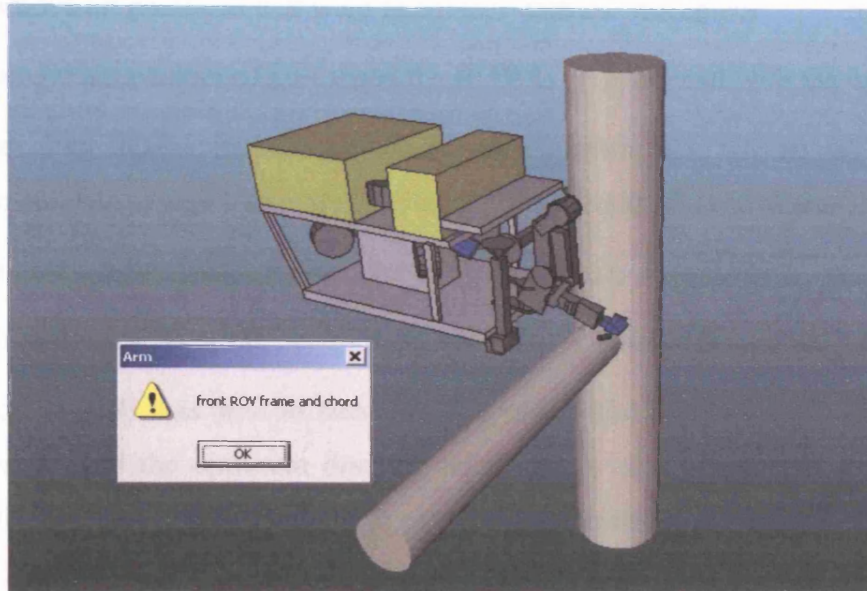


Figure 7.5 – Checking a position for ROV collision

From this point on, with these changes and the increased ability for the pre-processing to reject invalid positions, the system was capable of conducting docking checks on an ROV with up to three sticky feet, very quickly and with a much finer grid of candidate positions (the default grid used had a spacing of 0.25m). For example, considering the original workpiece and using a grid that extends from -1 to 0 in X, -2 to 1 in Y (to allow for more positions opposite the brace, because of 1. above) and Z from -1 to 1:

- Creation produces 585 positions.
- Removing positions inside the workpiece leaves 509.
- Removing positions that cause ROV collisions leaves 234.
- Removing positions where less than two sticky feet attach leads to just 17 remaining positions.

The work in this phase demonstrated that combining the ROV collision detection processing and the automatic attachment leg processing with the existing work had provided a significantly improved automated docking method.

Having completed this phase<sup>1</sup>, the docking library was able to:

1. Create a large grid of candidate docking positions
2. Remove all positions that were in contact with the workpiece
3. Remove all positions that caused the ROV to be in contact with the workpiece
4. Calculate an attachment leg value for each position (this was simply the ratio of the number of legs successfully attached to the total number of legs on the ROV)
5. Calculate a manipulator reach value (as previously described)
6. Write the results out to network definition files suitable for reading into NNW

At this point NNW was able to successfully read in the network definition files and solve them to find the optimum docking position. To aid interpretation, the position labels output to the strengths file (`.str`) gave the position co-ordinates in the ARM world co-ordinate frame (rather than the ARM workpiece co-ordinate frame used for the calculations). This allowed easy visualisation of the final position selected by NNW – by typing the label into ARM as the ROV position, it was possible to see the ROV in the chosen position immediately.

It should be noted that once the pre-processing had been completed and the remaining positions passed to NNW, there was little for NNW to do apart from select the position with the highest manipulator reach and attachment leg values. Since all attachment legs were often in use, this actually came down to simply selecting the position with the highest manipulator reach value. Where there were multiple positions with the same manipulator reach value, NNW always returned the first position. This process clearly demonstrated the effectiveness of the pre-processing, but it also brought into question the value of the NNW 'post-processing'. This point will be returned to later.

### **7.5. Fourth Automated Scenario (Weld Access Check and Deployment System)**

For the next stage it was decided to add more 'real-world' elements to make the system more useful. The five main changes were as follows:

---

<sup>1</sup> Some significant time was also spent on general bug fixing in the ARM software; in particular a problem was found, and eventually fixed, whereby the system would crash, apparently randomly, when certain workpiece files were loaded in sequence.

### *Chapter 7: Docking Planning Using Numerical Processing*

1. Use of two ARM features, one which allows the user to specify the weld segment of interest and another that conducts a simulated manipulator scan along the weld segment. These features were incorporated into the pre-processing with a number of enhancements – namely, the ability to call them programmatically (rather than on command from the user) and to conduct the manipulator scan as a simple kinematic check along the weld (rather than having to conduct a complete simulated scan). Another change was made so that the degree of access was returned as a numerical value which indicated the ratio of weld path that could be reached (so full access = 1.0) – see below. With this change it was possible for the Docking library to make an automatic check of all its candidate positions, and have returned the degree of access each one could achieve on the weld path of interest.
2. The manipulator kinematic path check made the manipulator reach value scored from the distance of the manipulator from the weld – as used up to this point – completely redundant. It was therefore removed, and from this point 'manipulator reach value' is used to mean the access score from the kinematic path check.
3. Consideration was given to the manipulator deployment system which is usually available on advanced ROV/manipulator systems, e.g. the extending and rotating boom on the ARM toolskid. It was decided to automate ARM's deployment system modelling feature which allowed the user to set fixed values for the deployment extension and rotate values. Instead, this was controlled programmatically so that the docking library could cycle through a sequence of extensions and rotations to check the access available at each arrangement (using the manipulator kinematic path check just described). This allowed the system to consider arrangements where, for example, the ROV system was able to inspect a long weld path by keeping the ROV on one side and using the deployment system to reach the other side.
4. The ARM collision detection system was extended to consider the deployment system (e.g. ARM's extending boom), i.e. each boom extension/rotation

arrangement was checked to ensure it did not bring the boom into contact with the workpiece<sup>1</sup>.

The kinematic path check operates as follows. First of all the user selects a section of weld, which may be either as clock positions, distances along the weld, or angles around the brace. Next the ARM software generates a fixed number of straight line segments (typically 1cm long) around the weld that approximates it very closely. Finally, all the positions at the intersections between these segments are checked to see if the manipulator can access them – i.e. an inverse kinematic check is conducted to see if a mathematical solution can be found to position the manipulator tool with the required position and orientation. This check necessarily considers the manipulator base position and so it takes account of the deployment boom extension and rotation. The access value then returned simply represents the ratio of points that could be accessed, for example, if 30 of 50 points could be accessed then the returned value will be 0.6. An acceptance limit can be set in the software which indicated the minimum acceptable value – for example, with an acceptance limit of 0.5 any positions with a lower access value were pruned away.

Figure 7.6 illustrates the new scenario considering the full ARM toolskid with its correctly separated attachment legs and its extending/rotating boom for the manipulator. The weld segment illustrated is the left-hand half as viewed from along the brace, i.e. from 6 to 12 o'clock in clock positions; the system has conducted a manipulator kinematic path check (under user command) and found that the manipulator can access 0.82 (displayed to the user as 82%) of the path from its current position and boom configuration.

---

<sup>1</sup> Note that the manipulator itself was not checked for collision as the boom was extended and rotated since its particular physical configuration could not be known without conducting a time consuming simulation of it accessing all parts of the weld. Once one or more optimum docking positions had been determined then, of course, a more detailed check of the manipulator configuration during access to the weld could be conducted.



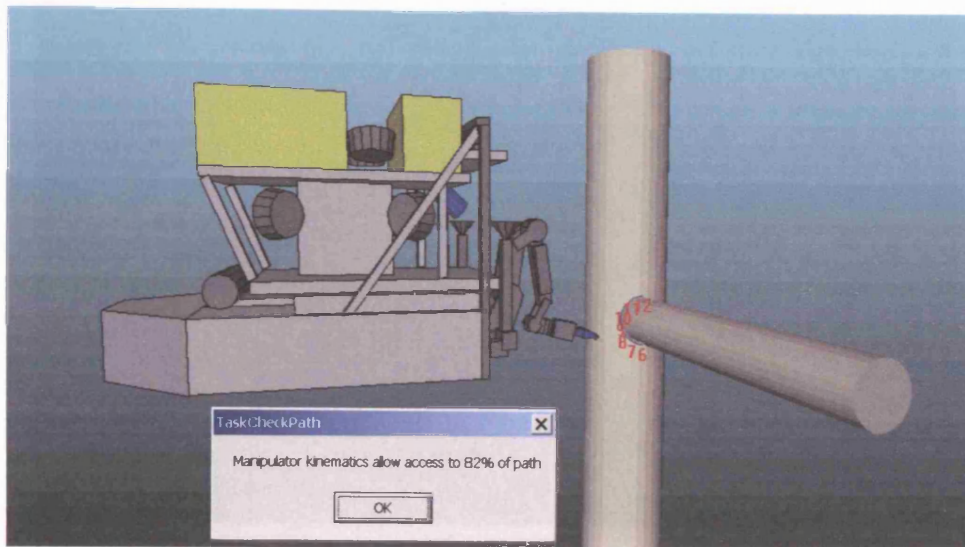


Figure 7.6 – Checking a position for manipulator kinematic access

Consideration of the deployment system, typically an extending and rotating boom, operates as follows (see *CheckForReach* in Appendix E). For each candidate position the boom is extended in increments (typically 0.25m); for ARM, this is from 0 to 2.0m. For each extension the boom is rotated in increments (typically 5°); for ARM this is from -180° to 180°. At each rotation a manipulator kinematic check is conducted. If the kinematic check succeeds then a collision check with the boom configuration is conducted. If this succeeds, the kinematic access value is compared with the current best value for that ROV position and replaces it if it is greater. The boom is then rotated again and the operation repeats; once the boom has reached its maximum rotation it resets to its minimum value, the boom is extended another increment and the operation repeats. Once the boom reaches its maximum value the ROV is moved to the next position with the boom reverting to its start extension and rotation. Once all positions have been checked, the best access value achieved for each one is checked against the acceptance limit threshold and if it is too low it is pruned away.

Figure 7.7 illustrates the use of the manipulator deployment boom (here extended to 1m and rotated by 180°) to allow the manipulator to access the far side of the brace (the weld path from 3 o'clock to 6 o'clock has been selected here). If this is done during the automated docking, and *Show Graphical Working* is on, the system displays a message as each arrangement is checked – either giving the ROV position, boom extend and rotate values, and access value achieved or, if it failed due to a collision, giving the objects in collision (generally the extended boom hitting some part of the workpiece).

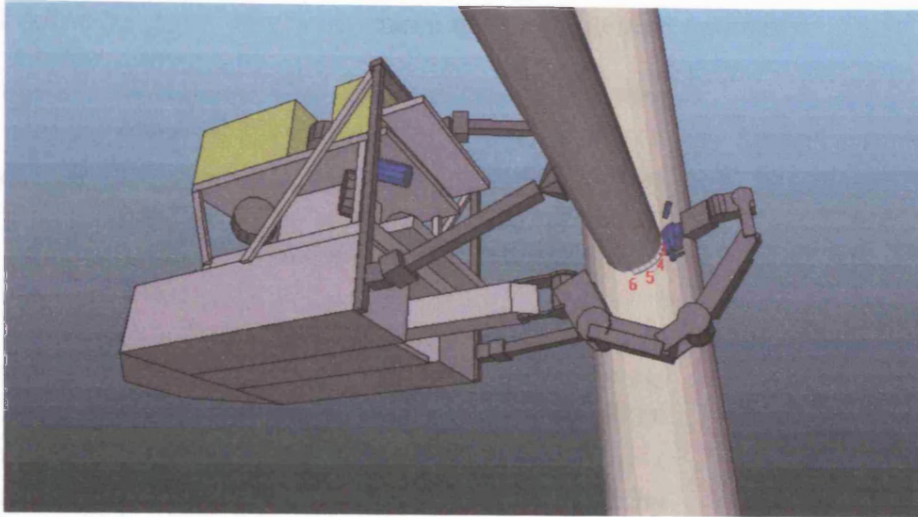


Figure 7.7 – Accessing the far side of a brace using the manipulator deployment system

It was at about this time that a number of significant changes were made to NNW, including the conversion from 16-bit to 32-bit processing and the rebuild as "NNW32" (as discussed in *Section 3.6. Neural Networks for Windows (NNW)*). The original intention had been to use the new deployment system extension and rotation values, and the modified manipulator reach value, as criteria to save to the network definition files, and thus to use them as inputs to the neural network processing. This was not done, however, as it was found that the results from the ARM processing, now that they considered most of the important criteria, were sufficiently detailed to allow a human operator to make a good selection of a docking position from the shortlist produced. This point will be returned to below.

## 7.6. Final Development

By this stage the ARM processing method had developed to the point that it was clearly able to determine a shortlist of good docking positions for the ROV system. However, the software needed to be edited in the code slightly each time for different scenarios and so a number of changes were made to make it much easier to use. The most obvious ones were to provide a means for the user to directly edit the various parameters of the processing, to run some or all stages independently, and then to directly view the results. A simple `Docking Position` menu had been added to the ARM Software during this work and so it was extended to provide the features required.



## Chapter 7: Docking Planning Using Numerical Processing

The final ARM Docking Position menu has the following options:

Settings...	Brings up the Docking Settings dialog box (see below)
Initialize Grid	Creates the list of candidate ROV positions for a grid of the size and granularity defined in the Settings dialog box
Check if Inside Workpiece	Conducts collision detection to see if any of the candidate positions are inside the workpiece components, and prunes them if necessary
Sticky Feet Reach	Obsolete in final version - replaced by Sticky Feet Positions as discussed in <i>Section 7.4. Third Automated Scenario (ARM Collision Detection and Attachment Leg Features)</i>
Check for Collisions	Conducts collision detection to see if the ROV at each position will clash with the workpiece, and prunes them if necessary
Sticky Feet Positions	Checks to see if enough attachment legs can attach at each position (typically 2 if 3 available) and, if so, stores the ratio of attached legs/number of legs available - if not, they are pruned away
Kinematic Access Check	Calculates the manipulator kinematic access value, i.e. what ratio of the required weld length can be accessed, and prunes away any below the specified acceptance threshold
Init. & Do All Checks	Conducts all steps above from Initialize Grid to Kinematic Access Check in sequence, one after the other
Write Neural Net Files	Writes out the current state of the list of candidate positions into the various neural net definition files (.str, .tem, .net, .wts, .loo) - usually selected after all the checks have been completed, e.g. through Init. & Do All Checks
Show Graphical Working	This is a toggle (ticked if on) that indicates whether the system should redraw the graphics during each check
Show Results...	Brings up the Docking Results dialog box (see below)
Shut Down	Cancels the current docking scenario and deletes the list of candidate positions

Settings...	
Initialize Grid	
Check if Inside Workpiece	
Sticky Feet Reach	
Check for Collisions	
Sticky Feet Positions	
Kinematic Access Check	
Init. & Do All Checks	Ctrl+D
Write Neural Net Files	
✓ Show Graphical Working	
Show Results...	Ctrl+U
Shut down	

## Chapter 7: Docking Planning Using Numerical Processing

Once **Show Graphical Working** is on the system redraws the ROV at each candidate position in turn, and conducts the requested check. For example, if it is on and then **Sticky Feet Positions** is selected the system will draw the ROV at each remaining candidate position with the sticky feet in the best attached arrangement determined at each position. It should be noted, however, that the system runs considerably faster with it switched off and so it should be used only when appropriate. For example, with the scenario described in the previous section and with a grid of 2m x 2m x 2m (405 elements) the optimisation check takes about 10 seconds with the graphics off, and about 15 minutes with them on. However, the graphical display is very valuable when used appropriately, for example:

- After a complete check has been run, it can be used to view the remaining candidate positions and their best attachment leg configurations
- It can be used to view specific checks, e.g. to see why particular positions led to an ROV collision.

The Docking Settings dialog box (Figure 7.8) allows the user to set the grid spacing, access value acceptance threshold, grid extent in X, grid extent in Y, grid extent in Z, the increment for the boom extension, and the increment for the boom rotation.

Docking Settings		X
Grid Spacing [m]:	0.25	
Acceptance Limit [%]:	50	
X Range [m]:	-1	0
Y Range [m]:	-2	1
Z Range [m]:	-1	1
Extend Step [m]:	0.25	
Rotate Step [deg]:	5	
OK		Cancel

Figure 7.8 – Docking Settings dialog box

The Docking Results dialog box (Figure 7.9) shows the user the current results of the list of candidate positions, specifically for each position its X, Y, Z location, boom

extension value (in metres), boom rotation angle (in degrees), then its manipulator access value and its leg attachment ratio. The dialog box comes up automatically at the end of a complete docking optimisation check (initiated via the *Init. & Do All Checks* command) if there are fewer than 20 remaining valid positions. It can also be called up directly by the user at any time.

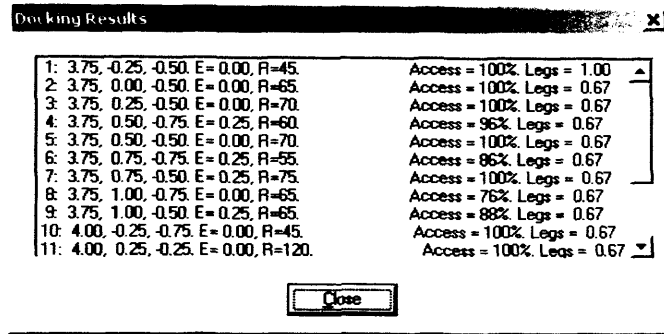


Figure 7.9 – Docking Results dialog box

## 7.7. Time Analysis

After the Final Development work described in the previous section, the last modification made to the docking optimisation code was the addition of a timing feature. This calculated the time taken for each of the phases of the optimisation, in order to allow an assessment to be made of the proportion of time spent on each of the different phases.

The results of running the final version of the software, modelling the full ARM System on the standard workpiece, gave the results in Table 7.1. All times are in milliseconds, and any tests that ran for more than one million milliseconds (about 17 minutes) were terminated.

Grid Spacing [m]	0.75	0.5	0.4	0.3	0.25	0.2	0.17	0.15	0.12	0.1
Number of Positions	108	405	726	1372	2601	4851	6912	10206	19652	32000
Initialisation of Grid	1	1	10	10	10	20	30	40	70	140
Check if Inside workpiece	10	10	10	20	50	90	110	171	340	540
Check for ROV Collision	81	300	521	962	1803	3365	4827	7140	13680	22172
Sticky Feet Attachment	190	521	841	1512	2794	5148	7301	10825	20700	33598
Manipulator Kinematic Check	64914	238272	428736	823865						

Table 7.1 – Timing Results for different optimisation phases

Note that this feature used the standard Windows timer (which operates from the PC hardware interrupt mapped to INT8 and runs at 18.2Hz) and is not guaranteed to be accurate to more than 55ms. The shortest times recorded (with the fast phases running on few positions) can therefore only be considered to be approximately correct.

The results are plotted in Figure 7.10 - a logarithmic scale is used because of the large spread of timing results. It can be seen that the time taken to do each of the calculations increases in proportion to the number of positions, as would be expected. It can also be seen that the manipulator kinematic check takes longest of all by a large margin, and this, too, can be explained since most of the checks just check the position against certain criteria (e.g. whether inside the workpiece or causing an ROV collision). However the kinematic check considers the deployment system and therefore conducts cycling of the boom through its extension and rotation range, doing the kinematic check at each increment (the default values were used, 0.25m extension increments over 2m, and 5° rotation increments over 360°, so the system was conducting about 650 checks at each position).

Of more interest is the relative speed of each phase. The checks were implemented in the given order simply because it was convenient – they increase in coding complexity in that order. The timing results confirm, however, that this complexity also relates to the time taken to conduct the calculations, and so this is the optimum order in which to conduct the checks.

Clearly it is valuable to conduct the fastest checks first in the hope of eliminating as many positions as possible before starting on the slower checks. However, it could be useful to investigate this further – for example, although checking for ROV collisions is slightly faster than checking for sticky feet attachment, if the latter removes significantly more candidate positions then it would be advantageous to conduct it first. This could be the subject of further work.

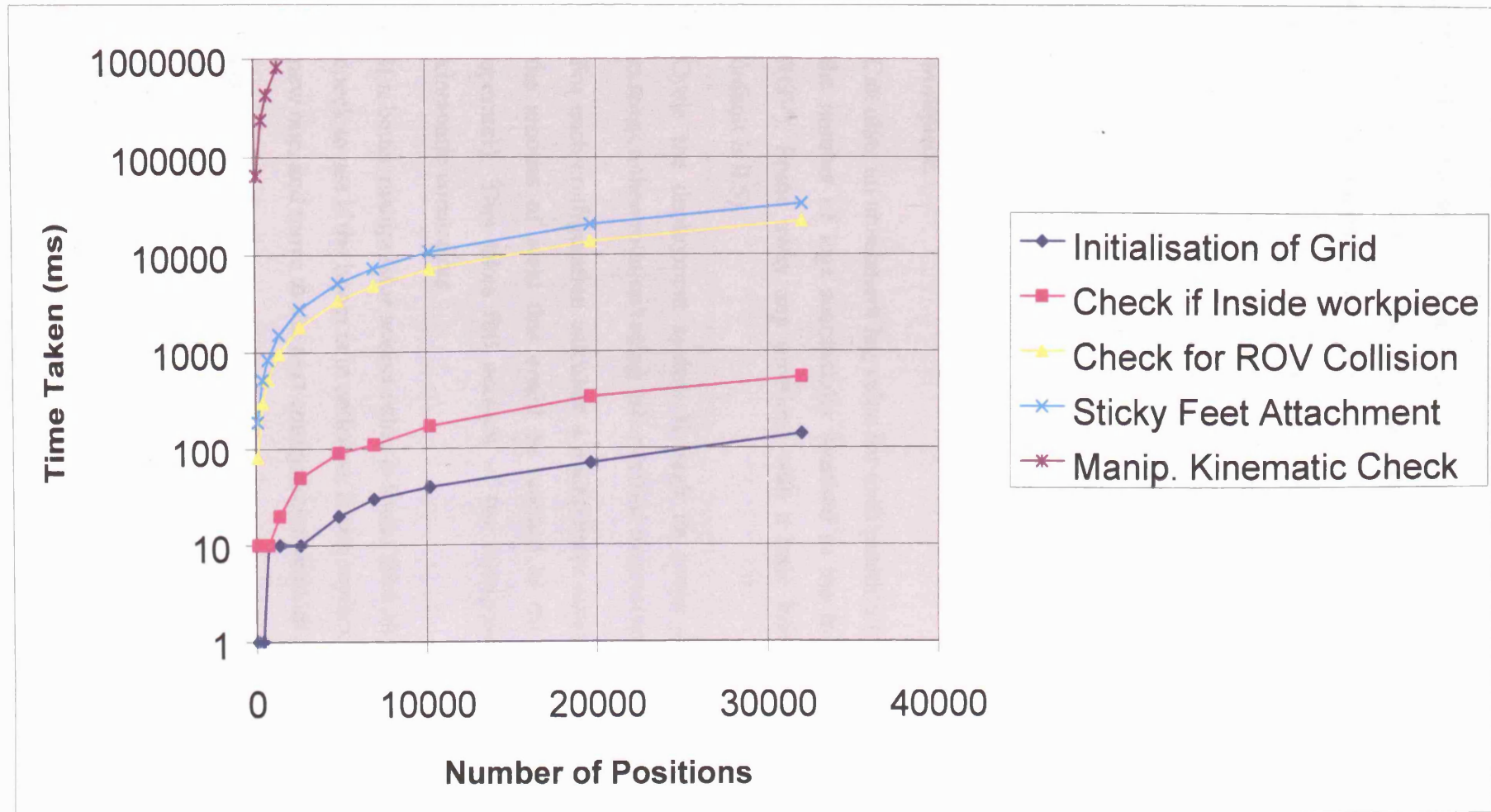


Figure 7.10 – Timing Results for Docking Optimisation Phases

## **7.8. Conclusions**

By the end of the development of the ARM processing described in this chapter the system was able to conduct the following procedure (this can be compared with the manual one given on pages 141-142):

1. Provide a number of optimisation parameters that the operator can set for a particular scenario via a dialog box, i.e. grid extent and granularity, required percentage of weld length accessed, and deployment system increments.
2. Create a grid of candidate docking positions based on the given parameters.
3. Remove all positions in contact with the workpiece.
4. Move the ROV to each position in turn and conduct the following checks.
5. Remove all positions that would cause the ROV to be in collision with the workpiece.
6. Calculate an attachment leg value for each position (this was simply the ratio of the number of legs successfully attached to the total number of legs on the ROV). Prune away any positions with a ratio below a given threshold (the default is 0.5).
7. Cycle the deployment system through its range at each position (typically extension then rotation) using the specified increments.
8. For each configuration calculate a manipulator access value (using the ratio of the amount of weld that could be reached to the amount requested by the operator). This takes full account of the manipulator's forward and inverse kinematic constraints.
9. If a better manipulator access value is found than already held for that position, check to see if the boom is in collision. If not, replace the existing value with the new one, and move to the next configuration until all are checked.
10. Having conducted all checks, prune away any positions with an access value lower than that specified, then display a dialog box listing all remaining positions and giving their extension/rotation values, access value and attachment leg ratio.

This process was very effective at providing a shortlist of valid positions for the ROV system to dock (sometimes a 'list' of just one or two positions) and, as has been described above, it was decided not to develop the neural network method for docking planning any further. From the early work on using NNW to solve simple docking optimisation problems it was clear that it could do so, but increasing complexity in the test scenarios produced a simple contradiction: if there were many candidate positions remaining after the ARM pre-processing (as there was with the earlier versions) then the size of definition files required and the amount of processing required by the neural network was extremely high (an  $n^2$  problem). However, if there were only a few positions then the information provided by the ARM processing was sufficient for a human operator to choose a position directly, without needing the neural network phase.

It is considered that this problem as finally defined was possibly not appropriate for neural network processing. Neural networks clearly have an advantage in very complex situations, where selection or activation of particular units (in this case positions) has an interrelated effect on other units. For example, there was an element of this in the second scenario, where the ROV and attachment leg positions represented different units in the same grid and so selections were linked, i.e. the selection of one position was affected by selection of adjacent positions. However, in the general case considered here where the ROV positions and leg positions may not be interconnected in any way it appears to be more efficient to consider them independently and the advantage of a neural network system is lost.

At this point a system had been developed, based on numerical processing in the ARM software, that was very effective at determining the optimum position for docking an ROV system in order to maximise access to a weld, and no further development on a neural network-based system was conducted. The remainder of this thesis will look at a number of real-life operational scenarios where the system saw extensive use.

---

## **CHAPTER 8:**

### **FIRST USES OF THE AUTOMATED DOCKING PLANNER**

---

#### **8.1. Introduction**

Although the Slingsby ARM System was never to be used operationally, as has been seen the ARM Software had found use as a simulation tool for ROV access checking and docking planning. This chapter will describe the use of the ARM Software with the newly developed automated docking planning system on two commercial jobs, the first for Woodside considering an Australian platform and the second for Elf considering a North Sea platform.

#### **8.2. Docking Planning for Woodside**

The ARM Software had been proven as a control system for the computer control of manipulators in inspection, welding and grinding trials, and even though it was not to see use in the North Sea, there was great interest in it from Australia and eventually, as will be seen below, it saw successful use conducting ACFM inspection for Woodside Offshore Petroleum Pty Ltd based in Perth.

Woodside is the operator of the North West Shelf Gas Project – Australia's largest resource development – off the north-west coast of Australia (see Figure 8.1). The first structure installed in the North West Shelf was the North Rankin Alpha (NRA), a tubular steel structure, which went in during 1982. The operational requirements for this platform included long term planning for marine growth removal and detailed NDT of selected welds [Batten 1988]. Equipment was developed first for marine growth removal since, as little as two years after installation, marine growth was building up fast and also because weld inspection necessitated the removal of marine growth first.

Two systems were developed in the 1980s for marine growth removal, a special purpose 'C'-shaped vehicle called Scimitar and an ROV tooling package designated Modular Offshore Cleaner 1 (MOC-1). Scimitar was designed for the bulk removal of marine growth along main members and was deployed into position by ROV. MOC-1 was a



custom designed toolskid mounting twin attachment arms, a manipulator carrying high pressure water jetting equipment, and an inspection camera – see Figure 8.2.

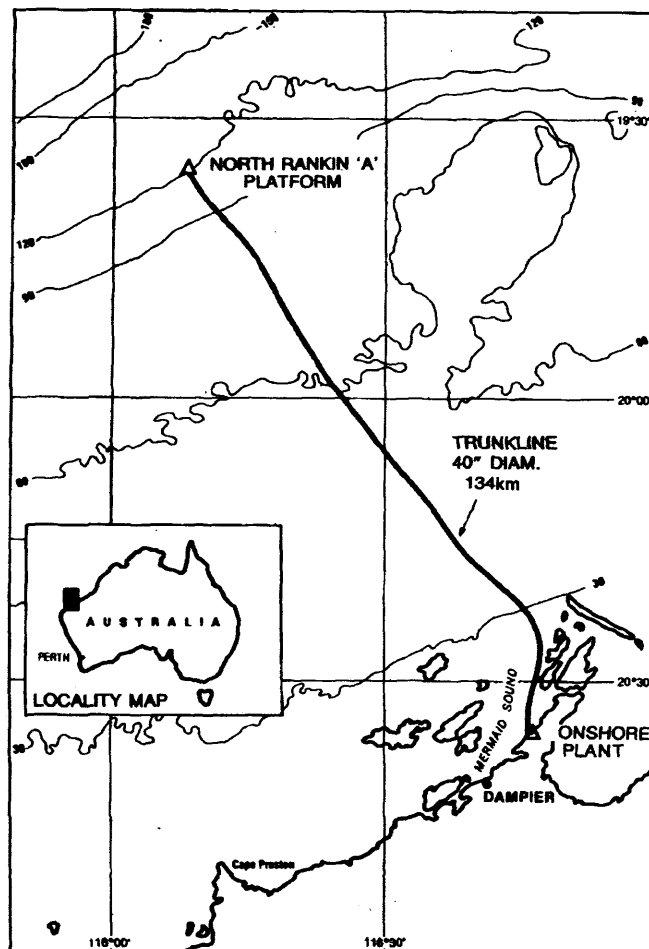


Figure 8.1 – Location of North Rankin Alpha platform [from Batten 1988]

MOC-1 was later replaced by a Nodal Inspection System (NIS) skid which had a large hydraulic claw underneath, for attaching onto horizontal braces, in place of the attachment legs; this saw extensive use for cleaning and visual inspection on NRA in the early 1990s. However, Woodside required a more effective means of weld inspection using a recognised NDT system such as ACFM and so a requirement was issued for the development of a new toolskid aimed at manipulator-deployed ACFM inspection.

The system was designed by Covus Corporation (known as Tritech International at the start of the work) and incorporated two Schilling manipulators on rotating mounts. It also had a hinging mechanism in the skid to allow the front half, with the attachment claw, to tilt and so attach to vertically diagonal braces – see Figure 8.3 (the tilt function is not being used in the illustration).

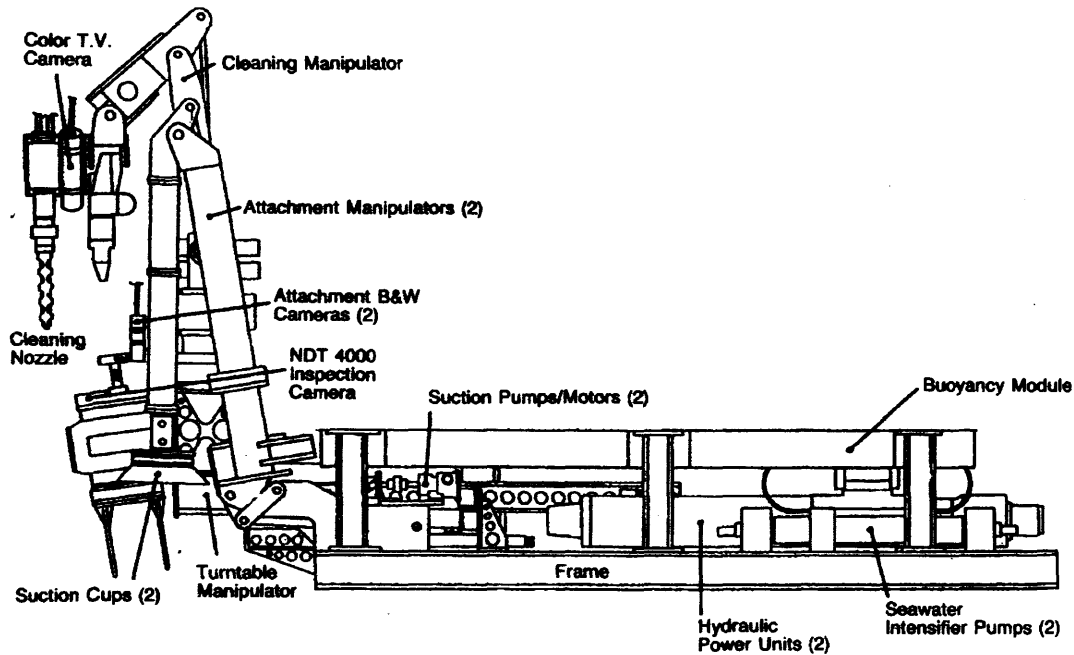


Figure 8.2 – Woodside MOC-1 inspection and cleaning toolkid [from Batten 1988]

The design appears to have originally also been designated NIS (or possibly NIS-2). The toolkid that was eventually built, however, and which is believed to have incorporated the rear frame and some of the other components from the earlier system, became generally known as NICS (for Nodal Inspection and Cleaning System).

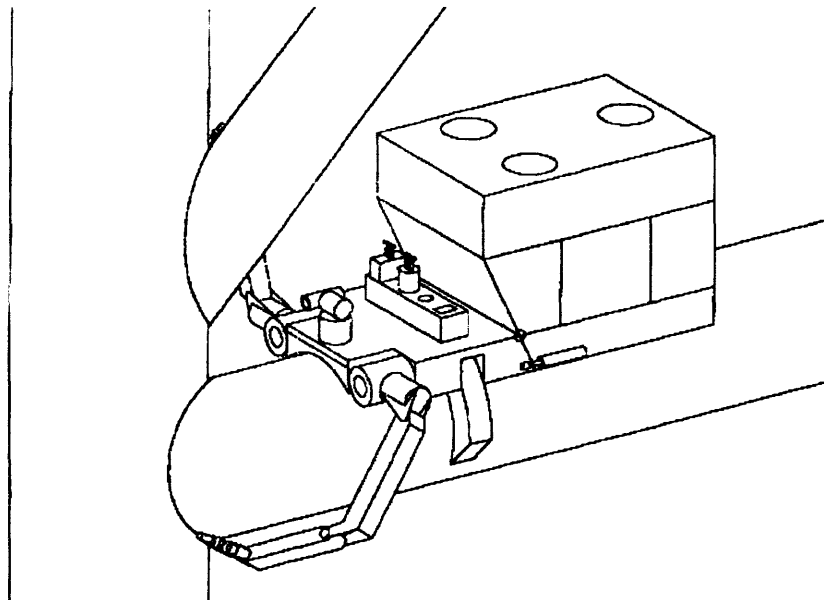


Figure 8.3 – Original NICS design (note toolkid hinge/tilt function)

In order for the project to proceed it was necessary, firstly, to determine if the proposed system was capable of inspecting the required nodes, and, secondly, to source a computer control system able to operate the manipulators for inspecting the complex

weld geometries. The ARM Software appeared to be the solution to both, and so at the outset a contract was issued to use the ARM Software to determine access to twenty-five nodal welds on NRA.

The first phase involved access checks on four selected welds and although it did not produce very conclusive access results, it did lead to the following conclusions:

- It indicated that there was little benefit in having the complex toolskid tilt function for the nodes being considered, and this was dropped from the design.
- The nodes were generally very difficult to access, and a number of different methods were investigated to improve this. The main ones incorporated the use of offsets (extension pieces) fitted between the toolskid and manipulator shoulders, and also offsets between each manipulator and the inspection probe.

The second phase involved access checks on the complete scope of twenty-five welds, including redoing the first four following new information being received on the ROV configuration and anode placement on the nodes. The first phase was conducted entirely manually (largely by the author).

The second phase was initially conducted manually (by the author and a colleague) but it soon proved unmanageable due to its sheer size, and the work slipped behind schedule. Since the ARM Processing work conducted for this thesis was approaching the point at which it could be used practically, extra effort was made to bring it to a useable point as soon as possible – essentially by conducting the work described in *Section 7.6. Final Development*. Once this was done, the access checking continued but using the automated docking planner.

It should be noted that this first real use of the automated docking planner did not fully require all the features developed for the system, specifically:

- The NICS skid had no sticky feet, and so the complex calculation of valid attachment leg configurations was not used.
- There was no extending deployment system so this was not considered, although the twin rotating deployment systems for the manipulators were fully considered.
- The toolskid had a V-shaped cut-out along its underside so that, in combination with the clamping jaws, it was forced to sit hard down on the centreline of the

brace. This removed the need to consider candidate docking positions off the centreline, i.e. there were constraints that  $Y = 0$ , and  $Z = \text{brace radius}$ <sup>1</sup>.

The optimisation therefore came down, primarily, to considering the access for each of the possible rotations of the deployment system for each of the X positions along the brace, while eliminating any that produced collisions. Conversely, compared to the Slingsby ARM System there were many more possible permutations that had to be considered, specifically manipulator offsets of 0, 0.5, 0.75 and 1.0m; and probe offsets of 0, 0.3 and 0.5m, each potentially at multiple angles (though this option was only used occasionally, as a last resort).

The manual part of the docking planning took place from 22<sup>nd</sup> to 28<sup>th</sup> April 1999; this was followed by completion work on the automated planning system from 29<sup>th</sup> April to 5<sup>th</sup> May. The docking planning then resumed using the automated planning system and took place from 6<sup>th</sup> to 20<sup>th</sup> May.

During the manual checking phase, the following welds were checked: numbers 13, 15, 17 and 18, plus half of 9, – a total of 4.5 in 5 working days, i.e. about 0.9 checks per day. During the automated optimisation phase, the following welds were checked: the other half of weld 9, plus welds 1-8, 10-12, 14, 16, 19-25 – a total of 20.5 in 11 days, i.e. about 1.8 checks per day.

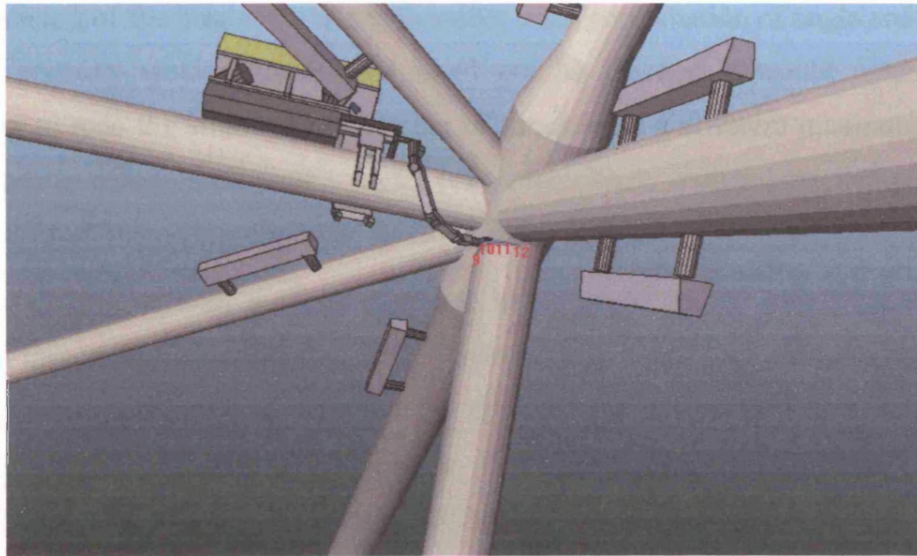


Figure 8.4 – Covus ARM System inspecting 10 to 10.30 on Woodside weld 9

<sup>1</sup> Z also has a small offset to the ROV origin that changes slightly depending on the relationship between the brace radius and the size of V-shaped cut-out, but is fixed for a particular brace.

These results imply, though not prove, a 100% increase in efficiency. Certainly, once the work was started using the automated planner the manual method was never returned to. It is interesting to look at the difference the automated planner made, for example on weld 9 which was partly done manually and partly done with the automated system. Weld 9 was a particularly challenging one because it was a diagonal extending out from the underside of a horizontal node, and hence the only access possible was from sitting on top of the node (or one of the braces joining to it) and reaching down.

The access check began on 28<sup>th</sup> April, and it was determined manually that it was possible to inspect from 7.30 to 10.30 with the right hand manipulator. This had a 30cm offset and was turned all the way over so that it was completely upside down (see Figure 8.4) and used a series of ROV positions along the central horizontal brace ( $X = 15.4, 15.2, 14.9$  then  $15.0$ , as shown, where the node centre is at  $12.2$ ).

However, finding a means of accessing beyond clock position 10.30 was very problematic. Firstly, the various  $X$  positions clearly indicated that the positioning was crucial but sensitive (i.e. it was possible to inspect parts of the weld from  $X=15.0$  that could not be reached from  $15.2$  or  $14.9$ , and that having checked in one direction, e.g. decreasing  $X$ , the next segment along may require repeating the check in the opposite direction, e.g. increasing  $X$ ). Secondly, having failed to reach beyond 10.30 the logical next thing to consider was the angle of rotation of the manipulator mount – but without any indication of the best angle to try, or rather what combination of angle and position. Thirdly, for any weld segment that could not be reached it would potentially be necessary to redo the simulation a number of times, with a different manipulator offset and/or probe offset each time. These problems clearly pointed to the use of the automated system and largely drove its final development.

Access checking with the automated system began on 6<sup>th</sup> May, using  $X$  increments of 5cm and increments in rotation,  $R$ , of  $5^\circ$ . It found that it was possible with  $X=14.7$  and  $R=160^\circ$  to inspect from 10.30 to 11.00 but no further; repeating the check with a 30cm probe offset allowed access from 11.00 to 11.30 (with  $X=14.5$  and  $R=160^\circ$ ); while a further check with a 0.75m manipulator offset but no probe offset allowed access from 11.30 to 12.00 (with  $X=14.35$  and  $R=170$ ).

Note that each of these used a completely different system configuration (different manipulator offsets and probe offsets) and so would have required a completely new manual check considering each position along the brace and multiple rotation angles for each configuration. The automated system was able to check a 3m length ( $X = 12.2$  to

15.2) in 5cm steps, considering 5° rotation increments at each step over 180°, making a total of  $61 \times 37 = 2257$  checks almost instantly (see Figure 8.5 which shows another view of the node, and the candidate positions on it – at 10cm spacing for clarity).

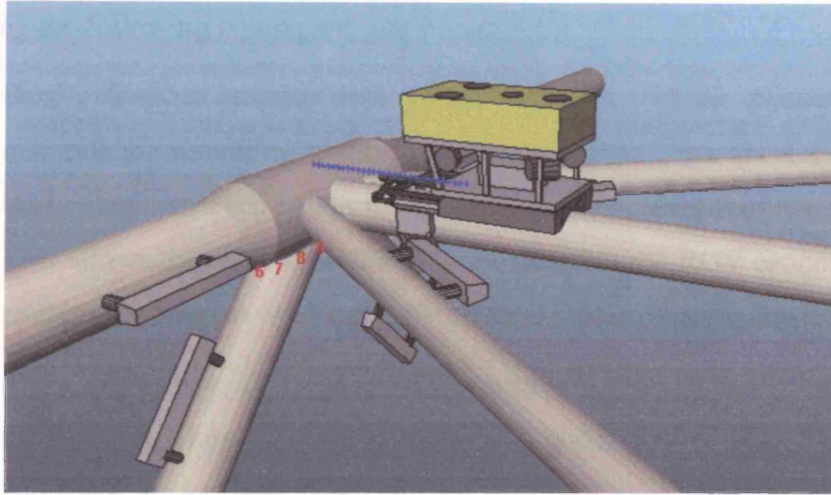


Figure 8.5 – Covus ARM System on the weld 9 node, showing automated docking positions considered

#### **8.2.1. NICS Toolskid**

Following the completion of the detailed simulation and docking planning phase, Woodside gave the go-ahead for construction of the NICS toolskid. It is described here, and its use offshore in conjunction with the automated docking planner system will be described in the next chapter.

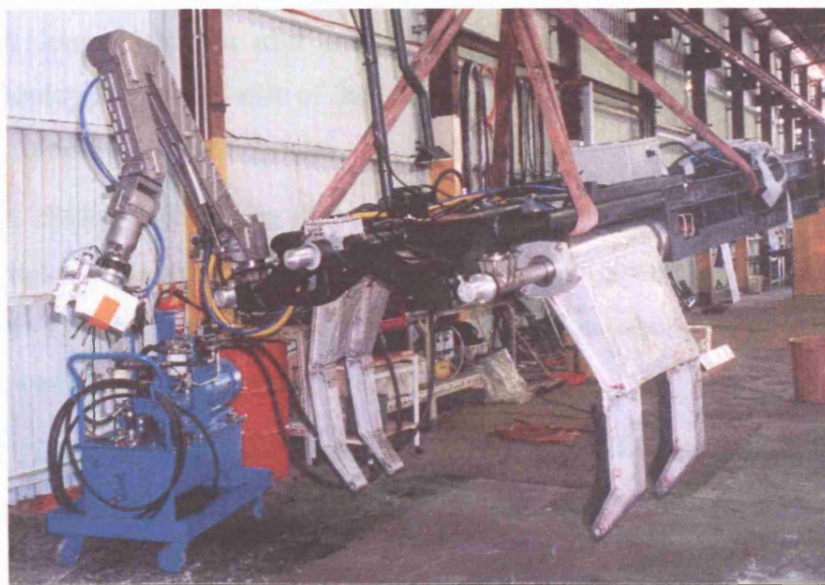


Figure 8.6 – The Covus NICS toolskid



The Covus NICS skid uses two Schilling Titan II or Titan III manipulators which are medium reach and fairly dextrous subsea manipulators. All the subsea ARM equipment is mounted in a toolskid capable of being carried on a work-class “ROV of opportunity”. This toolskid is an aluminium box and tubular frame structure mounting the following equipment (see Figure 8.6):

- 1) Dual shoulder rotate deployment systems. These consist of Titan manipulator mounting points on rotate actuators, one on each side of the skid. They can each rotate the manipulator shoulders through 180 degrees. This allows the arm to reach into work sites that the ROV cannot access, and enables the arms to work as easily on their side, or upside down.
- 2) Attachment claw. This consists of hydraulic fingers mounted on each side of the toolskid. They are opened and closed under control from the ROV cabin and allow the toolskid to be clamped onto the top of braces or anodes.
- 3) Integral inspection equipment, optional valve packs, etc. so that the only links required to the ROV, apart from the physical interface, are an umbilical communication link and a hydraulic supply.
- 4) Pressure vessels for the manipulator controller, ACFM inspection system and toolskid controller.
- 5) A hydraulic extending measurement probe to measure the distance from the front of the toolskid to the node to speed up registering the node position in the ARM Software.
- 6) A long hydraulic arm mounting a pan/tilt/rotate camera which can be deployed over the side of the toolskid for monitoring manipulator operations, including under the brace.
- 7) A sliding mechanism that allows the ROV to fly with the toolskid pushed back underneath it (so that it is balanced in the water) but which allows the toolskid to extend forwards of the ROV when docking (so that the ROV is clear of any overhanging braces).

As part of the work a manipulator mount (proposed by the author) was built for carrying the inspection probe, a touch switch for ARM to conduct workpiece modelling (see *Section 2.6.2. ARM Description*), and a small camera for viewing the positioning of the probe on the weld for inspection – see Figure 8.7.

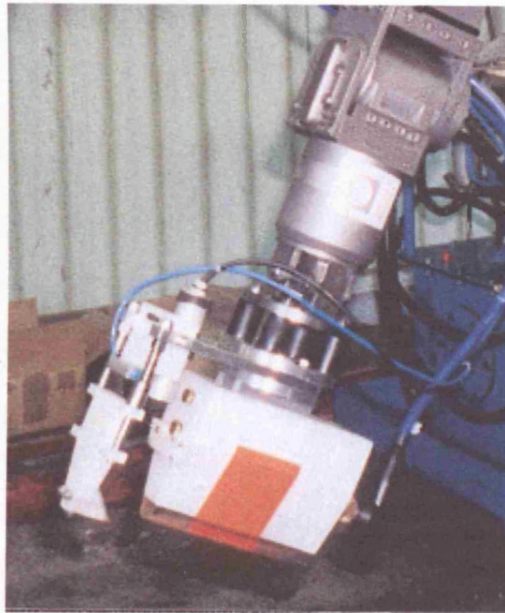


Figure 8.7 – The camera/probe mounting with touch switch and 45° mirror at left

### 8.3. Docking Planning for Elf

The first commercial job to use all features of the automated docking planner (including consideration of attachment legs) was actually the largest and most significant access check done with ARM until the culmination of the development of the ARM NICS system in the Australian work that is the subject of the following chapter.

Elf still required significant amounts of weld inspection to be conducted on the Claymore Alpha platform, the small RACAL system evaluated above (see *Section 5.4.2. RACAL Manipulator Evaluation*) having proven to be inadequate to the task. Elf therefore put out an Invitation to Tender (ITT) for the work so that a number of offshore ROV companies could propose different systems (typically ROV/manipulator combinations) to conduct the work. The significant point is that Elf required each of them to have their systems simulated in the ARM software in order to provide an objective assessment of the access capabilities of each system. This information, combined with estimates of likely system cost and work rate, would be used to determine which company would get the contract.

Four companies eventually put forward systems and had them evaluated in ARM: RovTech Limited, Sonsub International Limited, DSND Subsea Limited and Subsea Offshore Limited (SSOL). The interim results were passed back to each company with proposed changes to each system to improve its access. Eventually a final, detailed report was issued to each company and these were included in that company's bid to Elf.



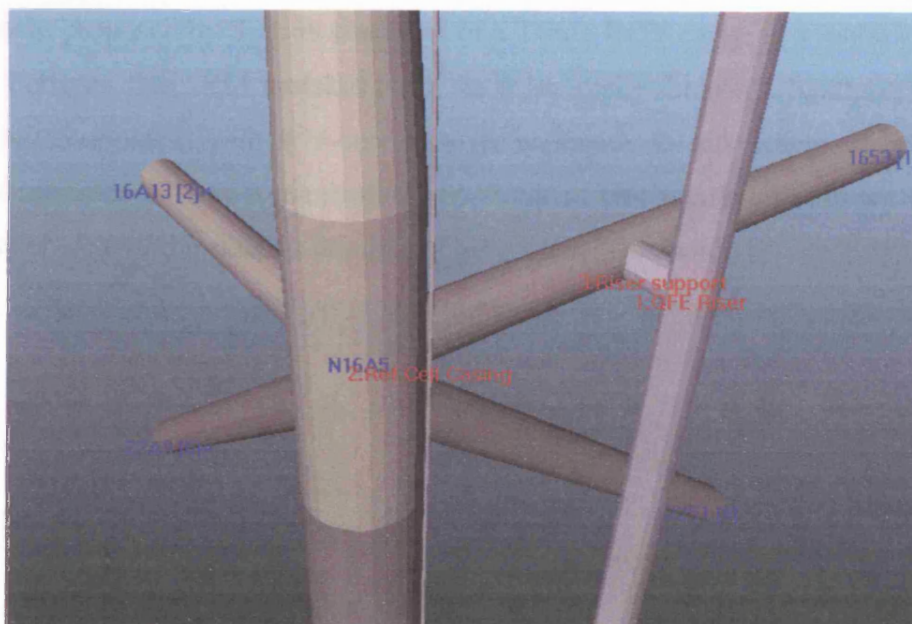


Figure 8.8 – ARM model of the node 16A5 (welds to be accessed are marked thus\*)

All four systems were evaluated for access on two welds (on braces 16A13 and 22A9) on the same node, 16A5, on the Claymore Alpha platform (see Figure 8.8).

#### 8.4. Competing Systems

The system proposed by RovTech consisted of a Spartan ROV carrying a toolskid similar to the ARM toolskid but with a Slingsby Hydrus manipulator on a fixed mount (rather than a rotating/extending boom); this is shown in Figure 8.9.

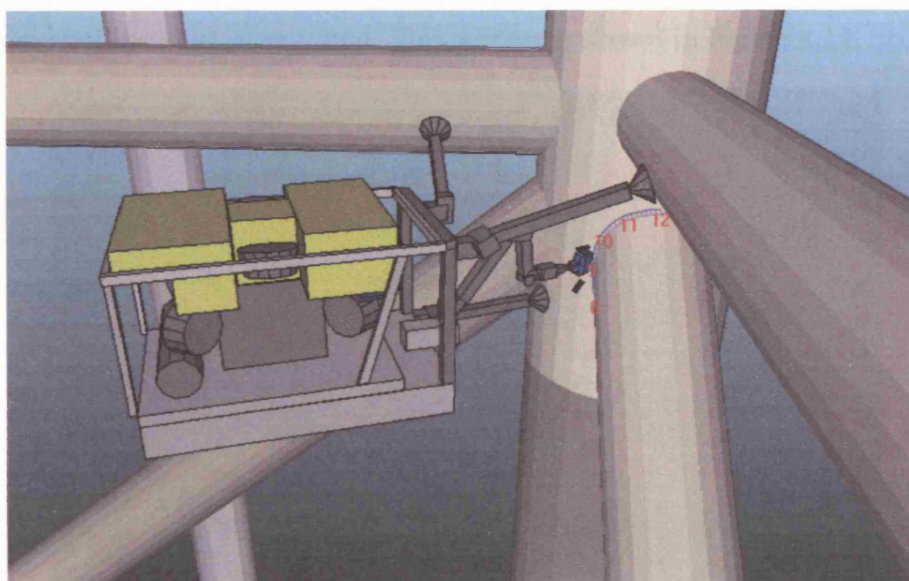


Figure 8.9 – RovTech system inspecting 9 o'clock on weld 22A9

The system proposed by Sonsub consisted of a Triton ROV carrying a toolskid that was a cross between the ARM toolskid (with its three sticky feet on a 'goalpost') and the Covus NICS toolskid with a V-cutout in its underside to aid sitting on braces. The toolskid mounted a Titan 3 manipulator on either of two rotating manipulator mounts, one on each side of the skid, as required. This system is shown in Figure 8.10.

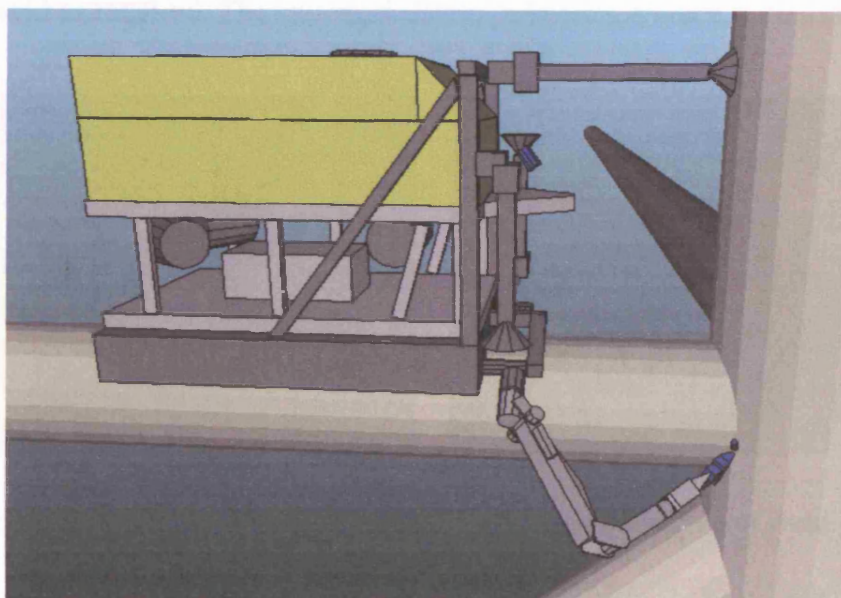


Figure 8.10 – Sonsub system inspecting 4 o'clock on weld 16A13; the manipulator is rolled over to 125°

The system proposed by SSOL consisted of a Pioneer HD ROV carrying a new toolskid that was similar to the ARM skid but using an SSOL design of attachment legs. The toolskid mounted a Titan 3 manipulator on either of two fixed manipulator mounts, one on either side of the skid, as required. This system is shown in Figure 8.11.

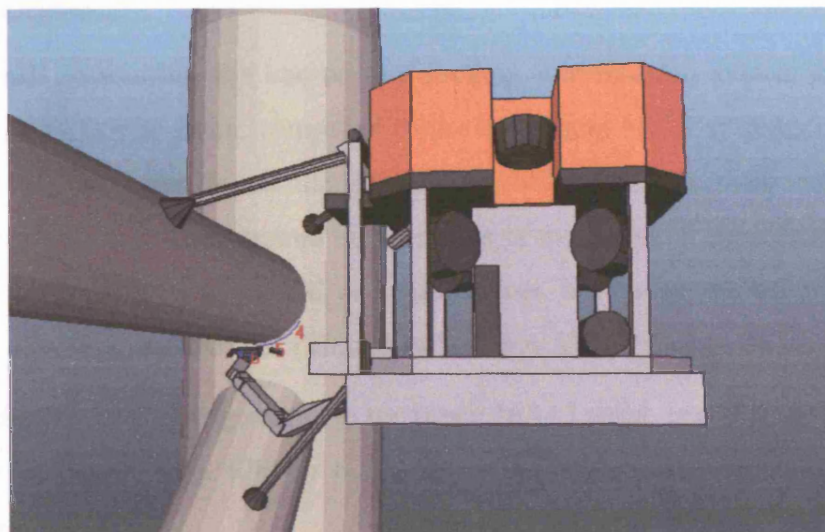


Figure 8.11 – SSOL system inspecting 6 o'clock on weld 16A13; manipulator is mounted upside down



The system proposed by DSND was radically different to those proposed by the other companies, and to any other system previously considered in ARM as it consisted of a telescopic manipulator mounted on a neutrally buoyant ROV. It was proposed that the ROV was held onto the brace by its thrusters while small wheels would rotate the ROV around the brace (including going upside down) so as to move the manipulator around the brace weld as required. The system is shown inspecting a brace weld in Figure 8.12.

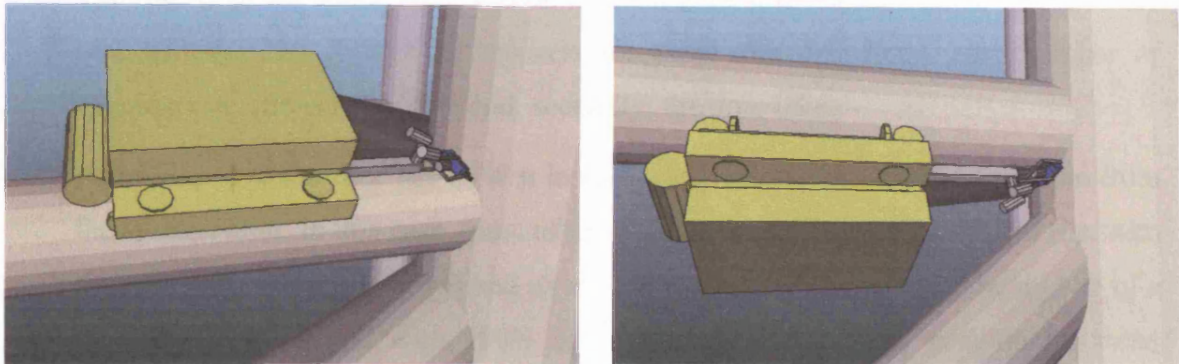


Figure 8.12 – DSND system inspecting the top and underside of weld 16A13

Because this system involved the ROV changing its position and orientation during the access task it could not be considered by the automated docking planner. However, the docking planner could be used for all three other systems, and was used throughout.

Since the other three systems all had attachment legs, and were not constrained to just sitting on the centre of a horizontal brace like the NICS skid, they used all features of the docking optimisation system and so are worthy of closer examination.

#### **8.4.1. Example Use**

It is worth examining the use of the docking optimisation system on the SSOL system in particular since, compared to the ARM and NICS systems considered so far, this had different attachment legs (of an SSOL design), different manipulator mounting system (a fixed mount on each side of the toolskid, capable of holding the manipulator upright, on its side, or upside down, but not at any other angle) and a slightly different manipulator (Titan 3 rather than ARM or Titan 2).

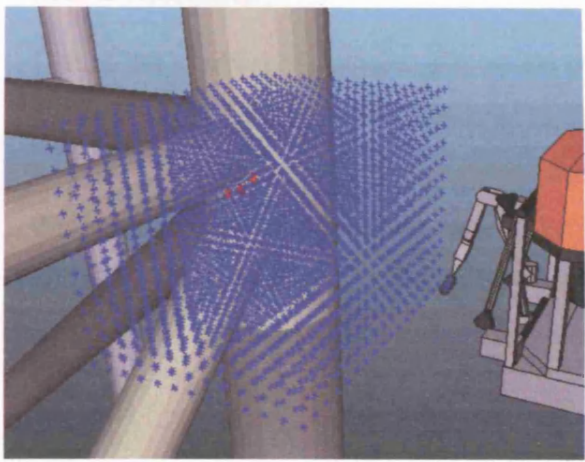
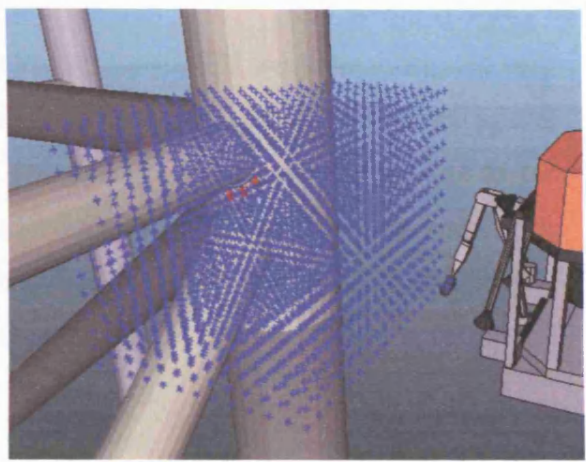
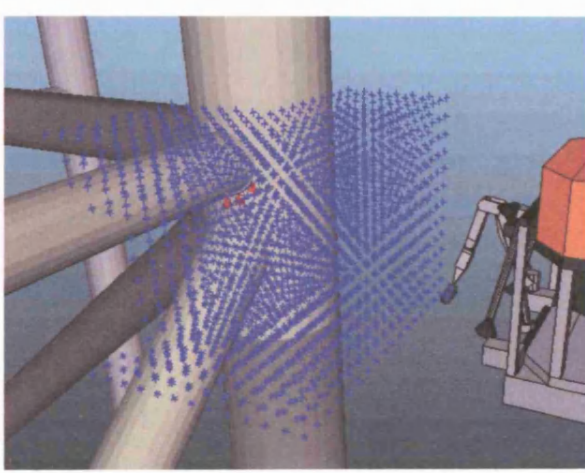
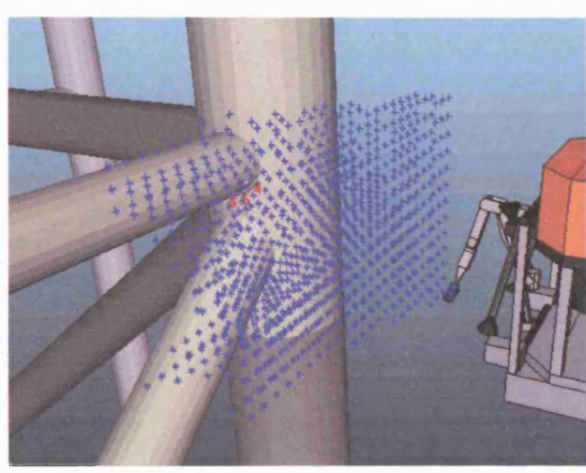
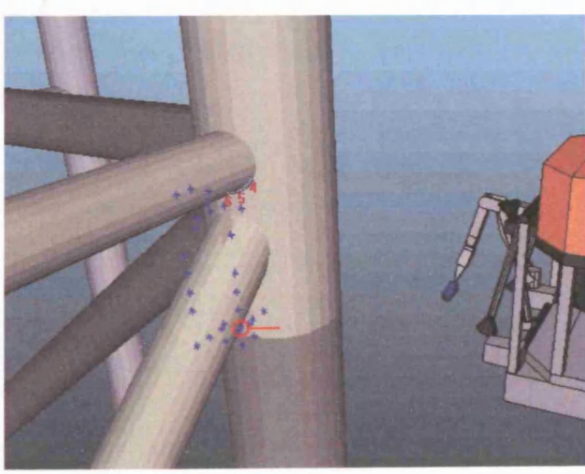
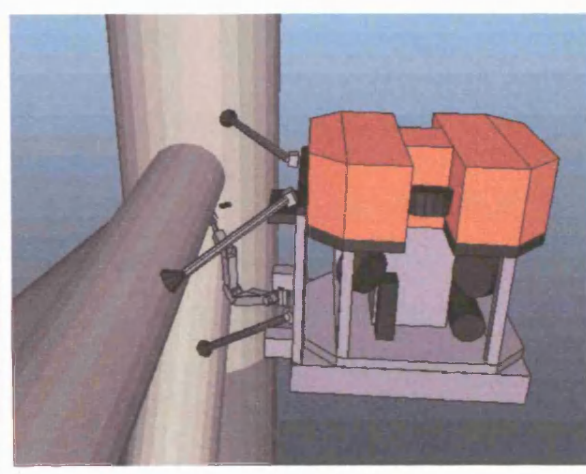
For example, considering access to the brace 16A13 weld, in order to find a position suitable for inspecting 4.00 to 6.00, the following parameters were used in the docking planner:

Grid Spacing:	0.25m
Acceptance Limit:	100% (since the aim was full access)
X, Y and Z ranges:	-3 to 0, 0 to 4, and -2 to 1 respectively
Extend Step:	Not used, since there was no extend mechanism
Rotate Step:	90 degrees

The results of the check are given on the next page (the required clock positions are highlighted). The figures in brackets for each step are, firstly, the number of candidate positions remaining and, secondly, the time taken.

After Step five the user has to step in and select a suitable candidate position from those remaining. In this case, most of the candidates used a rotation of 180 (i.e. with the manipulator upside down) and so one with this configuration, in the middle of a group of positions all with 100% access, was selected. This position has been highlighted in Step 5, determined by following a process of deduction; a useful improvement to the system would be a means for automatically highlighting a chosen position (see *Section 10.4.* ). An inspection of 5.00 is shown being conducted from this position in Step 6.

The complete docking optimisation process (Steps 1-5) took less than two and a half minutes on a 1.4GHz PC. From experience, it is known that doing the same process manually takes about an order of magnitude more so the time savings are considerable. Considering the whole access simulation process, including creation of the required workpiece and equipment models, and subsequent reporting of results, and so on (which do not benefit from the automated optimisation) it is estimated that a typical automated access check that includes attachment legs would be two to five times faster than a manual check. This is better than the actual result (estimated at two times faster) from the work described earlier in this chapter because the NICS system is constrained in position and orientation on a horizontal brace which makes manual planning easier than the general case using attachment legs.

	
<p>1: Creation of candidate positions (2873, &lt;1s)</p>	<p>2: Remove positions inside workpiece (2506, &lt;1s)</p>
	
<p>3: Remove positions causing collisions (1914, 13s)</p>	<p>4: Remove positions if feet cannot attach (980, 16s)</p>
	
<p>5: Remove positions without 100% access (29, 107s)</p>	<p>6: User selects a position from remaining candidates</p>

## **8.5. Conclusions**

Use of the automated docking planner on the Woodside and Elf work demonstrated very effectively its ability to find optimum docking positions very quickly, estimated at two to five times faster than manual planning. The system was equally applicable to the three ROV/manipulator combination systems competing for the Elf work, and to the NICS toolskid, as it was to the ARM toolskid at which it was originally aimed. The next chapter will describe its use on an operational offshore job.



---

## CHAPTER 9: OPERATIONAL USE OF THE AUTOMATED DOCKING PLANNER

---

### 9.1. Introduction

The results of the docking planning described in the previous chapter led Woodside not just to proceed with procurement of the full NICS toolskid but also adaptation of the ARM software to control its Titan manipulators. This led to manipulator Factory Acceptance Tests (FATs) in Seascale, England, which demonstrated that the ARM Software was capable of conducting ACFM weld inspection using a Titan 3 manipulator, see Figure 9.1.



Figure 9.1 – FATs in England of ARM Software controlling a Titan 3 manipulator

This was followed by full system FATs in Perth, Australia – see Figure 9.2. With the ARM System and toolskid connected together for the first time it was possible to confirm operation of the manipulators under computer control, the feedback of the shoulder rotate system, and the operation of the probe mounting and touch switch.



Figure 9.2 – Factory Tests in Australia of ARM Software controlling a Titan 2 manipulator on the NICS skid

Following successful FATs, the complete system went offshore in May 2000 only to remain on deck for the duration of the operation due to poor weather conditions. Finally, after remobilising in September 2000, the system was able to go to work inspecting nodes on NRA as planned. The offshore operation made extensive use of the automated docking planner both in advance of, and during, the work and will be described in this chapter.

## **9.2. ROV Support Vessel**

The system was mobilised on the North West Shelf Gas Project support vessel "Shelf Supporter" (see Figure 9.3) which is operated by a Woodside subsidiary, Mermaid Sound Port and Marine Services Pty Ltd. The Shelf Supporter is a dynamic positioning, 60m long, modified ME202 multi-role vessel, specifically intended to support ROV operations and provide other essential support functions. It has a built-in ROV control room, storage/workshop area, 3m diameter moonpool (for the secondary ROV) and accommodation for 11 personnel, in addition to the marine crew of 14, for 24 hour ROV operations [Batten 1988].





Figure 9.3 – Shelf Supporter ROV support vessel, forward and aft views

The layout of the Shelf Supporter, circa 1988, is shown in Figure 9.4 (by 2000 the Scimitar vehicle and LARS had been removed and the RCV150 secondary ROV replaced by a Scorpion).

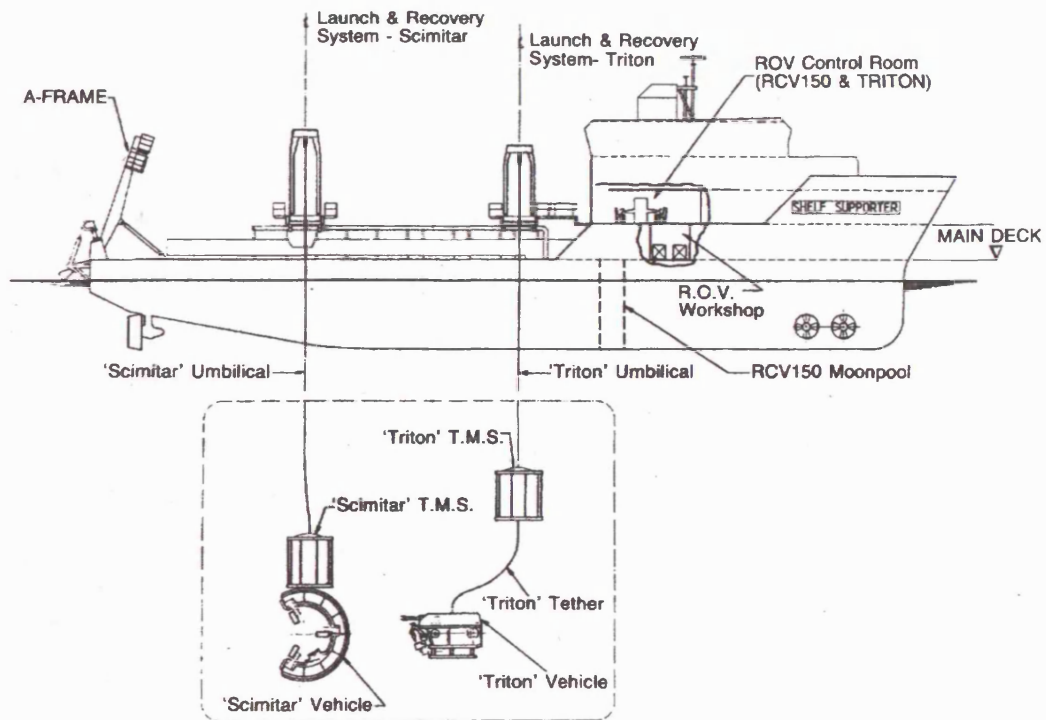


Figure 9.4 – Shelf Supporter ROV support vessel [from Batten 1988]

The primary ROV, a Perry Tritotech Triton, is deployed over the side using a large A-frame LARS (see Figure 9.5); this allows the deployment of the ROV with a top-hat TMS, and large underslung toolskids, such as NICS.



Figure 9.5 – The ROV station on Shelf Supporter, with the launch system (orange) folded over the Triton ROV (yellow)

The computer facilities for the ARM operators consisted of two systems:

- The primary ARM computer, a laptop, located in the ROV control cabin (see Figure 9.6) and provided with a trackball, plus a secondary LCD display (for viewing by the ROV pilot and other crew). This was connected to the manipulator master arm connector, for computer control of the manipulator (which is done by mimicking the commands sent from a master arm). It was also connected to the toolskid electronics in order to receive the feedback from the rotary actuators (for the manipulator mounts) and from the touch switch on the probe mounting.
- A secondary ARM computer, a desktop configuration, was located in an adjacent room and used to plan and consider changes to docking positions based on problems encountered or on new information as it was received (e.g. new or changed anode locations); this could be done by one operator using the automated docking planner while the other operated the ARM control system.



Figure 9.6 – ROV control room with ARM laptop computer in the foreground

### **9.3. Operations**

The work was conducted by Covus Corporation Pty Ltd with inspection personnel provided by SureSpek ISS Pty Ltd, both based in Perth, Australia, but with the manipulator computer controlled by the author and, for the second half of the work, a colleague, from General Robotics Limited, England.



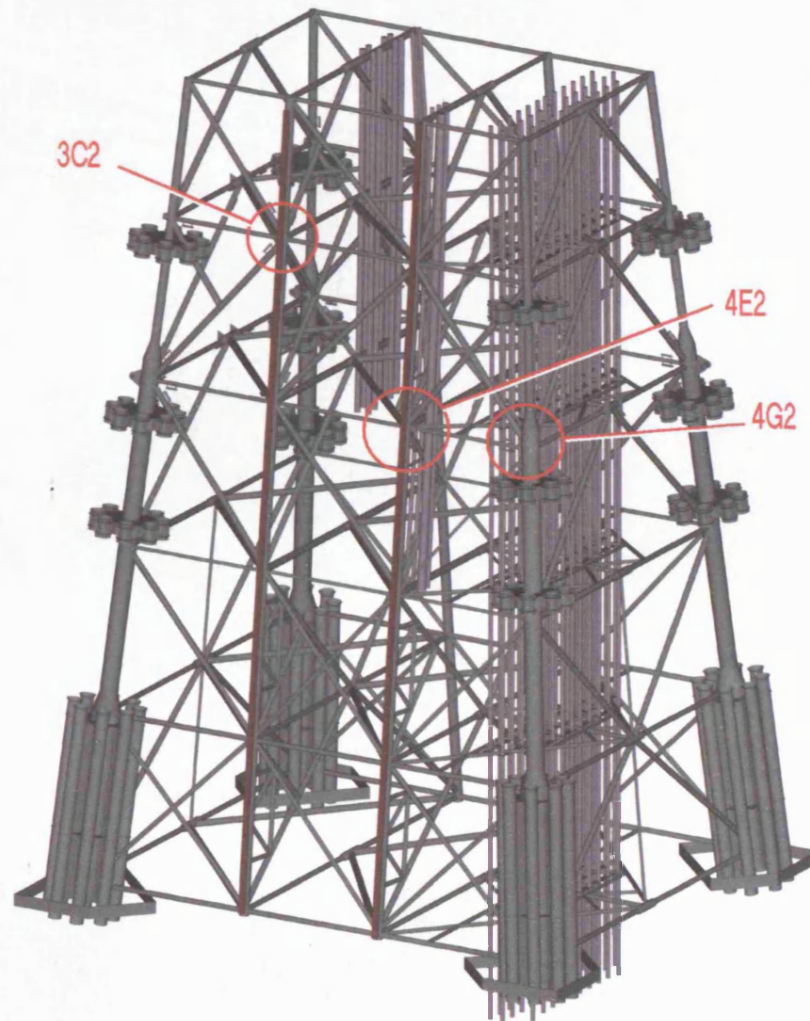


Figure 9.7 – East face of North Rankin Alpha showing inspected nodes  
[courtesy T. Heale]

Figure 9.7 shows the locations of the three nodes that were inspected; these were done in the order 4E2, 4G2, 3C2. The NICS system is shown being deployed in Figure 9.8. The view through its pan and tilt camera is shown in Figure 9.9 while docked on node 4G2. The equivalent arrangement in the ARM software is shown in Figure 9.10.



Figure 9.8 – Triton ROV mounting the NICS skid being deployed beside North Rankin

A full breakdown of the offshore operations is given at Appendix F, concentrating on the access achieved for ACFM inspection.



Figure 9.9 – Pan and tilt camera view showing inspection of 5.00 position on node 4G2 (manipulator is rolled over to 135°)



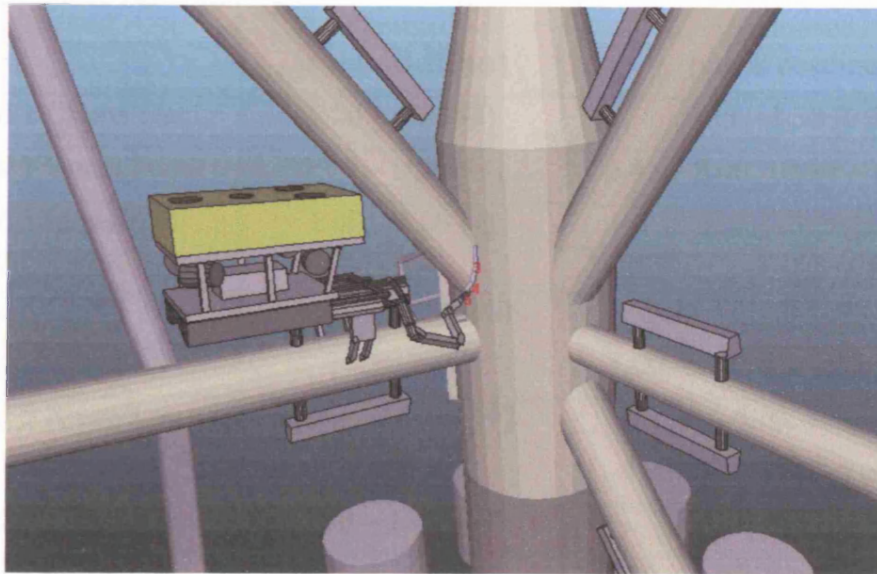


Figure 9.10 – Equivalent ARM view (weld from 2.30 to 5.00 is highlighted)

#### **9.4. Results**

Although it had been intended to inspect five welds, time had run out before three were fully complete. A summary of the results is given in Table 9.1. Although it is tempting to draw conclusions from these results with regard to the accuracy of planning resulting from the use of the ARM Software in general, and the automated docking planner in particular, it is not possible to do so with any confidence.

Specifically, the major reason for not achieving full access was simply the lack of time and it is the author's belief that full planned access could have been achieved given sufficient time (with the possible exception of the area around 10.00 on 4G2/Weld 5 where an unexpected bracket was found, obstructing access to the weld). With increasing time constraints during the operational work, priority was given to attempting access on the most straightforward parts of as many welds as possible, rather than aiming to achieve full access on fewer welds. The Planned full access considered the use of a number of different configurations (e.g. different manipulator shoulder and probe spacers and angled brackets) which improved access but were costly in time and could not be used during the offshore work because of time constraints. In particular, to save time the toolskid was reconfigured as little as possible, so often inspection was attempted with a configuration that was known not to be optimum but which could be used on a number of welds. This is why, for example, the right-angled probe mounting was not used until the last day.

	Planned <sup>1</sup> [clock positions]	Achieved [clock positions]
3C2/Weld 1 Chord Toe	5.00 – 1.00 [8]	6.45 – 1.30 [6.75]
3C2/Weld 1 Brace Toe	7.00 – 11.00 [4]	6.45 – 8.00, 10.30 – 11.15 [2]
3C2/Weld 1 Interstitial Weld Cap	1.00 – 5.00 [4]	1.00 – 3.00 [2]
4G2/Weld 5 Chord Toe	1.00 – 11.00 [10]	2.30 – 5.00, 7.00 – 9.00 5.00 – 7.00 (90° mount) [6.5]
4G2/Weld 5 Brace Toe	1.30 – 10.30 [10]	2.30 – 5.00, 7.30 – 8.30 5.00 – 7.30 (90° mount) [6]
4E2/Weld 8 Chord Toe	1.30 – 10.30 [9]	1.30 – 10.30 [9]
4E2/Weld 8 Brace Toe	1.30 – 10.30 [9]	1.30 – 5.00, 6.30 – 10.30 [7.5]

Table 9.1 – Summary of Planned versus Achieved weld access

A significant result was that no cracks were found in any of the weld segments inspected. This was clearly a very welcome finding from a safety point of view, with regard to the structural integrity of the platform and the personnel living and working on it. Unfortunately, it brought into question the requirement for future inspection with the NICS system, which necessarily had a significant cost attached to it, and may be one of the reasons why the planned follow-up inspection programme for the next year was cancelled.

## 9.5. Conclusions

The NICS system was deployed in September 2000 to clean and inspect nodal welds on North Rankin. It was very successful, proving to be up to ten times faster than manually controlled manipulator weld inspection that had been conducted in the North Sea. Due to many operational reasons, including equipment reliability (resulting largely from using a very old ROV) and vessel availability, the NICS system was in place inspecting welds for just 36 hours out of the total operational duration – however, in that time it inspected some 12m of weld metal (representing at least 250 probe readings) a feat that surpasses any other ROV weld inspection system.

---

<sup>1</sup> From General Robotics document GRL/TJL/093Welds "Examination of Weld Access for Inspection on the Woodside North Rankin A Platform Using the ARM Software Simulation System – Appendix One: Revised Access Checks".

### *Chapter 9: Operational Use of the Automated Docking Planner*

This excellent result was due in part to the success of the automated docking planner in finding optimum docking positions in a practical amount of time, including times offshore where operational constraints required docking planning to be conducted at short notice (i.e. the toolskid is currently in such-and-such a configuration – what else can be inspected before it is brought back up?). It may also be the case that it was the timely availability of the planner during the original access simulation work that allowed the whole operation to go ahead.



### 10.1. Summary of Results

Chapter 1 described how ROVs have been increasingly used to conduct underwater intervention tasks, in place of divers and submersibles, and showed that they are the only unmanned systems able to attach themselves onto jacket nodes to conduct inspection. In conjunction with a robotic control system (such as those described in Chapter 2) they are capable of conducting automated nodal weld inspection using techniques such as ACFM. A number of similar systems are described, primarily REMO, ATES and ARM.

Chapter 3 described the background and development of neural networks, and their application to manipulator control and offshore systems. It also described the development of new neural network software, NNW. This was extensively tested and verified in Chapter 4 which also described the four main types of neural network (interactive activation and competition, constraint satisfaction, pattern associator and back propagation) giving details of the theory behind them and their implementation in NNW. It showed techniques to help avoid networks getting caught in local maxima (particularly through the use of the Boltzmann Machine and an annealing schedule) and gave a full description of NNW features. It concluded by analysing the differences between the NNW results and those published for the well known PDP software.

Chapter 5 described various methods for docking with ROVs, looked at the background for conducting access checks for ROVs, and showed how the ARM software was developed so that it could initially conduct access checks but could later be used for manual docking planning, with various examples given. It concluded by describing the procedure for determining a docking position manually. Chapter 6 looked at using the neural network software to conduct docking planning on two manually defined scenarios (a single attachment leg coincident with the manipulator and a single attachment leg offset to port side of the ROV). Using the Schema Model on the second

scenario resulted in the system quite often getting stuck in local maxima. Using the Boltzmann Machine instead, and a sufficiently long annealing schedule, it was possible to reach the global maximum in all tests conducted.

Chapter 7 described the development of a software library to conduct automated docking planning using some existing features of the ARM Software and adding others. As well as being able to replicate the manual definitions created in the previous chapter the final version could also take account of manipulator kinematic access to a selected weld segment, correct kinematic attachment of the legs to the workpiece, collision detection of the complete ROV model with the workpiece, and use of any deployment extension and rotation functions. The Chapter also described the features of the automated planner in detail and investigated the time taken by each stage of the processing. Although the work was originally intended to produce a pre-processing phase before the use of the neural network system, after successive enhancements it functioned very well on its own and no further development of the neural network method took place.

Chapter 8 described the use of the automated docking planner on two real-life scenarios, one for Woodside for an Australian platform and one for Elf for a North Sea platform. These jobs indicated a significant increase in speed and efficiency when using the automated planner compared to manual planning. Chapter 9 describes an offshore operation which had been planned in advance largely through the use of the automated system, as described in the previous chapter, but also during which the automated system was used extensively for short term and speedy docking planning.

## **10.2. Neural Network Software**

Completely new Neural Network software was developed which could be configured for four different types of network structure (interactive activation and competition, constraint satisfaction, pattern associator and back propagation) and could solve problems in all these formats. The constraint satisfaction network was most appropriate for docking optimisation and was used successfully to select the best ROV location in a number of simplified docking scenarios. However, it required large matrices for representing candidate docking locations and for each constraint being considered, and it became increasingly impractical for large, complex docking scenarios. For this

reason, development of a neural network solution for docking optimisation was stopped in favour of a numerical automated planner.

The neural network software, nonetheless, worked well and could be used effectively on problems outside the docking domain. It could therefore be developed in the future as a general neural network tool, in which case it would be appropriate to consider the following improvements:

1. Add in the neural network subtypes not implemented (such as the Auto Associator, and the cascaded feed-forward, recurrent, sequential and competitive learning variants of the BP type); otherwise the dialog box handling of the parameters for these subtypes (see *Section C.4. Settings Menu*) should be removed.
2. Complete the implementation of a native file format for NNW. Using an appropriate binary format it may be possible to produce files capable of defining large numbers of positions and weights without the size and speed overheads of the PDP text format files.
3. Add in a general graphical feedback system, an idea that was considered during the work but not implemented. This would provide a graphical representation of the network and the unit activations and show the flow of activation along the connections between units, possibly through the use of colour like a contour plot. It could provide valuable feedback of the state of any network, for example showing the success of training and the sequence of operations during training and running, and also highlighting any areas of under-use or saturation.
4. Implement a definable data type that can alternately represent *float* or *double* (or other types such as *long double*) and use this throughout the NNW libraries, including all uses for local variables. It would then be possible to conduct more detailed comparisons of the effects of using different data types on the behaviour of the networks (as begun in *Section 4.10. Discussion of Deviations*) simply by redefining this type.

### 10.3. Automated Docking Software

By the end of the development of the docking software the system was able to successfully produce a shortlist of good docking locations in complex scenarios

considering a significant range of constraints. It did this by conducting the following procedure:

1. Provide a number of optimisation parameters that the operator can set for a particular scenario, e.g. grid size and granularity, required percentage of weld length accessed, and deployment system increments.
2. Create a grid of candidate docking positions based on the given parameters.
3. Remove all positions in contact with the workpiece.
4. Move the ROV to each position in turn and conduct the following checks.
5. Remove all positions that would cause the ROV to be in collision with the workpiece.
6. Calculate an attachment leg value for each position (the ratio of the number of legs successfully attached to the total number of legs on the ROV). Prune away any positions with a ratio below a given threshold.
7. Cycle the deployment system through its range at each position (typically extension then rotation) using the specified increments.
8. For each configuration calculate a manipulator access value (using the ratio of the amount of weld that could be reached to the amount requested by the operator) taking account of the manipulator's forward and inverse kinematic constraints.
9. If a better manipulator access value is found than already held for that position, check to see if the boom is in collision. If not, replace the existing value with the new one, and move to the next configuration until all are checked.
10. Having conducted all checks, prune away any positions with an access value lower than that specified, then display a dialog box listing all remaining positions and giving their extension/rotation values, access value and attachment leg ratio.

All the work considering docking locations was done as part of this research but it made use of existing libraries within the ARM software; this relationship is shown diagrammatically in Figure 10.1.

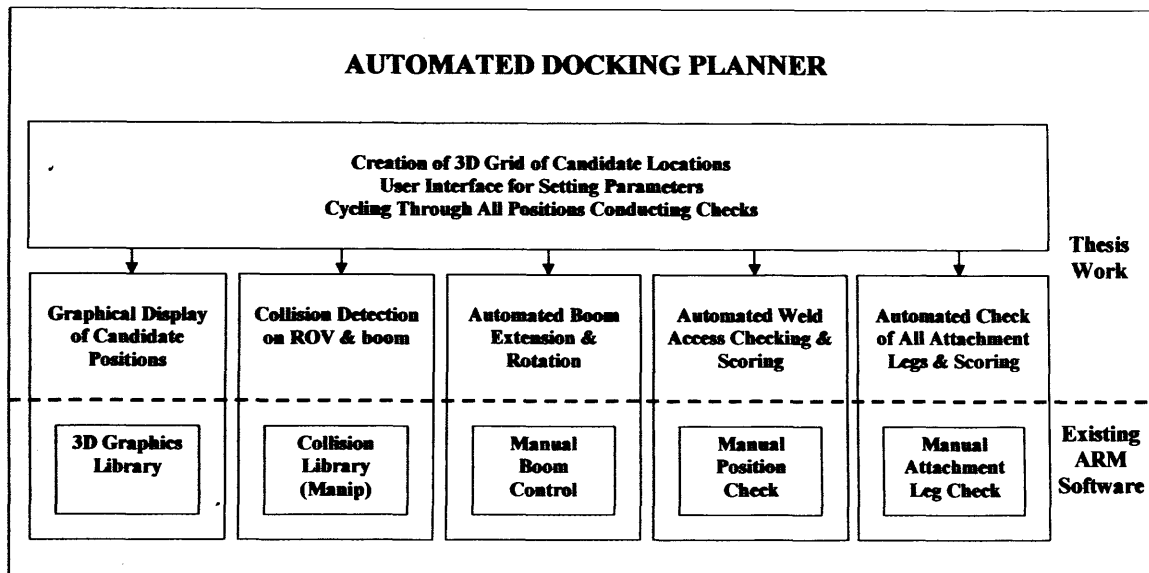


Figure 10.1 – Relationship between thesis work and existing ARM software

The following improvements could be made to the automated docking planner:

1. Currently the results of the planning are a display of remaining candidate positions in the ARM graphics window, and a dialog box listing their positions numerically. However, there is no direct correlation between them – it would be useful to select a position in the list and have it highlighted in the graphics window; even better would be to click on a position in the graphics window and be given its position and other details.
2. At the moment it is possible to set the minimum acceptable weld access value, but it would be useful to also be able to set the minimum acceptable leg attachment ratio (i.e. how many legs are required to be attached).

#### 10.4. Future Development

There may be general improvements that could be made for optimising docking positions from further investigation of standard numerical methods. A brief investigation appeared to show that the most promising methods for this type of problem are actually very similar to those employed here: "annealing methods... have solved some problems previously thought to be practically insoluble; they address directly the problem of finding global extrema in the presence of large numbers of undesired local extrema" [Press 2002]. Standard numerical methods may also have particular application in certain phases of the docking optimisation. For example, the cycling through of all deployment system extension and rotation increments may not be

required. With some investigation, for example plotting the access found for the various extension and rotation values, may suggest a more efficient means of choosing an extension and rotation value.

It may be that neural networks do have a useful part to play in automating docking planning. One idea would be to investigate using neural networks to apply a more interconnected approach to choosing an optimum docking position (as proposed in *Section 7.8. Conclusions*). For example, a more robust choice of position may depend on looking for *clusters* of positions with high access values, and avoiding outlying and single positions.

An alternative approach might be to use a learning type of neural network such as Back Propagation. Although the implementation would be difficult, it should be possible to train the network to come up with suitable docking locations by teaching it with a large set of results from past manual docking procedures.

### 10.5. Summary

This thesis looked at the development of the ROV and its advantages in conducting weld cleaning and inspection compared to other intervention methods, and also looked at how an ROV with a robotic manipulator is able to conduct advanced NDT inspection. It looked at the different types of ROV docking and at the difficulties of planning docking positions. It developed an automated docking planner that is significantly faster, more efficient and easier than manual planning, one that was able to conduct very complex docking planning for a number of different ROV systems on a range of complex underwater nodes. Using this planner it is now possible to quickly determine which nodes on a platform are economically worthwhile to inspect by ROV and which are not, and to quantify the access possible with given ROV/manipulator combinations. In addition, it is possible to determine the best toolskid configuration to launch with, and the best attachment leg arrangement to use including the best position on the node to place each foot. It is possible to determine in advance whether a particular inspection programme is economically viable and also what changes could be made to the proposed ROV system to improve its operational efficiency. Then when the system goes offshore, the planner is able to quickly provide answers to problems encountered, for example considering new information about obstructions at the worksite, or finding new

docking locations for a particular ROV/toolskid configuration that was not initially planned for.

The early work on the planner was based on specially written (though general purpose) neural network software, and this showed it was possible to use a neural network to select from a small number of docking locations in fairly simple scenarios. As the scenarios considered increased in complexity, however, the neural network system became increasingly unwieldy and inefficient and an alternative numerical processing method was developed. Overall, it was found that the numerical approach was more scalable and appropriate than a neural network approach for solving the general problem of the optimisation of docking locations for remotely operated vehicles.

---

## REFERENCES

---

References are given in the following forms:

- The standard form consists of the primary author's surname followed by the year of publication, thus [Ahmad 1989].
- Where there are two authors with the same surname, their first initial is also used, thus [Marsh R. 1999], [Marsh T. 1999].
- Where one author has two references from the same year, an alphabetic suffix is used to distinguish between them, thus [Rumelhart 1986a], [Rumelhart 1986b].
- Where a year of publication is not given, e.g. for commercial marketing brochures, the initials of the publication are given, thus [Sonsub IRST].

- |                        |   |
|------------------------|---|
| <b>Ahmad 1989</b>      | Ahmad Z.<br><i>"Fast Solution to the Inverse Kinematic Problem in Robotics via Multilayered Feedforward Networks"</i><br>MS Thesis, Drexel University, Philadelphia, USA, June 1989   |
| <b>Ahmad 1990</b>      | Ahmad Z. and Guez A.<br><i>"On the Solution to the Inverse Kinematics Problem"</i><br>Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 1990), Cincinnati, USA, May 1990                              |
| <b>Albus 1975</b>      | Albus J.S.<br><i>"A New Approach to Manipulator Control: The Cerebellar Model Articulation Control (CMAC)"</i><br>Transactions of the ASME, Journal of Dynamic Systems, Measurement and Control, September 1975, Volume 97, pp220-227 |
| <b>Aleksander 1990</b> | Aleksander I. and Morton H.<br><i>An Introduction to Neural Computing</i><br>Pub. Chapman & Hall, London, England, 1990, ISBN 0-412-  |



37780-2

- Allerton 1998** Allerton M., Larkum T.J., Lucas W. and Gibson D.  
*"Diverless Robot Wet Flux Cored Arc Welding"*  
 Proceedings of the Offshore Mechanics and Arctic Engineering Conference, Lisbon, Portugal, July 1998
- Aust 1988** Aust E., Dos Santos J.F., Bohm K.-H. and Hensel H.-D.  
*"Mechanized Hyperbaric Welding by Robots"*  
 Proceedings of Intervention '88, Bergen, Norway, April 1988, pp514-525
- Aust 1995** Aust E., Niemann H.-R., Böke M., Gustmann M. and Wesche A.  
*"Six-Years Development in Subsea Robotics"*  
 Proceedings of Underwater Intervention 1995 Conference, Houston, Texas, USA, January 1995, pp158-169
- Baker 1990** Baker J.H.A.  
*"Operational Implications of ROV Hydrodynamic Design"*  
 Proceedings of International Offshore Inspection Repair and Maintenance 1990 (IRM90) incorporating Remotely Operated Vehicle 1990 (ROV90) Exhibition and Conference, Aberdeen, Scotland, November 1990, Day 3
- Batten 1988** Batten C.J.  
*"Offshore Underwater IMR Including Marine Fouling Removal by ROV"*  
 Proceedings of Intervention '88, Bergen, Norway, April 1988, pp399-427
- Bell 1996** Bell C., Bayliss M. and Warburton R.  
*Handbook for ROV Pilot/Technicians*  
 Pub. Oilfield Publications Limited, Ledbury, Herefordshire, England, 1996  
 ISBN 1-870945-85-9
- Bishop 1993** Bishop J.M., Mitchell R.J. and Warwick K.  
*"Neural Networks in Automation Procedures"*  
 Proceedings of the Advanced Robotics and Intelligent Machines Conference (Research Seminar in Robotics and Potential for Exploration), Manchester, England, March 1993
- Blake 1989** Blake B., *Foreword*  
*Jane's Underwater Warfare Systems First Edition 1989-90*

- Ed. Blake B.  
Pub. Jane's Information Group, Coulsdon, Surrey, England,  
1989  
ISBN 0-7106-0884-5
- Boddy C. 1993** Boddy C.L., Hopper D.J.F. and Taylor J.D.  
*"Advanced Control Systems for Robotic Arms"*  
Proceedings of the Advanced Robotics and Intelligent Machines  
Conference (Research Seminar in Robotics and Potential for  
Exploration), Manchester, England, March 1993
- Boddy L. 1994** Boddy L., Wilkins M.F., Morris C.W., Tarran G.A., Burkill P.H.  
and Jonker R.R.  
*"Techniques for Neural Network Identification of Phytoplankton  
for the EurOPA Flow Cytometer"*  
Proceedings of OCEANS 94 Conference, Brest, France,  
September 1994, Volume I, pp565-569
- Bowen 1995** Bowen M.F. and Fletcher B.  
*"High-Resolution, Wide-Area Dam Inspections by Automated  
ROV"*  
Proceedings of Underwater Intervention 1995 Conference,  
Houston, Texas, USA, January 1995, pp144-157
- Brambilla 1996** Brambilla M.  
*"ATES Test Results"*  
Proceedings of ROV Technologies Conference, Aberdeen,  
Scotland, December 1996
- Broome 1986** Broome D.R.  
*"Intelligent Manipulators for Automated Subsea Inspection"*  
Proceedings of the IEE Colloquium on Robotics in NDT,  
October 1986
- Broome 1988** Broome D.R. and Greenshields M.C.  
*"A Path Following Controller for ROVs"*  
Proceedings of Intervention '88, Bergen, Norway, April 1988,  
pp646-647
- Broome 1989** Broome D.R.  
*"Improved Underwater Inspection Using Robotic  
Manipulators"*  
Proceedings of the SUT Conference on Advances in Underwater  
Inspection and Maintenance, Aberdeen, Scotland, May 1989

- Broome 1991** Broome D.R.  
*"Improved Supervisory Control System for Subsea Robotic Manipulators"*  
Proceedings of Intervention '91, Florida, USA, May 1991
- Broome 1993a** Broome D.R. and Larkum T.J.  
*"Graphical User Interface for an Advanced Telerobotic Control System"*  
Proceedings of the United Kingdom Simulation Society (UKSS) Conference, Keswick, England, September 1993
- Broome 1993b** Broome D.R., Larkum T.J. and Hall M.S.  
*"ARM Project: Software Control System"*  
Proceedings of Subtech '93, Aberdeen, Scotland, November 1993, pp35-41 (Volume 31 of Society for Underwater Technology: Advances in Underwater Technology, Ocean Science and Offshore Engineering)  
pub. Kluwer Academic Publishers, Dordrecht, Netherlands, 1993  
ISBN 0-7923-2544-3
- Broome 1994** Broome D.R. and Hall M.S.  
*"Advanced Telerobotic Control System"*  
Eds. Jamshidi M., Yuh J., Nguyen C.C. and Lumia R.  
Intelligent Automation and Soft Computing, TSI Press, 1994, Volume 2, pp411-418
- Broome 1995a** Broome D., Larkum T. and Hall M.  
*"Subsea Weld Inspection Using an Advanced Robotic Manipulator"*  
Proceedings of OCEANS 95 Conference, San Diego, California, USA, October 1995, Paper 154
- Broome 1995b** Broome D.R., Larkum T.J. and Hall M.S.  
*"Inspection of Subsea Nodal Welds by the ARM Robot Manipulator"*  
Proceedings of Subtech '95: SUT Conference on Addressing the Subsea Challenge, Aberdeen, Scotland, November 1995
- Broome 1996** Broome D.R. and Larkum T.  
*"Offshore Platform Inspection using ROVs"*  
Proceedings of ROV Technologies Conference, Aberdeen, Scotland, December 1996
- Brutzman 1995** Brutzman D.

- "Virtual World Visualization for an Autonomous Underwater Vehicle"*  
 Proceedings of OCEANS 95 Conference, San Diego, California, USA, October 1995, Paper 232
- Carre 1991** Carre M., Grignon P., Lefevre J-M.  
*"Chaine de Commande Generique pour le Pilotage Telerobotique des Porteurs en Bol de Generateur de Vapeur (A Command Sequence for the Control of Maintenance Robots in Nuclear Power Plants)"*  
 Proceedings of ORIA 91 Conference (Telerobotics in Hostile Environments: The Major Technical Bottlenecks), Marseille, France, December 1991, pp233-238
- Çavuşoğlu 2001** Çavuşoğlu M.C., Sherman A. and Tendick F.  
*"Bilateral Controller Design for Telemanipulation in Soft Environments"*  
 Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2001), Seoul, Korea, May 2001
- Chardard 2002** Chardard Y.  
*"Work AUV for Deep Water Intervention: Dream or Reality"*  
 Proceedings of Underwater Intervention 2002 Conference, New Orleans, Louisiana, USA, February/March 2002
- Cherruel 1994** Cherruel G., Autret Y. and Dupont F.  
*"Using SIGNAL for Developing Neural Control Systems"*  
 Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume III, pp116-120
- Clegg 1995** Clegg A.C., Dunnigan M.W. and Lane D.M.  
*"Force Control and Modelling of Hydraulic Underwater Manipulators"*  
 Proceedings of the IEE International Workshop on Advanced Robotics and Intelligent Machines, Salford, England, April 1995, pp1-8
- Cohn 1994** Cohn D.A., *"Neural Network Exploration Using Optimal Experiment Design"*  
*Advances in Neural Information Processing Systems 6*  
 Eds. J. Cowan et al  
 Morgan Kaufmann 1994
- Colina-Morles 1993** Colina-Morles E. and Mort N.  
*"Neural Network-Based Adaptive Control Design"*

- Journal of Systems Engineering, Springer-Verlag, England, January 1993, Volume 3 No. 1 pp9-14
- Dapper 1997** Dapper M., Maaß R., Zahn V. and Eckmiller R.  
*"Neural Force Control (NFC) for Complex Manipulator Tasks"*  
 Proceedings of the International Conference ICANN97, Lausanne, Switzerland, October 1997, pp787-792
- Dapper 1998** Dapper M., Maaß R., Zahn V. and Eckmiller R.  
*"Neural Force Control (NFC) Applied to Industrial Manipulators in Interaction with Moving Rigid Objects"*  
 Proceedings of IEEE International Conference on Robotics and Automation (ICRA 1998), Leuven, Belgium, 1998, pp2048-2053
- Declercq 1994** Declercq F., Dumortier F., De Keyser R. and Van Cauwenberghe A.  
*"Real-Time Control of a Robot Using Neural Networks"*  
 Proceedings of the Third IEEE Conference on Control Applications, Glasgow, Scotland, August 1994, Volume 2, pp1061-1066
- Dhruv 2000** Dhruv N. and Tendick F.  
*"Frequency Dependence of Compliance Contrast Detection"*  
 Proceedings of the ASME Dynamic Systems and Control Division, DSC-Vol. 69-2, 2000, Volume 2
- Dissanayake 1993** Dissanayake M.W.M.G.  
*"Neural Network Based Distance Functions for Robot Obstacle Avoidance"*  
 Journal of Systems Engineering, 1993, Volume 3 No. 1, pp1-8, Springer-Verlag, England
- Djouani 1994** Djouani K. and Hamam Y.  
*"Ship Optimal Path Planning and Artificial Neural Nets for Berthing"*  
 Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume I, pp785-790
- Dotan 1991** Dotan O.  
*"The Role of 3D Graphic Software Tools in Teleoperation"*  
 Proceedings of ORIA 91 Conference (Telerobotics in Hostile Environments: The Major Technical Bottlenecks), Marseille, France, December 1991, pp92-101
- Drolet 2000** Drolet L., Michaud F. and Côté J.

- "An Adaptable Navigation System for an Underwater ROV"*  
 Proceedings of PRECARN-IRIS International Symposium on Robotics (ISR), Montréal, Canada, 2000, pp244-245
- Dubrovsky 1994** Dubrovsky N.A. and Rimskey-Korsakova L.K.  
*"A Simulation Network of First Order Auditory Neurons for Preprocessing of Acoustic Signals"*  
 Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume II, pp235-238
- Duncan 1990** Duncan N.  
*"The Development of ROV NDT Tooling"*  
 Proceedings of International Offshore Inspection Repair and Maintenance 1990 (IRM90) incorporating Remotely Operated Vehicle 1990 (ROV90) Exhibition and Conference, Aberdeen, Scotland, November 1990, Day 4
- Dunnigan 1993** Dunnigan M.W. and Lane D.M.  
*"Evaluation and Reduction of the Dynamic Coupling Between an ROV and Manipulator"*  
 Proceedings of the IEE Colloquium on the Control and Guidance of Underwater Vehicles, December 1993
- Dunnigan 1996** Dunnigan M.W., Lane D.M., Clegg A.C. and Edwards I.  
*"Hybrid Position/Force Control of a Hydraulic Underwater Manipulator"*  
 IEE Proceedings, Control Theory and Applications, March 1996, Volume 143 No. 2, pp145-151
- El-Hawary 1994** El-Hawary F. and Li J.  
*"Artificial Neural Network for Additive Noise Filtering Techniques"*  
 Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume I, pp323-329
- Ellis 1994** Ellis R., Simpson R., Culverhouse P.F., Parisini T., Williams R., Reguera B., Moore B. and Lowe D.  
*"Expert Visual Classification and Neural Networks: Can General Solutions Be Found?"*  
 Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume I, pp330-334
- Elsharkawi 1992** Elsharkawi A. and Reinisch H.  
*"Neural Based Controller for Robotic Fine-manipulation"*  
 Journal of Systems Engineering, Springer-Verlag, England,

1992, No. 2 pp224-231

**Even 1991**

Even P. and Fournier R.

*"Interactive Geometric Description and Task Specification for Intervention Robotics"*

Proceedings of ORIA 91 Conference (Telerobotics in Hostile Environments: The Major Technical Bottlenecks), Marseille, France, December 1991, pp163-172

**Evensen 1988**

Evensen G.

*"Diverless Underwater Inspection"*

Proceedings of Intervention '88, Bergen, Norway, April 1988, pp428-444

**Feng 1994**

Feng X., Qiu R. and Yun D.Y.Y.

*"A Graphic Simulator for Autonomous Underwater Vehicle Docking Operations"*

Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume I, pp152-157

**Fletcher 1995**

Fletcher B. and Greelish S.

*"Precision Automation of ROV Inspections for the Nuclear Industry"*

Proceedings of Underwater Intervention 1995 Conference, Houston, Texas, USA, January 1995, pp79-83

**Fletcher 1997**

Fletcher B.

*"Talon: A Uniquely Integrated Remotely Operated Vehicle"*

Proceedings of Underwater Intervention 1997 Conference, Houston, Texas, USA, February 1997, pp102-107

**Garmulewicz 2000**

Garmulewicz J.A.

*"The Application of the AUto-ROV<sup>TM</sup> System in Deepwater Field Developments"*

Proceedings of Underwater Intervention 2000 Conference, Houston, Texas, USA, January 2000, Paper D.1.2.

**Gibson 2002**

Gibson J. and English J.

*"The U.S. Navy ADS2000"*

Proceedings of Underwater Intervention 2002 Conference, New Orleans, Louisiana, USA, February/March 2002

**Given 1991**

Given D, Ed.

*ROV Review, Fourth Edition, 1991-92*

Pub. Windate Enterprises Incorporated, Spring Valley,

- California, USA, 1991  
ISBN 0-9623145-2-8
- Glasius 1995** Glasius R., Komoda A. and Gielen S.  
*"Neural Network Dynamics for Path Planning and Obstacle Avoidance"*  
Neural Networks journal, 1995, Volume 8 (1), pp125-133
- Gomes 1995** Gomes J. and Barraso V.  
*"Blind Equalization Using A Radial Basis Function Neural Network"*  
Proceedings of OCEANS 95 Conference, San Diego, California, USA, October 1995, Paper 101
- Graham 1991** Graham D.P.W. and D'Eleuterio G.M.T.  
*"Robotic Control Using a Modular Architecture of Cooperative Artificial Neural Networks"*  
Proceedings of the International Conference ICANN91, Espoo, Finland, June 1991
- Grant 1993** Grant E.  
*"Machine Intelligence"*  
Proceedings of the Advanced Robotics and Intelligent Machines Conference (Research Seminar in Robotics and Potential for Exploration), Manchester, England, March 1993
- Greig 1989** Greig A.R. and Broome D.R.  
*"Tactile Sensing of Complex Geometry Workpieces"*  
Proceedings of the IFAC/IFIP Conference INCOM89 (International Conference on Problems in Manufacturing Technology), Madrid, Spain, September/October 1989
- Greig 1992** Greig A.R.  
*"Automatic Inspection of Complex Geometry Welds"*  
PhD Thesis, University College, University of London, England, April 1992
- Greig 1994** Greig A.R. and Broome D.R.  
*"Development of a Robotic Underwater Manipulator"*  
Transactions of the Institute of Marine Engineers (IMarE), May 1994, Volume 106 Part 5, pp217-229
- Guo 1995** Guo J., Chiu F.C. and Wang C.-C.  
*"Adaptive Control of an Autonomous Underwater Vehicle Tested Using Neural Networks"*



- Proceedings of OCEANS 95 Conference, San Diego, California, USA, October 1995, Paper 132
- Hallset 1994** Hallset J.O. and Berre G.  
“*Modular Integrated Man-Machine Interaction and Control (MIMIC)*”  
Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume II, pp108-112
- Hallset 2000** Hallset J.O.  
“*Automated ROV Manoeuvring*”  
Proceedings of Underwater Intervention 2000 Conference, Houston, Texas, USA, January 2000, Paper A.1.3.
- Hansen 1988** Hansen R.K. and Bjørnø L.  
“*Underwater Robotics – a Flexible Robot Concept with Novel Acoustic Sensing Abilities*”  
Proceedings of Intervention '88, Bergen, Norway, April 1988, pp369-379
- Harbur 1999** Harbur S.  
“*Force Feedback Manipulator Systems: The Myths / The Facts*”  
Proceedings of Underwater Intervention 1999 Conference, Houston, Texas, USA, January 1999, Paper H.2.force
- Harman 1988** Harman M.V.  
“*Advanced Remotely Operated Work System (AROWS)*”  
Proceedings of Intervention '88, Bergen, Norway, April 1988, pp223-243
- Hartley 1992** Hartley D.W.  
“*The Multi-Role Vehicle (MRV) – ROV for the Future*”  
Proceedings of IOCE 92 Conference, Aberdeen, Scotland, October 1992, Day 2
- Hartley 1993** Hartley D.W., Clapham P.D. and Dowkes W.M.  
“*ARM Project: System Hardware*”  
Proceedings of Subtech '93, Aberdeen, Scotland, November 1993, pp19-33 (Volume 31 of Society for Underwater Technology: Advances in Underwater Technology, Ocean Science and Offshore Engineering)  
Pub. Kluwer Academic Publishers, Dordrecht, Netherlands, 1993  
ISBN 0-7923-2544-3

- Hattori 1988** Hattori M., Nomoto M. and Aoki T.  
*"Sea Going Tests of Deep ROV, Dolphin-3K"*  
Proceedings of Intervention '88, Bergen, Norway, April 1988,  
pp610-619
- Hayward 1991** Hayward A., *"The ROV Industry in Review"*  
*ROV Review, Fourth Edition, 1991-92*  
Ed. Given D.  
Pub. Windate Enterprises Incorporated, Spring Valley,  
California, USA, 1991  
ISBN 0-9623145-2-8
- Headworth 1988** Headworth C. and Dines C.  
*"An Operational Subsea Wireline System"*  
Proceedings of Intervention '88, Bergen, Norway, April 1988,  
pp380-396
- Heale 1999** Heale T. and Larkum T.J.  
*"ARM and Rovsim: Extending Our Reach"*  
*Industrial Robot Journal*  
Ed. Loughlin C.  
Pub. MCB University Press, Bradford, West Yorkshire, England,  
1999, Volume 26 No. 3, pp202-208  
ISSN 0143-991X
- Hinton 1986** Hinton G.E. and Sejnowski T.J., *"Learning and Relearning in Boltzmann Machines"*  
*Parallel Distributed Processing – Explorations in the Microstructure of Cognition, Volume 1: Foundations*  
Ed. Rumelhart D.E., McClelland J.L. and the PDP Research Group  
Pub. MIT Press, Massachusetts, USA, 1986, pp282-317  
ISBN 0-262-18120-7
- Höglund 1988** Höglund K., Husebye R., Lindland H. and Nesse E.  
*"Wellman – a Multipurpose Intervention System"*  
Proceedings of Intervention '88, Bergen, Norway, April 1988,  
pp173-184
- Hopfield 1982** Hopfield J.J.  
*"Neural Networks and Physical Systems with Emergent Collective Properties"*  
Proceeding of the National Academy of Science USA 79, 1982,  
227

pp2554-2558.

**Hornfeld 2002**

Hornfeld W.

*“DeepC: the German Development Project for an Intelligent Deep Sea Robot”*

Proceedings of Underwater Intervention 2002 Conference, New Orleans, Louisiana, USA, February/March 2002

**Hsu 1994**

Hsu L., Costa R.R., Lizarralde F., da Cunha J.P.V.S., Scieszko J.L., Romanov A.V., Wollmann D. Jr. and Sant'Anna A.C.M.

*“Underwater Vehicle Dynamic Positioning Based on a Passive Arm Measurement System”*

Proceedings of the 2<sup>nd</sup> Workshop on Mobile Robots for Subsea Environments, IARP, Monterey, California, USA, 1994

**Hsu 1999**

Hsu L., Costa R.R., Lizarralde F. and da Cunha J.P.V.S.

*“Passive Arm Based Dynamic Positioning System for Remotely Operated Underwater Vehicles”*

Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 1999), Detroit, Michigan, USA, 1999

**Hughes 1988**

Hughes G. and Broome D.R.

*“Automated Deployment of Weld Inspection Systems for Tubular Nodes”*

Proceedings of Intervention '88, Bergen, Norway, April 1988, pp261-275

**Ingebretsen 2002**

Ingebretsen E.

*“Cost Effective and Safe Intervention with ROV: Technical and Managerial Aspects”*

Proceedings of Underwater Intervention 2002 Conference, New Orleans, Louisiana, USA, February/March 2002

**Johansen 2000**

Johansen S.

*“Added Safety and Increased Efficiency Using Computer Visualization Technology in the Underwater Intervention Industry”*

Proceedings of Underwater Intervention 2000 Conference, Houston, Texas, USA, January 2000, Paper A.1.1.

**Johansen 2001**

Johansen S.

*“DP Navigation using Virtual Reality Technology in Underwater Intervention Operations”*

Proceedings of Underwater Intervention 2001 Conference, Tampa, Florida, USA, January 2001

- Johnson 1990** Johnson M.A. and Leahy M.B. Jr.  
*"Adaptive Model-Based Neural Network Control"*  
 Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 1990), Cincinnati, USA, May 1990
- Jordan 1986** Jordan, M.I., *"An Introduction to Linear Algebra in Parallel Distributed Processing"*  
*Parallel Distributed Processing – Explorations in the Microstructure of Cognition, Volume 1: Foundations*  
 Ed. Rumelhart D.E., McClelland J.L. and the PDP Research Group  
 Pub. MIT Press, Massachusetts, USA, 1986, Chapter 9, pp365-422  
 ISBN 0-262-18120-7
- Kamenev 1995** Kamenev O.T.  
*"Training Two-Layer Neural Network Model for Tomography Data Processing"*  
 Proceedings of OCEANS 95 Conference, San Diego, California, USA, October 1995, Paper 294
- Kasabov 1993** Kasabov N.K.  
*"Hybrid Connectionist Production Systems: An Approach to Realising Fuzzy Expert Systems"*  
 Journal of Systems Engineering, Springer-Verlag, England, January 1993, Volume 3 No. 1 pp15-21
- Kato 1996** Kato N. and Lane D.M.  
*"Co-ordinated Control of Multiple Manipulators in Underwater Robots"*  
 Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 1996), Minneapolis, USA, April 1996, pp2505-2510
- Kenzie 1990** Kenzie B.W., Mudge P.J., Lank A.M., Koch B.E., Christensen J.R. and Jellesen E.K.  
*"Development of Mechanised Ultrasonic Flaw Detection Technology for Underwater Inspection of Complex Geometry Joints in Offshore Tubular Structures"*  
 Proceedings of International Offshore Inspection Repair and Maintenance 1990 (IRM90) incorporating Remotely Operated Vehicle 1990 (ROV90) Exhibition and Conference, Aberdeen, Scotland, November 1990, Day 2
- Koval 1994** Koval E.V.

- "Automatic Stabilization System of Underwater Manipulation Robot"*  
Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume I, pp807-812
- Lane 1991** Lane D.M., Dunnigan M.W., Knightbridge P.J. and Quinn A.W.  
*"Planning and Control for Co-ordination of Underwater Manipulators"*  
Proceedings of the IEEE International Conference, Edinburgh, Scotland, March 1991
- Lane 1994** Lane D.M. and Quinn A.W.  
*"Computational Issues in Motion Planning for Autonomous Underwater Vehicles with Manipulators"*  
Proceedings of the IEEE Oceanic Engineering Soc. Symposium on Autonomous Underwater Vehicle Technology, Boston, USA, July 1994, pp255-262
- Lane 1995** Lane D.M. and Knightbridge P.J.  
*"Task Planning and World Modelling for Supervisory Control of Robots in Unstructured Environments"*  
Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 1995), Nagoya, Japan, May 1995
- Langrock 1993** Langrock D.G.  
*"ARM Project: Requirements for the Manipulator System"*  
Proceedings of Subtech '93, Aberdeen, Scotland, November 1993, pp15-18 (Volume 31 of Society for Underwater Technology: Advances in Underwater Technology, Ocean Science and Offshore Engineering)  
Pub. Kluwer Academic Publishers, Dordrecht, Netherlands, 1993  
ISBN 0-7923-2544-3
- Langrock 1994** Langrock D.G. and Broome D.R.  
*"Advanced Telerobotic Controller"*  
Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume II, pp157-162
- Larkum 1994a** Larkum T. and Broome D.  
*"Advanced Controller for an Underwater Manipulator"*  
Proceedings of the Third IEEE Conference on Control Applications, Glasgow, Scotland, August 1994, Volume 2, pp1081-1086

- Larkum 1994b** Larkum T.J., Broome D.R., Maddocks I. and Hartley D.  
*"Graphical User Interface for an Advanced Telerobotic Control System"*  
Proceedings of the Emerging Technologies in Advanced Robotics (ETAR 94) Conference, Windermere, England, September 1994
- Larkum 1996a** Larkum T.  
*"Subsea Weld Inspection Using the ARM System"*  
*Underwater Contractor*  
Ed. Bevan J.  
Pub. Resort Marketing and Publishing Limited, Weymouth, Dorset, England, November/December 1996, p12  
ISSN 1362-0487
- Larkum 1996b** Larkum T.J. and Broome D.R.  
*"Nel Centro Nazionale Iperbarico di Aberdeen Testato l'Ultimo Robot-Saldatore Sottomarino"*  
*COCIS (Comitato Città Sotterranea)*  
Ed. Villoresi G.  
Pub. Associazione per l'Utilizzo del Sottosuolo, Milan, Italy, October 1996, No. 22/23 Anno 5, pp34-40
- Larkum 1998** Larkum T.J. and Broome D.R.  
*"ARM: A Proven System for the Inspection of Subsea Welds"*  
*Measurement + Control*  
Ed. O'Brien C. and Carter C.  
Pub. Institute of Measurement and Control, London, England, April 1988, Volume 31 No. 3, pp68-72  
ISSN 0020-2940
- Larkum 2000** Larkum T.J. and Hall M.S.  
*"ROV Control Software Finally Goes Modular"*  
*International Ocean Systems*  
Ed. Barton R.  
Pub. Astrid Powell Associates, Twickenham, England, January/February 2000, Volume 4 No. 1  
ISSN 1460-4982
- Larkum 2002** Larkum T.  
*"ROV Simulation, Visualization and Supervisory Control"*  
Proceedings of Underwater Intervention 2002 Conference, New

Orleans, Louisiana, USA, February/March 2002

**Last 1991**

Last G. and Williams P.

*An Introduction to ROV Operations*

Pub. Oilfield Publications Limited, Ledbury, Herefordshire, England, 1991

ISBN 1-870945-23-9

**Lemoine 1995**

Lemoine D., Schilling R. and Rasmussen D.

*"Precision Automation of ROV Inspections for the Nuclear Industry"*

Proceedings of Underwater Intervention 1995 Conference, Houston, Texas, USA, January 1995, pp79-83

**Lots 2000**

Lots J.-F., Lane D.M. and Trucco E.

*"Application of 2½D Visual Servoing to Underwater Vehicle Station-Keeping"*

Proceedings of OCEANS 2000 Conference, Rhode Island, USA, September 2000, pp1257-1264

**Maaß 1998**

Maaß R., Dapper M. and Eckmiller R.

*"Neural Trajectory Optimization (NTO) for Manipulator Tracking of Unknown Surfaces"*

Springer Proceedings of International Conference on Artificial Neural Networks (ICANN98), Skovde, Sweden, 1998, pp893-898

**Mair 1990**

Mair J.

*"Standard Methods Applied for Cost Effective ROV Intervention on Subsea Production Systems"*

Proceedings of International Offshore Inspection Repair and Maintenance 1990 (IRM90) incorporating Remotely Operated Vehicle 1990 (ROV90) Exhibition and Conference, Aberdeen, Scotland, November 1990, Day 3

**Mann 1990**

Mann J.

*"Advanced ROV for Underwater Inspection and Maintenance"*

Proceedings of International Offshore Inspection Repair and Maintenance 1990 (IRM90) incorporating Remotely Operated Vehicle 1990 (ROV90) Exhibition and Conference, Aberdeen, Scotland, November 1990, Day 4

**Marsh R. 1991**

Marsh R.J., *"Divers Forever!"*

*ROV Review, Fourth Edition, 1991-92*

Ed. Given D.

- Pub. Windate Enterprises Incorporated, Spring Valley, California, USA, 1991  
ISBN 0-9623145-2-8
- Marsh R. 1996** Marsh R.J., *Foreword*  
*Remotely Operated Vehicles of the World – '96/7 Edition*  
Ed. Simons J.  
Pub. Oilfield Publications Limited, Ledbury, Herefordshire, England, 1996  
ISBN 1-870945-81-6
- Marsh R. 2000** Marsh R.J., *Foreword*  
*Remotely Operated Vehicles of the World – Fourth Edition*  
Ed. Simons J.  
Pub. Oilfield Publications Limited, Ledbury, Herefordshire, England, 2000  
ISBN 1-902157-16-8
- Marsh R. 2002** Marsh R.J., *Foreword*  
*Remotely Operated Vehicles of the World – Fifth Edition*  
Ed. Simons J.  
Pub. Oilfield Publications Limited, Ledbury, Herefordshire, England, 2002  
ISBN 1-902157-28-1
- Marsh T. 1992** Marsh T.R.  
*“ROV Requirements, Future Needs & Relationships – An Operator’s View”*  
Proceedings of IOCE 92 Conference, Aberdeen, Scotland, October 1992, Day 2
- McLain 1995** McLain T.W., Rock S.M. and Lee M.J.  
*“Experiments in the Coordination of Underwater Manipulator and Vehicle Control”*  
Proceedings of OCEANS 95 Conference, San Diego, California, USA, October 1995, Paper 153
- McClelland 1986** McClelland D.E., Rumelhart D.E. and Hinton G.E., *“The Appeal of Parallel Distributed Processing”*  
*Parallel Distributed Processing – Explorations in the Microstructure of Cognition, Volume 1: Foundations*  
Ed. Rumelhart D.E., McClelland J.L. and the PDP Research Group



- Pub. MIT Press, Massachusetts, USA, 1986, Chapter 1, pp3-44  
ISBN 0-262-18120-7
- McClelland 1988**      McClelland J.L. and Rumelhart D.E.  
*Explorations in Parallel Distributed Processing - A Handbook of Models, Programs, and Exercises*  
MIT Press, Massachusetts, USA, 1988  
ISBN 0-262-63113-X
- McMaster 1994**      McMaster R.S., Nixon J.H., Boyle B.G. and Fouchier D.  
*"Task Enhancement of an Underwater Robotic Arm by Graphical Simulation Techniques"*  
Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume II, pp163-167
- Mejia 1994**            Mejia C., Thiria S., Crépon M. and Badran F.  
*"A Neural Network Approach for Wind Retrieval from the ERS-1 Scatterometer Data"*  
Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume I, pp76-80
- Middleton 1993**      Middleton D.J. and Lyons S.  
*"NEWTSUIT Advanced Diving System"*  
Proceedings of Subtech '93, Aberdeen, Scotland, November 1993, p175 (Volume 31 of Society for Underwater Technology: Advances in Underwater Technology, Ocean Science and Offshore Engineering)  
Pub. Kluwer Academic Publishers, Dordrecht, Netherlands, 1993  
ISBN 0-7923-2544-3
- Miller 1990**            Miller W.T. III, Hewes R.P., Glanz F.H. and Kraft L.G. III  
*"Real-Time Dynamic Control of an Industrial Manipulator Using a Neural-Network-Based Learning Controller"*  
IEEE Transactions on Robotics and Automation, February 1990, Volume 6 No. 1, pp1-9
- Minsky 1969**          Minsky M. and Papert S.  
*Perceptrons: An Introduction to Computational Geometry*  
MIT Press, Massachusetts, USA, 1969
- Mills 1993**            Mills G., Barrett C., Gorman N. and Wagner S.  
*"Operational Experiences with Atmospheric Diving Systems"*  
Proceedings of Subtech '93, Aberdeen, Scotland, November

- 1993, pp177-187 (Volume 31 of Society for Underwater Technology: Advances in Underwater Technology, Ocean Science and Offshore Engineering)  
Pub. Kluwer Academic Publishers, Dordrecht, Netherlands, 1993  
ISBN 0-7923-2544-3
- Moller 1996** Moller E.  
*"Advances in Manipulator Technology: ATES – an Application of the Above Technology"*  
Proceedings of International ROV Forum '96, Aberdeen, Scotland, February 1996
- Morasso 1991** Morasso P. and Solari M.  
*"A Neural Implementation of Analogic Planning Methods"*  
Proceedings of the International Conference ICANN91, Espoo, Finland, June 1991
- MTD 1989** Marine Technology Directorate Limited  
*Underwater Inspection of Steel Offshore Installations: Implementation of a New Approach*  
Pub. Marine Technology Directorate Limited 1989  
ISBN 1-870553-03-9
- Parkes 1996** Parkes S.  
*"ARM Trials Success"*  
*Underwater Contractor*  
Ed. Bevan J.  
Pub. Resort Marketing and Publishing Limited, Weymouth, Dorset, England, July/August 1996, p27  
ISSN 1362-0487
- Pedlow 1996** Pedlow P.  
*"ROVs... Old vs New"*  
Proceedings of ROV Technologies Conference, Aberdeen, Scotland, December 1996
- Pegman 1999** Pegman G.  
*"Automating Teleoperation"*  
*Industrial Robot Journal*  
Ed. Loughlin C.  
Pub. MCB University Press, Bradford, West Yorkshire, England, 1999, Volume 26 No. 3, pp184-187

ISSN 0143-991X

**Pennison 1997**

Pennison N. and Raine G.A.

*"An Alternative to Conventional Diver Weld Inspection"*

*Oil & Gas Gazette*

July 1997, pp27-32

**Press 2002**

Press W.H., Teukolsky S.A., Vetterling W.T. and Flannery B.P.

*"Numerical Recipes in C++: The Art of Scientific Computing"*

Pub. Cambridge University Press, Cambridge, England, 2002,  
Chapter 10: "Minimisation or Maximisation of Functions"

ISBN 0-521-75033-4

**Pretlove 1999**

Pretlove J.

*"Teleroobotics: Merging Man With Machine"*

*Industrial Robot Journal*

Ed. Loughlin C.

Pub. MCB University Press, Bradford, West Yorkshire, England,  
1999, Volume 26 No. 3, pp159-160

ISSN 0143-991X

**Primrose 1988**

Primrose G.M.

*"Deepwater Intervention for Subsea Production Systems"*

Proceedings of Intervention '88, Bergen, Norway, April 1988,  
pp644-645

**Raine 1995**

Raine G.A. and Lugg M.C.

*"ROV Inspection of Welds – a Reality"*

Proceedings of Underwater Intervention 1995 Conference,  
Houston, Texas, USA, January 1995, pp123-129

**Raine 1996a**

Raine G.A.

*"ROV Weld Inspection – the Next Stage"*

Proceedings of International ROV Forum '96, Aberdeen,  
Scotland, February 1996

**Raine 1996b**

Raine G.A. and Lugg M.C.

*"ROV Inspection of Welds – a Reality"*

Proceedings of International ROV Forum '96, Aberdeen,  
Scotland, February 1996

**Raine 1997**

Raine G.A.

*"ROV Weld Inspection with a Mid Size ROV and an ACFM"*

- Array*"
- Proceedings of Underwater Intervention 1997 Conference,  
Houston, Texas, USA, February 1997, pp51-59
- Renard 1988** Renard D.B.
- "ROV/Diverless Interventions on Oseberg Subsea Production Systems"*
- Proceedings of Intervention '88, Bergen, Norway, April 1988,  
pp202-222
- Ricci 1996** Ricci F.
- "Use of Robotics for Inspection of Offshore Structure – Below Water (from R and D Program to Offshore Operation)"*
- Proceedings of International ROV Forum '96, Aberdeen,  
Scotland, February 1996
- Rosenblatt 1962** Rosenblatt F.
- Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*
- Pub. Spartan Books, New York, USA, 1962
- Rumelhart 1986a** Rumelhart D.E., Hinton G.E. and Williams R.J., *"Learning Internal Representations by Error Propagation"*
- Parallel Distributed Processing – Explorations in the Microstructure of Cognition, Volume 1: Foundations*
- Ed. Rumelhart D.E., McClelland J.L. and the PDP Research Group
- Pub. MIT Press, Massachusetts, USA, 1986, Chapter 8, pp318-362
- ISBN 0-262-18120-7
- Rumelhart 1986b** Rumelhart D.E., Smolensky P., McClelland J.L. and Hinton G.E., *"Schemata and Sequential Thought Processes in PDP Models"*
- Parallel Distributed Processing – Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*
- Ed. McClelland J.L., Rumelhart D.E. and the PDP Research Group
- Pub. MIT Press, Massachusetts, USA, 1986, Chapter 14, pp8-57
- ISBN 0-262-13218-4
- Russell 1990** Russell G.T.
- "Intelligent Control for Remotely Operated Vehicles"*
- Proceedings of Subsea Control and Data Acquisition,

- Technology and Experience, London, England, April 1990, pp97-108 (Volume 22 of Society for Underwater Technology: Advances in Underwater Technology, Ocean Science and Offshore Engineering)  
Pub. Kluwer Academic Publishers, Dordrecht, Netherlands, 1990  
ISBN 0-7923-0698-8
- Sanner 1994** Sanner R.M. and Slotine J.-J. E.  
*"Function Approximation, "Neural Networks", and Adaptive Nonlinear Control"*  
Proceedings of the Third IEEE Conference on Control Applications, Glasgow, Scotland, August 1994, Volume 2, pp1225-1232
- Sarzeaud 1994** Sarzeaud O., Stéphan Y. Le Corre F. and Kerléguer L.  
*"Neural Meshing of a Geographical Space in Regard to Oceanographic Data Location"*  
Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume I, pp335-339
- Savut 1985** Savut M., Wray A.M., Ihnatowicz E. and Broome D.R.  
*"Automated Inspection of Subsea Structures"*  
Proceedings of the International Symposium on Developments in Subsea Engineering Systems, London, England, May 1985
- Sayers 1994** Sayers C., Stein M., Lai A. and Paul R.  
*"Teleprogramming to Perform Sophisticated Underwater Manipulative Tasks Using Acoustic Communications"*  
Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume II, pp168-173
- Schilling 1996** Schilling R.  
*"Advances in Manipulator Technology – the Future of Subsea Robotics"*  
Proceedings of International ROV Forum '96, Aberdeen, Scotland, February 1996
- Shirasaki 1988** Shirasaki Y., Asakawa K., Kojima J., Homma H. and Tamura R.  
*"Deep Sea ROV 'Marcas-2500' for Cable Maintenance and Repair Works"*  
Proceedings of Intervention '88, Bergen, Norway, April 1988, pp627-641
- Skyberg 1988** Skyberg Ø.

- "Access Verification Utilizing Computer Aided Design"*  
Proceedings of Intervention '88, Bergen, Norway, April 1988,  
pp655-656
- Slingsby 1996**      *"ARM Trials Success"*  
*Underwater Systems Design*  
Pub. Astrid Powell Associates, Twickenham, England,  
July/August 1996, p39  
ISSN 1460-4982
- Smith 1990**      Smith B.J.  
*"Eureka-EU191 Programme – an Inspection Tool Kit for  
Advanced ROV"*  
Proceedings of International Offshore Inspection Repair and  
Maintenance 1990 (IRM90) incorporating Remotely Operated  
Vehicle 1990 (ROV90) Exhibition and Conference, Aberdeen,  
Scotland, November 1990, Day 4
- Smolensky 1986**      Smolensky P., *"Information Processing in Dynamical Systems:  
Foundations of Harmony Theory"*  
*Parallel Distributed Processing – Explorations in the  
Microstructure of Cognition, Volume 1: Foundations*  
Ed. Rumelhart D.E., McClelland J.L. and the PDP Research  
Group  
Pub. MIT Press, Massachusetts, USA, 1986, Chapter 6, pp194-  
281  
ISBN 0-262-18120-7
- Somers 1992**      Somers T.  
*"Mobil/Marquest ROV Dynamic Positioning Trials"*  
Proceedings of IOCE 92 Conference, Aberdeen, Scotland,  
October 1992, Day 2
- Sonsub ATES RSE**      *"ATES Remote Systems Engineering"*  
Sonsub International  
Marketing material
- Sonsub CN3/1**      *"Sonsub Pioneers Advanced Telerobotic System"*  
Sonsub International  
Challenger Newsletter, Vol.3, Issue 1, p1-2
- Sonsub CN4/2**      *"New Telepresence Technology Promises to Revolutionize ROV  
Operations"*  
Sonsub International

- Challenger Newsletter, Vol.4, Issue 2, p3
- Sonsub IRST**      *"The Innovators of Remote Systems Technology"*  
Sonsub Services Pty. Ltd.  
Marketing material
- Sonsub Triton ROVS**      *"Triton Remote Operated Vehicle Services"*  
Sonsub International  
Marketing material
- Sortland 1990**      Sortland B.  
*"Performance Optimisation of ROV Systems"*  
Proceedings of International Offshore Inspection Repair and Maintenance 1990 (IRM90) incorporating Remotely Operated Vehicle 1990 (ROV90) Exhibition and Conference, Aberdeen, Scotland, November 1990, Day 3
- Stephan 1995**      Stephan Y.  
*"Inverting Tomographic Data with Neural Nets"*  
Proceedings of OCEANS 95 Conference, San Diego, California, USA, October 1995, Paper 231
- Stolt Comex Seaway RR**      *"REMO Robotics"*  
Stolt Comex Seaway  
Marketing material
- Sutton 1994**      Sutton R., Johnson C. and Robert G.N.  
*"Depth Control of an Unmanned Underwater Vehicle Using Neural Networks"*  
Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume III, pp121-125
- Tanaka 1991**      Tanaka K., Shimizu M. and Tsuchiya K.  
*"A Solution to an Inverse Kinematics Problem of a Redundant Manipulator using Neural Networks"*  
Proceedings of the International Conference ICANN91, Espoo, Finland, June 1991
- Terre 1995**      Terre T., Golenzer J. and Solaiman B.  
*"Tracking and Identification of Ray Acoustic Arrivals by Means of Neural Networks"*  
Proceedings of OCEANS 95 Conference, San Diego, California, USA, October 1995, Paper 187



- Thornton 2001** Thornton M.A., Randall R. and Albaugh E.K.  
*"A Survey of Atmospheric Diving Suits from Past to Present"*  
 Proceedings of Underwater Intervention 2001 Conference,  
 Tampa, Florida, USA, January 2001
- Tisdall 1994** Tisdall J., Hall M., Larkum T. and Broome D.  
*"Testing a Stereo Vision System for Motion Compensation"*  
 Proceedings of OCEANS 94 Conference, Brest, France,  
 September 1994, Volume II, pp182-187
- Tisdall 1995** Tisdall J. and Broome D.  
*"Adaptive Force Control for the ARM Manipulator"*  
 Proceedings of OCEANS 95 Conference, San Diego, California,  
 USA, October 1995, Paper 302
- Tisdall 1997** Tisdall J.  
*"Compliant Force Control for Automated Sub-Sea Inspection"*  
 PhD Thesis, University College, University of London, England,  
 August 1997
- Townsend 1999** Townsend W.T. and Guertin J.A.  
*"Teleoperator Slave – WAM Design Methodology"*  
*Industrial Robot Journal*  
 Ed. Loughlin C.  
 Pub. MCB University Press, Bradford, West Yorkshire, England,  
 1999, Volume 26 No. 3, pp167-177  
 ISSN 0143-991X
- Turner 1993** Turner J.  
*"ARM Project: Vision System"*  
 Proceedings of Subtech '93, Aberdeen, Scotland, November  
 1993, pp43-49 (Volume 31 of Society for Underwater  
 Technology: Advances in Underwater Technology, Ocean  
 Science and Offshore Engineering)  
 Pub. Kluwer Academic Publishers, Dordrecht, Netherlands,  
 1993  
 ISBN 0-7923-2544-3
- Van Den Hooff** Van Den Hooff H. and Kronemeijer D.A.  
**1988** *"The Realization of an Automated Underwater Structural  
 Inspection System: a Complex Multidisciplinary Task"*  
 Proceedings of Intervention '88, Bergen, Norway, April 1988,  
 pp276-287

- Van Der Smagt 1991** Van Der Smagt P.P. and Kröse B.J.A.  
*"A Real-Time Learning Neural Robot Controller"*  
 Proceedings of the International Conference ICANN91, Espoo, Finland., June 1991
- Van Der Veen 2000** Van Der Veen N.  
*"AUV Technology: Why? & Will We?"*  
 Proceedings of Underwater Intervention 2000 Conference, Houston, Texas, USA, January 2000, Paper D.1.1.
- Van Der Zwaan 2001** Van Der Zwaan S., Bernardino A. and Santos-Victor J.  
*"Vision Based Station Keeping and Docking for Floating Vehicles"*  
 Proceedings of European Control Conference 2001 (ECC2001), Porto, Portugal, September 2001
- Vinsen 1988** Vinsen R.J.W. and Malone R.B.  
*"ROV Intervention – the Status and the Future"*  
 Proceedings of Intervention '88, Bergen, Norway, April 1988, pp462-476
- Walter 1993** Walter J.A. and Schulten K.J.  
*"Implementation of Self-Organizing Neural Networks for Visuo-Motor Control of an Industrial Robot"*  
 IEEE Transactions on Neural Networks, January 1993, Volume 4 No. 1, pp86-95
- Wang 1994** Wang Q., Greig A.R. and Broome D.R.  
*"Intelligent Gain Scheduling (IGS) using neural networks for robotic manipulators"*  
 Proceedings of the Workshop on Neural Network Applications and Tools, Liverpool, England, September 1993.  
 Pub. IEEE Computer Soc. Press, pp103-108, 1994  
 ISBN 0-8186-5845-2/94
- Westwood 1993** Westwood J.  
*"Man V. Machine – 10 Years On"*  
 Proceedings of Subtech '93, Aberdeen, Scotland, November 1993, pp207-217 (Volume 31 of Society for Underwater Technology: Advances in Underwater Technology, Ocean Science and Offshore Engineering)  
 Pub. Kluwer Academic Publishers, Dordrecht, Netherlands, 1993

ISBN 0-7923-2544-3

**Wray 1985**

Wray A.M.

*"A Computer Controlled Manipulator System for Autonomous Inspection"*

PhD Thesis, University College, University of London, England, 1985

**Wright 2002**

Wright G. and Bamford A.

*"Advanced Long Baseline Acoustics for Precise Deep Water Structure Mating"*

Proceedings of Underwater Intervention 2002 Conference, New Orleans, Louisiana, USA, February/March 2002

**Wüst 1994**

Wüst J.C. and van Noort G.J.H.L.

*"Neural Network Current Prediction for Shipping Guidance"*

Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume I, pp58-63

**Yeung 1989**

Yeung D.-Y. and Gekey G.A.

*"Using a Context-Sensitive Learning Network for Robot Arm Control"*

Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 1989), Scottsdale, USA, May 1989

**Zerr 1994**

Zerr B., Maillard E. and Gueriot D.

*"Sea-Floor Classification by Neural Hybrid System"*

Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume II, pp239-243

---

## APPENDIX A: PUBLICATIONS BY THE AUTHOR

---

- |   |   |   |  |
|---|---|---|--|
| 1 | Allerton M.,<br>Larkum T.J.,<br>Lucas W. and<br>Gibson D. | Diverless Robot Wet Flux<br>Cored Arc Welding                             | Proceedings of the Offshore<br>Mechanics and Arctic<br>Engineering Conference,<br>Lisbon, Portugal, July 1998  |
| 2 | Broome D.R.<br>and Larkum T.J.                            | Graphical User Interface for<br>an Advanced Telerobotic<br>Control System | Proceedings of the United<br>Kingdom Simulation Society<br>(UKSS) Conference, Keswick,<br>England, September 1993  |
| 3 | Broome D.R.,<br>Larkum T.J. and<br>Hall M.S.              | ARM Project: Software<br>Control System                                   | <p>Proceedings of Subtech '93,<br/>Aberdeen, Scotland, November<br/>1993, pp35-41 (Volume 31 of<br/>Society for Underwater<br/>Technology: Advances in<br/>Underwater Technology, Ocean<br/>Science and Offshore<br/>Engineering)</p> <p>pub. Kluwer Academic<br/>Publishers, Dordrecht,<br/>Netherlands, 1993</p> <p>ISBN 0-7923-2544-3</p> |
| 4 | Broome D.,<br>Larkum T. and<br>Hall M.                    | Subsea Weld Inspection<br>Using an Advanced Robotic<br>Manipulator        | Proceedings of OCEANS 95<br>Conference, San Diego,<br>California, USA, October 1995,<br>Paper 154  |
| 5 | Broome D.R.,<br>Larkum T.J. and<br>Hall M.S.              | Inspection of Subsea Nodal<br>Welds by the ARM Robot<br>Manipulator       | Proceedings of Subtech '95:<br>SUT Conference on Addressing<br>the Subsea Challenge,<br>Aberdeen, Scotland, November<br>1995   |
| 6 | Broome D.R.<br>and Larkum T.                              | Offshore Platform Inspection<br>using ROVs                                | Proceedings of ROV<br>Technologies Conference,<br>Aberdeen, Scotland, December<br>1996   |
| 7 | Heale T. and<br>Larkum T.J.                               | ARM and Rovsim: Extending<br>Our Reach                                    | <p><i>Industrial Robot Journal</i></p> <p>Ed. Loughlin C.</p> <p>Pub. MCB University Press,</p>  |

*Appendix A: Publications by the Author*

- |    |   |  |   |
|----|---|--|---|
|    |   |  | Bradford, West Yorkshire,<br>England, 1999, Volume 26 No.<br>3, pp202-208<br>ISSN 0143-991X   |
| 8  | Larkum T. and<br>Broome D.                                    | Advanced Controller for an<br>Underwater Manipulator   | Proceedings of the Third IEEE<br>Conference on Control<br>Applications, Glasgow,<br>Scotland, August 1994, Volume<br>2, pp1081-1086   |
| 9  | Larkum T.J.,<br>Broome D.R.,<br>Maddocks I. and<br>Hartley D. | Graphical User Interface for<br>an Advanced Telerobotic<br>Control System                          | Proceedings of the Emerging<br>Technologies in Advanced<br>Robotics (ETAR 94)<br>Conference, Windermere,<br>England, September 1994   |
| 10 | Larkum T.   | Subsea Weld Inspection<br>Using the ARM System   | <i>Underwater Contractor</i><br>Ed. Bevan J.<br><br>Pub. Resort Marketing and<br>Publishing Limited, Weymouth,<br>Dorset, England,<br>November/December 1996, p12<br>ISSN 1362-0487             |
| 11 | Larkum T.J. and<br>Broome D.R.                                | Nel Centro Nazionale<br>Iperbarico di Aberdeen<br>Testato l'Ultimo Robot-<br>Saldatore Sottomarino | <i>COCIS (Comitato Città<br/>Sotterranea)</i><br>Ed. Villoresi G.<br><br>Pub. Associazione per l'Utilizzo<br>del Sottosuolo, Milan, Italy,<br>October 1996, No. 22/23 Anno<br>5, pp34-40        |
| 12 | Larkum T.J. and<br>Broome D.R.                                | ARM: A Proven System for<br>the Inspection of Subsea<br>Welds                                      | <i>Measurement + Control</i><br>Ed. O'Brien C. and Carter C.<br><br>Pub. Institute of Measurement<br>and Control, London, England,<br>April 1988, Volume 31 No. 3,<br>pp68-72<br>ISSN 0020-2940 |
| 13 | Larkum T.J. and<br>Hall M.S.                                  | ROV Control Software<br>Finally Goes Modular   | <i>International Ocean Systems</i><br>Ed. Barton R.<br><br>Pub. Astrid Powell Associates,<br>Twickenham, England,<br>January/February 2000, Volume<br>4 No. 1<br>ISSN 1460-4982                 |
| 14 | Larkum T.   | ROV Simulation,<br>Visualization and   | Proceedings of Underwater<br>Intervention 2002 Conference,  |

*Appendix A: Publications by the Author*

	Supervisory Control	New Orleans, Louisiana, USA, February/March 2002
15	Tisdall J., Hall M., Larkum T. and Broome D.	Testing a Stereo Vision System for Motion Compensation
		Proceedings of OCEANS 94 Conference, Brest, France, September 1994, Volume II, pp182-187

---

## APPENDIX B:

### DETAILED SOFTWARE DEVELOPMENT HISTORY

---

The main milestones in the development of WinNeural are outlined below:

Description	WinNeural Version Number
Creation of WinNeural software, initial implementation (using <code>VectorN</code> data structures, see <i>Section 4.2.2. IAC Implementation</i> ) Menus Dialog boxes Scheme of architecture	0.00 – 0.04
Construction of network library Major bug fixing	0.05
File handling added Interactive Activation and Competition type completed Implementation of new data structure, <code>NEURON</code> , to reduce multiple uses of <code>VectorN</code> (see <i>Section 4.3.2. CS Implementation</i> ) First discrepancy found for “spontaneous generalisation” between WinNeural and published PDP results (see <i>Section 4.10. Discussion of Deviations</i> )	0.06
Initial Constraint Satisfaction network, Schema Model, construction File and display handling extended	0.07
File and display handling extended (plus bug fixing) Implementation of new data types, <code>STRINGVAR</code> and <code>DOUBLEVAR</code> , to replace multiple uses of <code>VectorN</code> (see <i>Section 4.3.2. CS Implementation</i> ) Bug fixing	0.08
Change of compiler from QuickC to Visual C++ Incorporation of new Windows File dialog boxes (from Windows file <code>comdlg32.dll</code> )	0.09
File handling library created	0.10



## Appendix B: Detailed Software Development History

Memory handling rewritten and optimised	
File handling improved	0.11
Constraint Satisfaction, Schema Model, completed	
Constraint Satisfaction, Boltzmann Model, completed	0.12
Further discrepancies found between WinNeural and PDP results (see <i>Section 4.10. Discussion of Deviations</i> )	
Constraint Satisfaction, Harmony Model, completed	0.13
Pattern Associator network construction	0.14
Pattern Associator completed	0.15
Back Propagation network construction	0.16
Bug fixing	

Table B.1 – Milestones in the Development of WinNeural

Below are some example screenshots of WinNeural during its development.

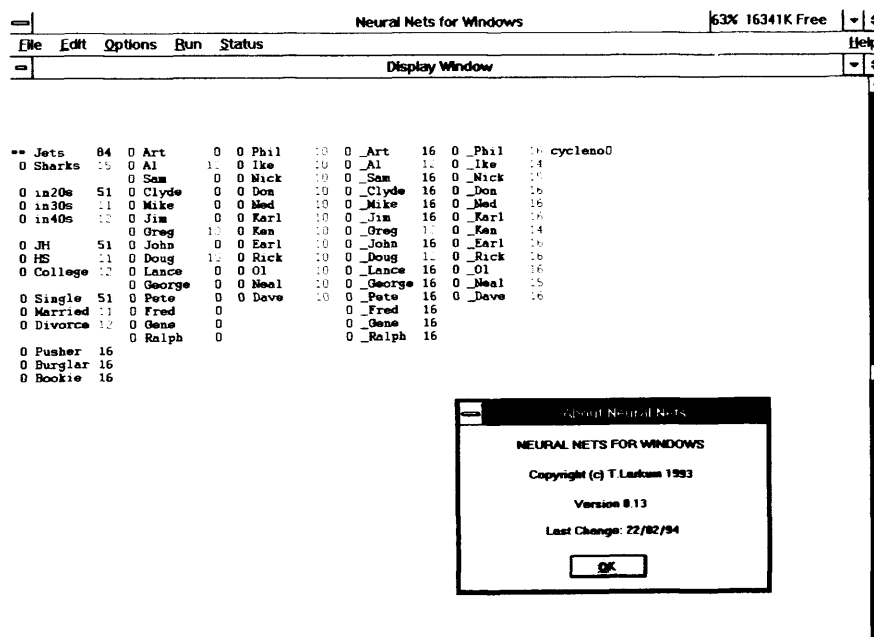


Figure B.1 – WinNeural Version 0.13 running on Windows 3.1

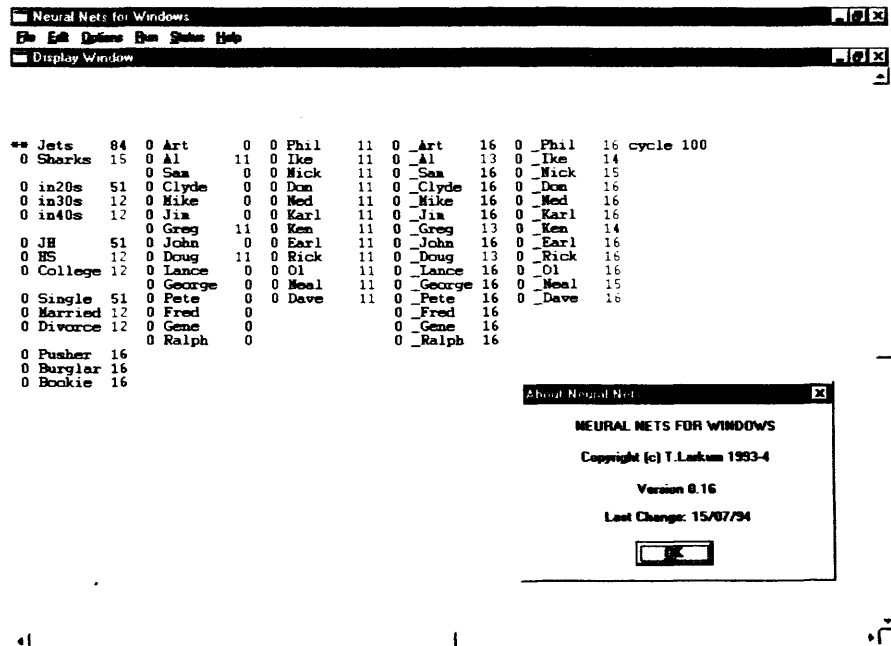


Figure B.2 – WinNeural Version 0.16 running on Windows 95

By this stage in the development of WinNeural, technology and the general software environment had moved on such that it was decided to restart the development of the software. This redevelopment had the following main elements:

- where practical and appropriate move from the C language to C++
- development of a new user interface using new facilities in the Visual C++ environment, and written entirely in C++
- retention of neural network library files in C
- change of name to “Neural Networks for Windows” (NNW).

The main milestones in the development of NNW are outlined below:

Description	NNW Version Number
Creation of NNW interface	1.0
Conversion from 16bit to 32bit handling	“NNW32” 1.0
Improvements to user interface, for example: <ul style="list-style-type: none"> <li>• better display of network definitions</li> <li>• addition of scroll bars</li> <li>• appropriate menu greying</li> </ul>	1.1
Back Propagation network type completed	1.11
Improvements to user interface to make testing easier and faster, final bug fixing, modifications to menu structure	1.2

## Appendix B: Detailed Software Development History

Table B.2 – Milestones in the Development of Neural Networks for Windows

Below are some example screenshots of NNW during its development:

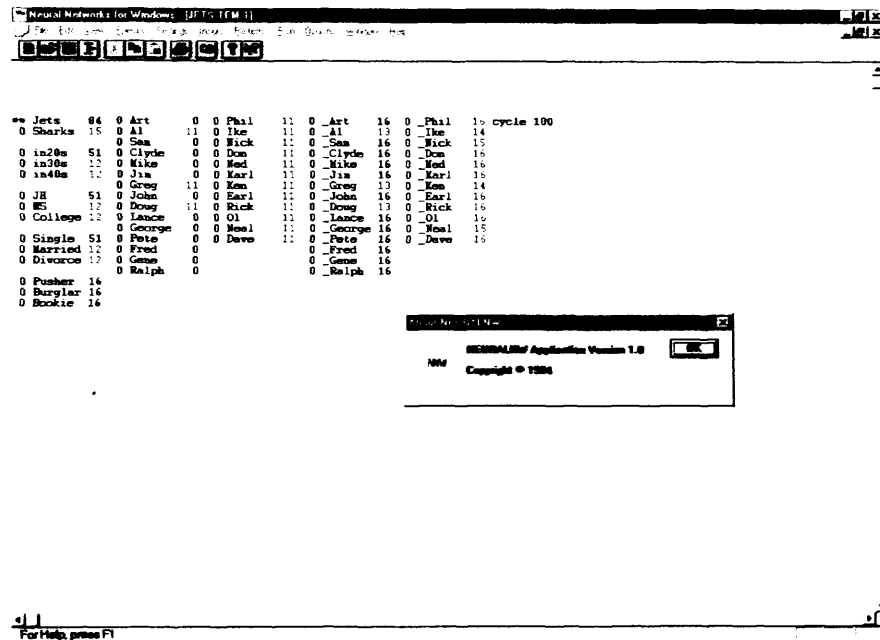


Figure B.3 – NNW Version 1.0 running on Windows 98

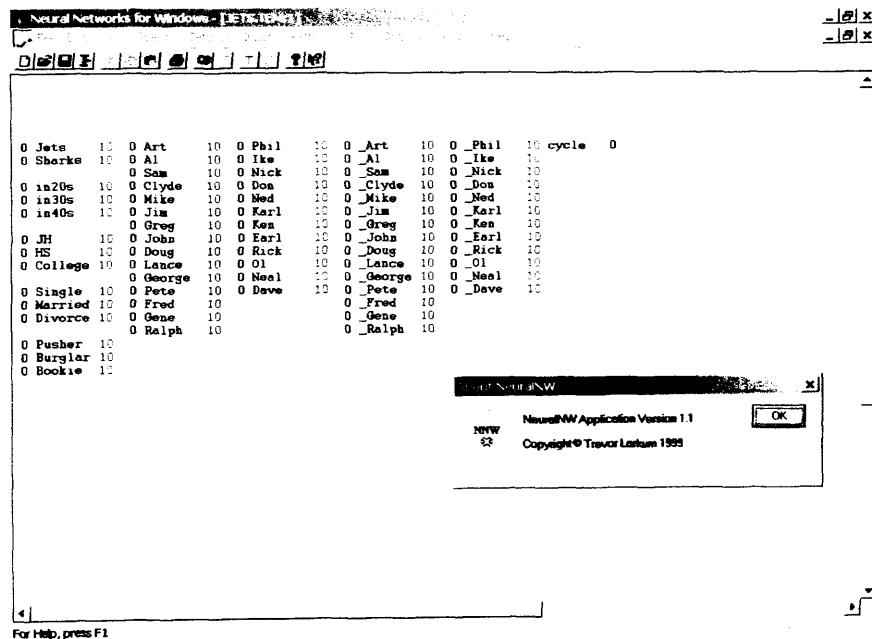


Figure B.4 – NNW Version 1.1 running on Windows 2000

---

## APPENDIX C: NEURAL NETWORK SOFTWARE FEATURES

---

### C.1. File Menu

The File menu has the following items:

New	Creates a new network – brings up the New Neural Net dialog box (see below)		
Close	Closes down the current network	New	Ctrl+N
Import PDP File...	Brings up the Windows standard File Open dialog box so the user can select a PDP compatible file	Open...	Ctrl+O
Print...	Brings up the Windows standard Print dialog box so the user can select printing options	Close	
Print Preview	Changes to preview mode to show how the screen display will appear when printed out	Save	Ctrl+S
Print Setup	Brings up the standard Windows Print Setup dialog box so the user can change printing options (e.g. portrait/landscape orientation)	Save As...	
MRU list	This is a list of Most Recently Used (MRU) files that can be loaded by simply selecting them	Import PDP File...	Ctrl+I
Exit	Closes the current network and leaves the program	Print...	Ctrl+P
		Print Preview	
		Print Setup...	
		1 XOR.TEM	
		2 8X8.TEM	
		3 XOR.TEM	
		4 JETS.TEM	
		Exit	

The File Menu also has options for Open..., Save and Save As... which operate on neural net files specific to NNW (extension .nnw); however PDP compatible files were used throughout the work described in this thesis.

Selecting File:New brings up the New Neural Net dialog box shown in Figure C.1; this is used to select the main type of the network (subtypes can then be set from the options menu or via character strings in the network definition files as they are read in).

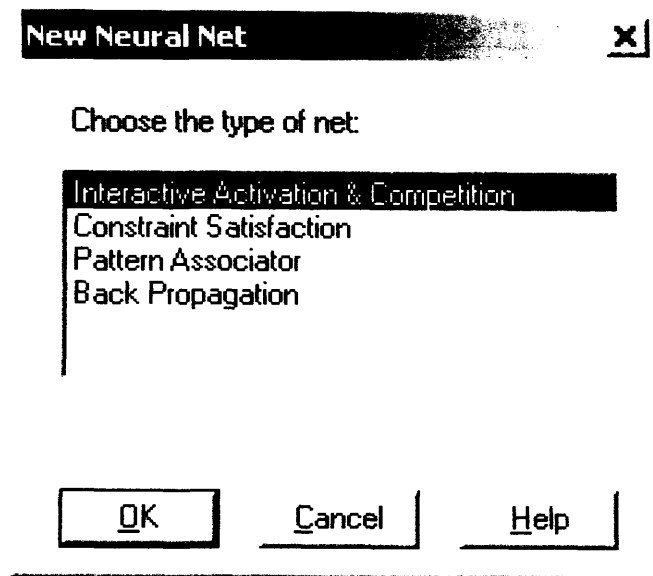


Figure C.1 – File New / New Neural Net dialog box

## C.2. Edit Menu

The Edit menu has the following items:

Input Values...	Brings up the Set Input dialog box (see below)	Input Values...
Reset Inputs	Resets all inputs to their initial state	Reset Inputs
Neuron Weights...	Brings up the Set Weight Value dialog box (see below)	Neuron Weights...

The Set Input dialog box (see Figure C.2) allows the user to change an input by typing in the unit name and the new activation value.

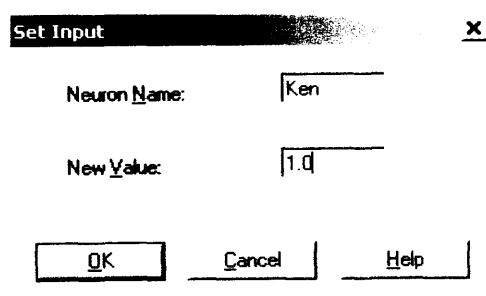


Figure C.2 – Edit Input Values / Set Input dialog box

The Set Weight Value dialog box (see Figure C.3) allows the user to change a weight between units by entering the two unit names and the new weight value.

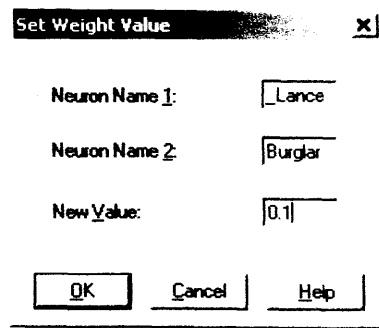


Figure C.3 – Edit Neuron Weights / Set Weight Value dialog box

### C.3. View Menu

The View menu has the following items:

Toolbar	Toggles display of the Toolbar on or off	
Status Bar	Toggles display of the Status Bar along the bottom of the program on or off	✓ Toolbar
Contents shows Weights	If checked, the Contents window will display weights (instead of patterns)	✓ Status Bar
Contents shows Patterns	If checked, the Contents window will display patterns (instead of weights) – only available if at least one pattern has been loaded or created	✓ Contents shows Weights Contents shows Patterns
Run Results	Brings up the Run Results dialog box (see below)	Run Results

The Run Results dialog box (see Figure C.4) displays the current values of the network cycle number, goodness value and total sum of squares (*tss*).

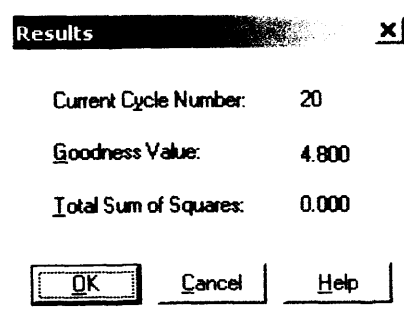


Figure C.4 – Run Results dialog box

## C.4. Settings Menu

The Settings menu has the following items:

System...	Brings up the System Settings dialog box (see below)
Display...	Brings up the Display Settings dialog box (see below)
Strengths...	Brings up the Strength Parameters dialog box (see below)
Activations...	Brings up the Activation Parameters dialog box (see below)
Rates...	Brings up the Rate and Other Parameters dialog box (see below)

System...

Display...

Strengths...

Activations...

Rates...

The System Settings dialog box (Figure C.5) allows the user to set the number of cycles per run, the random number seed, the number of epochs required (see *Section 4.6.1. PA Theory*), and the error criterion value (see *Section 4.6.3. PA Implementation*). Note that the random number seed will be changed automatically whenever `Run:Restart` (see below) is selected (see *Section 4.3.3. Schema Model Theory*).

The modes for training and for testing can be set independently to be either multiple stepping (run continuously until the system reaches the required number of cycles or epochs respectively) or single stepping, when the program halts after each step (cycle or epoch) until the user continues.

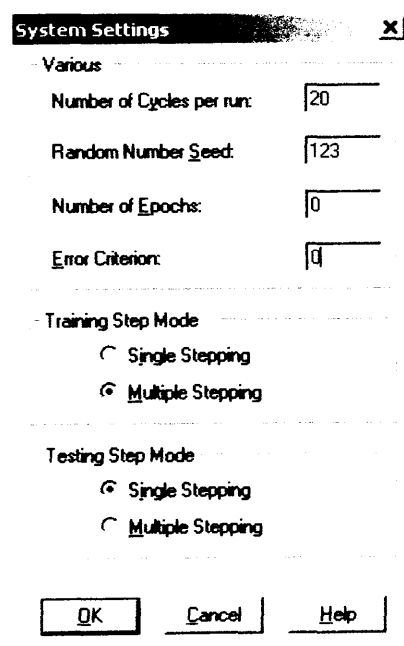


Figure C.5 – System Settings dialog box



The Display Setting dialog box (see Figure C.6) sets whether the screen is updated after every cycle or only on completion of all cycles.

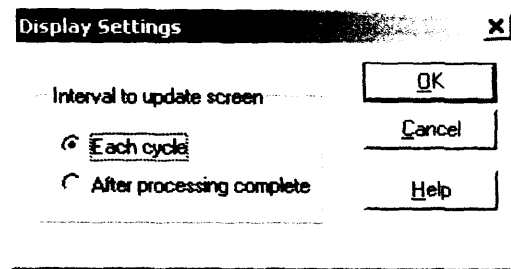


Figure C.6 – Display Settings dialog box

The Strength Parameters dialog box (see Figure C.7) allows the user to set the required values for the excitatory input ( $\alpha$ ), inhibitory input ( $\gamma$ ) and external input strength ( $estr$ ) – see Section 4.2.1. *IAC Theory*. It also allows the user to set the internal input strength ( $istr$ ) – see Section 4.3.3. *Schema Model Theory* – and the harmony constant ( $\kappa$ ) – see Section 4.5.1. *Harmony Model Theory* (it also allows the user to set the decay strength,  $\beta$ , but this is not applicable to the network types considered in this work).

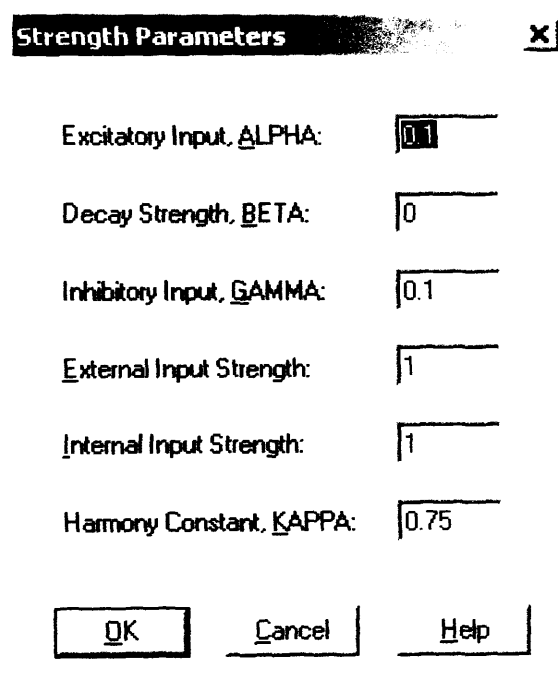
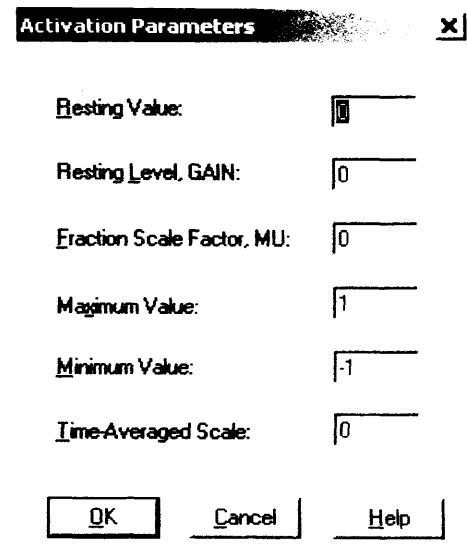


Figure C.7 – Strength Parameters dialog box

The Activation Parameters dialog box (see Figure C.8) allows the user to set the required values for the system resting value ( $rest$ ), maximum value ( $max$ ) and minimum value ( $min$ ) – see Section 4.2.1. *IAC Theory* (it also allows the user to set the resting level, fraction scale factor, and time-averaged scale but these are not applicable to the network types considered in this work).



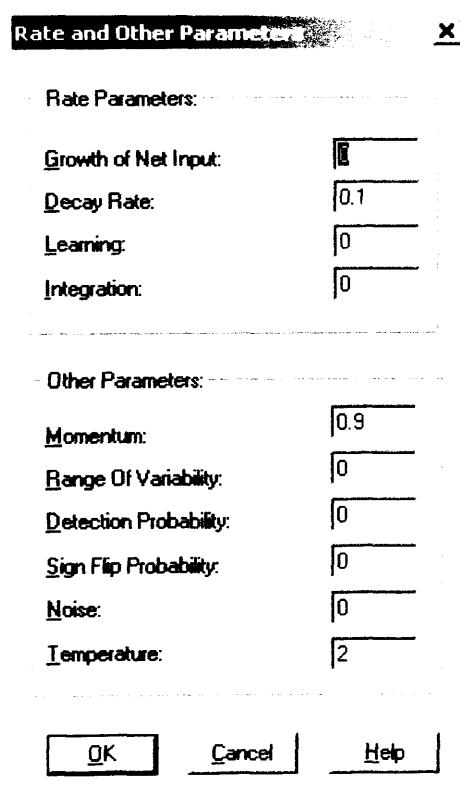
The 'Activation Parameters' dialog box contains the following fields and controls:

Parameter	Value
Resting Value:	0
Resting Level, GAIN:	0
Fraction Scale Factor, MU:	0
Maximum Value:	1
Minimum Value:	-1
Time-Averaged Scale:	0

Buttons: OK, Cancel, Help

Figure C.8 – Activation Parameters dialog box

The Rate and Other Parameters dialog box (see Figure C.9) allows the user to set the required values for the decay rate – see *Section 4.2.1. IAC Theory*, learning rate, noise and temperature - see *Section 4.6.3. PA Implementation* (it also allows the user to set the growth of net input, integration, momentum, range of variability, detection probability and sign flip probability parameters but these are not applicable to the network types considered in this work).



The 'Rate and Other Parameters' dialog box is divided into two sections:

**Rate Parameters:**

Parameter	Value
Growth of Net Input:	0
Decay Rate:	0.1
Learning:	0
Integration:	0

**Other Parameters:**

Parameter	Value
Momentum:	0.9
Range Of Variability:	0
Detection Probability:	0
Sign Flip Probability:	0
Noise:	0
Temperature:	2

Buttons: OK, Cancel, Help

Figure C.9 – Rate and Other Parameters dialog box

## C.5. Patterns Menu

The Patterns menu has the following items:

Select Input...	Brings up the Select Pattern dialog box (see below)	Select Input...
Select Target...	Brings up the Select Pattern dialog box (see below)	Select Target...
Select Pattern...	Brings up the Select Pattern dialog box (see below)	Select Pattern...
Enter Pattern...	Brings up the Enter Pattern dialog box (see below)	Enter Pattern...

The Select Pattern dialog box (Figure C.10) allows the user to choose a pattern from the list of patterns available (i.e. those already read in from file, or created by the user via the Enter Pattern dialog box described below). It can be used in three ways:

1. To select just the input pattern (when the text reads "Choose the input pattern"). In this case the first half of the pattern is used as the input pattern.
2. To select just the target pattern (when the text reads "Choose the target pattern"). In this case the second half of the pattern is used as the target pattern.
3. To select a complete pattern (when the text reads "Choose the test pattern"). In this case the first half of the pattern is used as the input pattern and the second half is used as the target pattern.

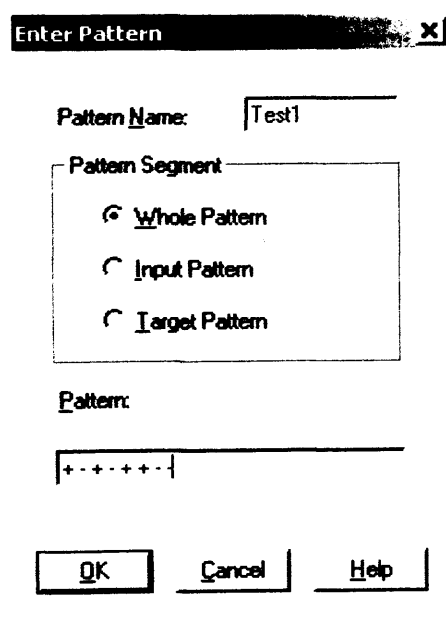


Figure C.10 – Enter Pattern dialog box

The Enter Pattern dialog box (see Figure C.10) allows the user to create a new pattern by entering it directly (as a series of numbers, or + for 1.0 and – for –1.0), giving it a name, and specifying whether it represents a whole pattern (i.e. both input and target segments) or just an input or target pattern.

## C.6. Run Menu

The Run menu has the following items:

Go / Train	Starts the system running (i.e. settling if IAC or CS, training if PA or BP)
Step Through	Runs just the next step (only available if the system has been started with Go/Train, and if the Training Step Mode is set to Single Stepping – see <i>Section C.4. Settings Menu</i> )
Reset	Returns the network to its starting state (see <i>Section 4.2.1. IAC Theory</i> )
New Start	Returns the network to its starting state after generating a new random number seed (see <i>Section 4.3.3. Schema Model Theory</i> ). This is equivalent to entering a new seed directly, see <i>Section C.4. Settings Menu</i> above, and then using Reset
Test	Tests the network against the current pattern (for selection see <i>Section C.5. Patterns Menu</i> )
Test All	Tests the network against all available patterns
Step Through	Tests just the next epoch (only available if the system has been started with Test or Test All, and if the Testing Step Mode is set to Single Stepping – see <i>Section C.4. Settings Menu</i> )
Stop	Stops the current run or test (only available if the system has been started with Go/Train, Test or Test All and the appropriate Step Mode is set to Single Stepping)

Go / Train	F5
Step Through	
Reset	F7
New Start	F8

Test
Test All
Step Through
Stop

## C.7. Options Menu

The Options menu has the following items:

## Appendix C: Neural Network Software Features

PDP Group Update	If checked, the PDP group update is being used (otherwise Grossberg update)	
Grossberg Update	If checked, the Grossberg Update is being used (otherwise PDP Group update)	
Schema Model	If checked, the network is set to use the Schema model	✓ PDP Group Update Grossberg Update
Boltzmann Model	If checked, the network is set to act like a Boltzmann machine	✓ Schema Model Boltzmann model
Harmony Model	If checked, the network is set to act like a Harmonium	Harmony Model
Hebb Learning	If checked, Hebb learning is on	Hebb Learning
Delta Learning	If checked, Delta rule learning is on	Delta Learning
Learning is on	If checked, Learning is on	✓ Learning is on
Follow is on	If checked, Follow mode is on	✓ Follow is on
Training is Permuted	If checked, training patterns are permuted (otherwise sequential)	Training is Permuted
Learn each Pattern	If checked, the network will update itself after each pattern (otherwise after each epoch)	Learn each Pattern ✓ Learn each Epoch
Learn each Epoch	If checked, the network will update itself after each epoch (otherwise after each pattern)	Clamping On
Clamping On	If checked, clamping mode is on	

For details of the differences between the PDP Group Update and the Grossberg Update rules, see *Section 4.2.2. IAC Implementation*; for the differences between the Schema, Boltzmann and Harmony models of constraint satisfaction, see *Sections 4.3.3. Schema Model Theory, 4.4.1. Boltzmann Machine Theory and 4.5.1. Harmony Model Theory*.

For the Hebb and Delta rules and Sequential and Permuted Training see *Section 4.6.1. PA Theory*; for Learning Mode see *Section 4.6.3. PA Implementation*. For Follow Mode see *Section 4.8.2. Solving the XOR Problem*. For Clamping Mode see *Section 4.3.1. CS Theory*.

### C.8. Window Menu

The Window menu has the following items:

Cascade	Arrange non-minimised windows in a cascaded manner (i.e. overlapping)	Cascade
Tile	Arrange non-minimised windows in a tiled manner (i.e. side by side)	Tile
Arrange Icons	Arranged minimised windows along the bottom of the application window	Arrange Icons
[Window names]	Select a window name to give it focus (i.e. open it if closed, and bring it to the front). The first window is usually the network display window, and the second one is the network contents (i.e. internal weights) window	<div> <div>✓ 1 CUBE.TEM:1</div> <div>2 CUBE.TEM:2</div> </div>

## C.9. Help Menu

The Help menu has the following items:

Index	Intended to bring up the NNW help file (not implemented)	Index
About NNW...	Brings up the About NNW dialog box (see below)	About NNW...

The About NNW dialog box (Figure C.11) provides information regarding version, copyright and date of the NNW application.

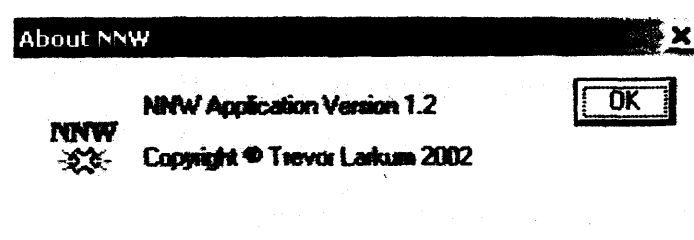


Figure C.11 – Help / About NNW dialog box

## C.10. Toolbar



The NNW toolbar has buttons for the following commands (identical to the equivalent menu options), from left to right: File:New, File:Open, File:Save, File:Import

PDP File, File:Print, Run:Go/Train, Run:Train Step Through (shown greyed out), Run:Test All, Run:Test All Step Through (shown greyed out), Help>About and Help:Context Help.

### **C.11. PDP Network Definition Files**

NNW is capable of reading in PDP-compatible network definition files; these have the following forms (in brief):

- |                                 |   |
|---------------------------------|---|
| <b>Strengths File ('.str'):</b> | This defines the values of the main parameters required (otherwise defaults are used), such as <i>min</i> , <i>max</i> , <i>decay</i> , <i>estr</i> , plus an annealing schedule if appropriate (of the form <i>starting temperature / number of cycles / target temperature</i> ), plus names for the units in the network. It may also give the names of Network and Weights files to be read in. |
| <b>Template File ('.tem'):</b>  | This provides a template for the layout of parameters in the display window, and how they are to be displayed (scale factor, etc.).   |
| <b>Network File ('.net'):</b>   | This defines the number of inputs, outputs and total units, and the number of units to update per cycle, and the values to be used for the various weights between units (if a separate Weights files has not been specified).  |
| <b>Weights File ('.wts'):</b>   | This gives numerical values for all weights and biases in the network.  |
| <b>Pattern File ('.pat'):</b>   | This contains a list of pattern names and their values.   |
| <b>Look File ('.loo'):</b>      | This defines the display window position at which variables are to be output.   |

These file types (which contain many complex elements) will not be described further here – a full specification is given in [McClelland 1988] and examples are included in Chapter 6, and in Appendix D, and on the attached CD-ROM.

## ROV.tem File

## Manual ROV Docking Test

Cycleno	\$
updateno	\$
cuname	\$
goodness	\$
temperature	\$

[illegible]

## ROV.100 File

A 10x10 grid of dots. The numbers 0 through 9 are placed at the top of the grid, and the numbers 10 and 11 are placed at the bottom. The grid is composed of 10 rows and 10 columns of dots. The numbers are placed at the following intersections (row, column):

- 0: (0, 3)
- 1: (0, 4)
- 2: (0, 5)
- 3: (0, 6)
- 4: (0, 7)
- 5: (0, 8)
- 6: (0, 9)
- 7: (0, 10)
- 8: (0, 11)
- 9: (0, 12)
- 10: (9, 3)
- 11: (9, 10)



## ROV.net File

[illegible]

### ROV2.str File (Schema Model)

```
set dlevel 1
get network rov2.net
get weights rov2.wts
set mode clamp 1
set param estr 1.0
set param istr 0.2
set ncyc 50
get unames RV+100 RV+10+1 RV+1+1+1 RV-100 RV-10+1 RV-1+1+1 RV+10-1
RV+1+1-1 RV-10-1 RV-1+1-1 AL+100 AL+10+1 AL+1+1+1 AL-100 AL-10+1
AL-1+1+1 AL+10-1 AL+1+1-1 AL-10-1 AL-1+1-1
```

## ROV2.tem File

```
define: layout
```

## Manual ROV Docking Test - Offset Port Foot

=====

\$ Possible Locations

Cycleno	\$
updateno	\$
cuname	\$
goodness	\$
temperature	\$

## Sticky Foot

end

rovdock	look	1	\$ 0	activation	1 10 1	rov2.loo
cycleno	variable	1	\$ 1	cycleno	7 1	
updateno	variable	1	\$ 2	updateno	7 1	
uname	variable	1	\$ 3	cuname	-8 1.0	
goodness	floatvar	1	\$ 4	goodness	7 1.0	
temperature	floatvar	1	\$ 5	temperature	7 1.0	
weight	matrix	5	\$ 0	weight	h 4 10.0 0 16 0 16	
weight	vector	5	\$ 2	uname	v 6 1 0 16	
weight	vector	5	\$ 5	uname	h 4 1 0 16	

## ROV2.lco File

9 74

A 20x20 dot grid. The top edge is labeled with numbers 0 through 19. The left edge is labeled with numbers 0 through 19. The grid consists of 20 columns and 20 rows of dots.

definitions:  
 1. 20

## DV2.wts File (Boltzmann Machine)

265

---

## APPENDIX E: DOCKING LIBRARY MAIN SOURCE CODE

---

### Docking.h

```

/*****
/* FILE DOCKING.H */
/*-----*/
/* File Name : \Include\Docking.h */
/* Class Name : CDocking */
/* Purpose : Class header file */
/* Author : T.Larkum */
/* Written on : 08/12/95 */
/*-----*/
/* Copyright (c) Technical Software Consultants Ltd. 1995. */
/*-----*/
/* HISTORY */
/*-----*/
/* Changed by Date Reason */
/*-----*/
/*-----*/
/*****/

#ifndef _INC_DOCK
#define _INC_DOCK

/*-----*/
/* INCLUDE FILES */
/*-----*/

#include <canddate.h>
#include <vector.h>

/*-----*/
/* DEFINITIONS */
/*-----*/

#define MIN_REACH_DISTANCE 0.5
#define MAX_REACH_DISTANCE 2.0
#define MIN_ATTACH_DISTANCE 1.1
#define MAX_ATTACH_DISTANCE 1.6
#define FOOT_STICKING_RANGE 0.5 // Foot sticking point can be
                                // +- this distance.

/*-----*/
/* CLASS DEFINITION */
/*-----*/

class CDocking : public CWnd
{
public:
    CDocking( );
    ~CDocking( );

protected:
    int WriteLayoutFile();
    int WriteWeightsFile();
    int WriteStrengthsFile();
    int WriteTemplateFile();
    int WriteNetworkFile();
    void PruneList();
    DECLARE_SERIAL( CDocking )

```

## Appendix E: Docking Library Main Source Code

```
// Attributes
protected:
    CObList m_posList;
    CVector m_vWeldCofG;
    VECTOR m_vecBaseOrient, m_vecBasePos;
    double m_dGridSpacing, m_dxStart, m_dxEnd, m_dyStart, m_dyEnd;
    double m_dZStart, m_dZEnd, m_dExtendStep, m_dRotateStep;

// Operations
public:
    BOOL AttachmentLegsAvailable();
    void ShowResults();
    int Initialize();
    int CheckForCollisions();
    int CheckForROVCollisions();
    int CheckForReach();
    int CheckStickyFeetReach();
    int CheckStickyFeetPositions();
    int WriteDefinitionFiles();
    int CloseDown();
    void Draw();
    void ChangeSettings();
    void ToggleRedraws();
    virtual void Serialize( CArchive& ar );
    // Following copied from scribble (ensures type safe handling of CObList):
    CCandidate* NewCandidate( CVector* pvTemp );
    CCandidate* GetNextCandidate( POSITION &pos );
    POSITION GetFirstCandidatePos();
    void DeleteCandidateList();

// Helper functions
protected:
    double m_dAcceptanceLimit;
    BOOL m_bShowRedraws;
    BOOL m_bDrawCandidates;
    BOOL CollisionSituation(BOOL bReport);
    void WorldCoords(LPVECT pVec, BOOL bForward = TRUE);
    void WorldCoords(CVector* pVec, BOOL bForward = TRUE);

// Overrides
/* Having the message map allows basic use of ClassWizard to create command
// functions, but they still don't get called
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CDocking)
//}}AFX_VIRTUAL

public:
// Implementation
//{{AFX_MSG(CDocking)
afx_msg void OnDockSettings();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
*/
};

#endif // _INC_DOCK
```

## Docking.cpp

```

/*****
/* FILE DOCKING.CPP */
/*-----*/
/* File Name : \Docking\Docking.cpp */
/* Class Name : CDocking */
/* Purpose : Implementation of the CDocking class */
/* Author : T.Larkum */
/* Written on : 08/12/95 */
/*-----*/
/* Copyright (c) Technical Software Consultants Ltd. 1995. */
/*-----*/
/* HISTORY */
/*-----*/
/* Changed by Date Reason */
/*-----*/
/*-----*/
/*****
/* INCLUDE FILES */
/*-----*/

#include <stdafx.h>
#include <math.h> // for fabs

#include <docking.h>
#include <colision.h>
#include <gui.h>
#include <plot.h>
#include <path.h>
#include <workpce.h>
#include <stringex.h>
#include <guimenu.h>
#include <jobs.h>
#include <toolskid.h>
#include <manip.h>
#include <task.h>
#include <message.h> // for kinematic access msg
#include <coordsys.h> // Just so we can set the extend/rotate values directly
#include <docksetdlg.h>
#include <dockresults.h>
#include <genmaths.h>

/*-----*/
/* DEFINITIONS */
/*-----*/

#define GRID_SPACING 0.25 // Size of candidate position grid in meters
#define GRID_SIZE 1.00
#define EXTEND_STEP 0.25
#define ROTATE_STEP (PI/36.0); // every 5 degrees
#define EXCLUSIVE_VALUE -1.00
#define NO_CONNECTION 0.00
#define DEFAULT_BIAS 0.10
#define FIELD_WIDTH 6
#define PRECISION_DP 2
#define LAYOUT_ROWS 9
#define LAYOUT_COLUMNS 74
#define LAYOUT_STARTPOS 15
#define LAYOUT_EXTRAPOS 6 // Extra variables (access, sticky feet) n rows below nodes

#define EXTRA_CRITERIA 2

/*-----*/
/* MFC MACROS */
/*-----*/

IMPLEMENT_SERIAL( CDocking, CWnd, 1 );

/* Having the message map allows basic use of ClassWizard to create command
// functions, but they still don't get called!
BEGIN_MESSAGE_MAP(CDocking, CWnd)
//{{AFX_MSG_MAP(CDocking)
ON_COMMAND(IDM_DOCK_SETTINGS, OnDockSettings)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
*/

/*-----*/
/* CLASS IMPLEMENTATION */
/*-----*/

CDocking::CDocking()
{
    m_bDrawCandidates = TRUE;

```

## Appendix E: Docking Library Main Source Code

```
m_bShowRedraws = FALSE;
m_dxStart = -GRID_SIZE;
m_dyStart = -GRID_SIZE;
m_dzStart = -GRID_SIZE;
m_dxEnd = 0.0;//GRID_SIZE;
m_dyEnd = GRID_SIZE;
m_dzEnd = GRID_SIZE;
m_dGridSpacing = GRID_SPACING;
m_dExtendStep = EXTEND_STEP;
m_dRotateStep = ROTATE_STEP;
m_dAcceptanceLimit = 0.5;
}

CDocking::~CDocking()
{
}

CCandidate* CDocking::NewCandidate( CVector* pvTemp )
{
    CCandidate* pCandidateItem = new CCandidate(pvTemp);
    m_posList.AddTail( pCandidateItem );

    return pCandidateItem;
}

CCandidate* CDocking::GetNextCandidate( POSITION &pos )
{
    return (CCandidate*)m_posList.GetNext( pos );
}

POSITION CDocking::GetFirstCandidatePos()
{
    return m_posList.GetHeadPosition();
}

int CDocking::Initialize( )
{
    double dx, dy, dz;
    CVector vCandidate;

    VECTOR vecWeldCofG;
    WPGetWorkpieceLocation(WORK_PIECE, NULL, &vecWeldCofG);
    GetVector( &vecWeldCofG, &dx, &dy, &dz );
    m_vWeldCofG.SetAll( dx, dy, dz );

    WPGetBaseLocation(&m_vecBaseOrient, &m_vecBasePos);
    // If we already have a list, delete it:
    DeleteCandidateList();

    // Create a set of candidate objects, each representing a point on a cubic grid:
    for( dx = m_dxStart; dx <= m_dxEnd; dx += m_dGridSpacing )
        for( dy = m_dyStart; dy <= m_dyEnd; dy += m_dGridSpacing )
            for( dz = m_dzStart; dz <= m_dzEnd; dz += m_dGridSpacing )
            {
                vCandidate.SetAll( dx, dy, dz);
                NewCandidate(&vCandidate);
            }
    return m_posList.GetCount();
}

int CDocking::CheckForCollisions()
{
    CVector *pvTemp;
    double dx, dy, dz;
    VECTOR vecTemp;
    BOOL bCollision;

    POSITION pos = GetFirstCandidatePos();
    while( pos != NULL )
    {
        CCandidate* pCandidate = GetNextCandidate( pos );

        // check it:
        if( pCandidate->GetIsPossible() )
        {
            pvTemp = pCandidate->GetPos();
            pvTemp->GetAll( &dx, &dy, &dz );
            MakeVector( (LPVECT)&vecTemp, dx, dy, dz );
            WorldCoords( &vecTemp );
            // Check for collision with anything (name is misleading):
            bCollision = ColCheckPointIsInWorkpiece( (LPVECT)&vecTemp );
            if( bCollision )
                pCandidate->SetIsPossible(FALSE);
        }
    }
    PruneList();
    return m_posList.GetCount();
}
```

## Appendix E: Docking Library Main Source Code

```

int CDocking::CheckForROVCollisions()
{
    CVector *pvTemp;
    double dX, dY, dZ;
    VECTOR vecOrient, vecCand, vecOldPos, vecNewPos;

    WPGetWorkpieceLocation( WORK_PIECE, &vecOrient, &vecOldPos );
    m_bDrawCandidates = FALSE;
    JobCheckForCollision( FALSE );

    POSITION pos = GetFirstCandidatePos();
    while( pos != NULL )
    {
        CCandidate* pCandidate = GetNextCandidate( pos );

        // check it:
        if( pCandidate->GetIsPossible() )
        {
            pvTemp = pCandidate->GetPos();
            pvTemp->GetAll( &dX, &dY, &dZ );
            MakeVector( (LPVECT)&vecCand, dX, dY, dZ );

            CopyVector(&vecOldPos, &vecNewPos);
            // Set the workpiece location to be relative to candidate position:
            WorldCoords(&vecNewPos, FALSE);
            VectorSubtract(&vecNewPos, &vecCand, &vecNewPos);
            WPSetWorkpieceLocation( WORK_PIECE, &vecOrient, &vecNewPos );
            MenuOnEditLocation(m_bShowRedraws, FALSE);

            // Check for collisions against ROV:
            if (CollisionSituation(m_bShowRedraws))
                pCandidate->SetIsPossible(FALSE);
            // Check for collisions against manip:
            else if (ColCheckLinksForContact(FALSE))
                pCandidate->SetIsPossible(FALSE);
        }
    }
    PruneList();

    // Reset to start position:
    WPSetWorkpieceLocation( WORK_PIECE, &vecOrient, &vecOldPos );
    MenuOnEditLocation(m_bShowRedraws, FALSE);
    m_bDrawCandidates = TRUE;

    return m_posList.GetCount();
}

// This is original version, later replaced by CheckStickyFeetPositions
int CDocking::CheckStickyFeetReach()
{
    CVector *pvTemp;
    double dDistance, dRoll, dPitch, dYaw, dX, dY, dZ, dChordRadius, dBraceRadius;
    TUBULAR tbChord, tbBrace;
    VECTOR vecChordPos, vecChordOrient, vecBraceOffset, vecBraceOrient;
    VECTOR vecZAxis, vecChordAxis, vecBraceAxis, vecBracePos, vecZero, vecTemp;
    MATRIX mChordOrient, mBraceOrient;

    // Get the locations and dimensions of the chord and current brace:
    WPGetTubularData( WORK_PIECE, (LPTUBE)&tbChord );
    dChordRadius = tbChord.dMainDiameter/2.0;
    WPGetWorkpieceLocation( WORK_PIECE, (LPVECT)&vecChordOrient, (LPVECT)&vecChordPos );
    int nBrace = PathGetNumber( CURRENT_COMP );
    WPGetTubularData( nBrace, (LPTUBE)&tbBrace );
    dBraceRadius = tbBrace.dMainDiameter/2.0;
    WPGetWorkpieceLocation( nBrace, (LPVECT)&vecBraceOrient, (LPVECT)&vecBraceOffset );

    // Set up vectors to represent their directions, first get orientation matrices:
    MakeVector( (LPVECT)&vecZAxis, 0.0, 0.0, 1.0 );
    MakeVector( (LPVECT)&vecZero, 0.0, 0.0, 0.0 );
    GetVector( (LPVECT)&vecChordOrient, &dRoll, &dPitch, &dYaw );
    VectorGetRotationMatrix( dRoll, dPitch, dYaw, (LPMAT)&mChordOrient );
    GetVector( (LPVECT)&vecBraceOrient, &dRoll, &dPitch, &dYaw );
    VectorGetRotationMatrix( dRoll, dPitch, dYaw, (LPMAT)&mBraceOrient );

    // ...then orientate unit vector to component direction:
    MatrixVectorMultiply( (LPMAT)&mChordOrient, (LPVECT)&vecZAxis, (LPVECT)&vecChordAxis );
    WPTransformVector( (LPMAT)&mBraceOrient, (LPMAT)&mChordOrient, (LPVECT)&vecBraceOffset,
        (LPVECT)&vecZAxis, (LPVECT)&vecBraceAxis, ORIENT_COMP_WP );

    // now make vecBraceOffset the true world position:
    WPTransformVector( (LPMAT)&mBraceOrient, (LPMAT)&mChordOrient, (LPVECT)&vecBraceOffset,
        (LPVECT)&vecZero, (LPVECT)&vecBracePos, ORIENT_AND_TRANSLATE );

    /*-----*/
    VectorScalarMultiply((LPVECT)&vecBraceAxis, dChordRadius, (LPVECT)&vecTemp);
    VectorAdd((LPVECT)&vecBracePos, (LPVECT)&vecTemp, (LPVECT)&vecBracePos);
    /*-----*/

    POSITION pos = GetFirstCandidatePos();
    while( pos != NULL )

```



## Appendix E: Docking Library Main Source Code

```

{
    CCandidate* pCandidate = GetNextCandidate( pos );

    // check it:
    if( pCandidate->GetIsPossible() )
    {
        pvTemp = pCandidate->GetPos();
        pvTemp->GetAll( &dX, &dY, &dZ );
        MakeVector( (LPVECT)&vecTemp, dX, dY, dZ );
        WorldCoords(&vecTemp);

        // Assume we can't reach either initially:
        pCandidate->SetIsPossible(FALSE);

        // Is the position close enough to the chord:
        dDistance = VectorDistanceFromLine( (LPVECT)&vecTemp,
            (LPVECT)&vecChordPos, (LPVECT)&vecChordAxis );
        // Its valid to be along or opposite to chord direction
        dDistance = fabs(dDistance);
        dDistance -= dChordRadius;
        if( dDistance > MIN_ATTACH_DISTANCE && dDistance < MAX_ATTACH_DISTANCE )
            pCandidate->SetIsPossible(TRUE);
        // or the brace
        dDistance = VectorDistanceFromLine( (LPVECT)&vecTemp,
            (LPVECT)&vecBracePos, (LPVECT)&vecBraceAxis );
        if (dDistance < 0.0)
        {
            VECTOR vecDist;
            // vecTemp is on the -ve vecBraceAxis side of vecBracePos, so
            // is actually the nearest point we could stick, +-
            VECTORSubtract( (LPVECT)&vecTemp, (LPVECT)&vecBracePos, (LPVECT)&vecDist);
            // dIntersection would be negative on this side of vecBracePos
            double dIntersection = -1.0 *
            DotProduct( (LPVECT)&vecBraceAxis, (LPVECT)&vecDist);
            if (dIntersection < FOOT_STICKING_RANGE)
                // Let it go, it's close enough
                dDistance = -dDistance;
        }

        dDistance -= dBraceRadius;

        if( dDistance > MIN_ATTACH_DISTANCE && dDistance < MAX_ATTACH_DISTANCE )
            pCandidate->SetIsPossible(TRUE);
    }
}
PruneList();
return m_posList.GetCount();
}

int CDocking::CheckStickyFeetPositions()
{
    CVector *pvTemp;
    double dX, dY, dZ;
    VECTOR vecOrient, vecCand, vecOldPos, vecNewPos;
    int nLegsAvailable, nLegsStuck;

    if (!AttachmentLegsAvailable())
        return m_posList.GetCount();

    WPGetWorkpieceLocation( WORK_PIECE, &vecOrient, &vecOldPos );
    m_bDrawCandidates = FALSE;

    POSITION pos = GetFirstCandidatePos();
    while( pos != NULL )
    {
        CCandidate* pCandidate = GetNextCandidate( pos );

        // check it:
        if( pCandidate->GetIsPossible() )
        {
            pvTemp = pCandidate->GetPos();
            pvTemp->GetAll( &dX, &dY, &dZ );
            MakeVector( (LPVECT)&vecCand, dX, dY, dZ );

            CopyVector(&vecOldPos, &vecNewPos);
            // Set the workpiece location to be relative to candidate position:
            WorldCoords(&vecNewPos, FALSE);
            VectorSubtract(&vecNewPos, &vecCand, &vecNewPos);
            WPSetWorkpieceLocation( WORK_PIECE, &vecOrient, &vecNewPos );
            MenuOnEditLocation(FALSE, FALSE);

            // Check for sticky feet positions (arm.c Best Posn):
            nLegsAvailable = nLegsStuck = 0;
            bTSWPCoords = TRUE;
            double dFootAttachValue = 0.0;

            for( nTSLegID=PORT; nTSLegID<=UPPER; nTSLegID++ )

```

## Appendix E: Docking Library Main Source Code

```

        {
            if( bTSLegAvailable[nTSLegID] )
            {
                nLegsAvailable++;
                if (TSFindBestAttachmentPoint(nTSLegID, m_bShowRedraws, FALSE))
                {
                    CString strTmp;
                    VECTOR vecPos;
                    if (m_bShowRedraws)
                    {
                        GetVector( &vecCand, &dX, &dY, &dZ );
                        strTmp.Format("Pos: %6.3lf, %6.3lf, %6.3lf", dX, dY, dZ);
                        AfxMessageBox(strTmp);
                        MakeVector(&vecPos, 0.0, 0.0, 0.0);
                        TSConvLegToWorld(nTSLegID, &vecPos, NULL);
                        VectorAdd(&vecPos, &vecCand, &vecPos);
                        GetVector(&vecPos, &dX, &dY, &dZ);
                        strTmp.Format("Leg: %d, %6.3lf, %6.3lf, %6.3lf", nTSLegID, dX, dY, dZ);
                        AfxMessageBox(strTmp);
                    }

                    nLegsStuck++;
                    GUICalculateLegPositions( );
                }
            }
            /*
                else // this foot couldn't stick
            {
                pCandidate->SetIsPossible(FALSE);
                // if a single failure at this location, jump out completely:
                break;
            }
            */
        }
        if (nLegsAvailable > 0)
        {
            if (nLegsStuck < 2)
                pCandidate->SetIsPossible(FALSE);
            else
                pCandidate->SetLegsAttachedRatio((double)nLegsStuck/(double)nLegsAvailable);
        }
        nTSLegID=PORT;
        // Draw current location (whether abandoned or not):
        if (nLegsStuck > 0 && m_bShowRedraws)
        {
            // show feet deployed
            MenuOnEditLocation(TRUE, FALSE);
            MenuOnEditSuctionFoot();
            // stow feet
            // this seems to produce slightly strange graphics:
            // (might need to use full redraw rather than MenuOnEditSuctionFoot? Haven't tried)
            // TSStowDeployAllLegs(TRUE);
            // MenuOnEditSuctionFoot();
        }
    }
    PruneList();

    // Reset to start position:
    WPSetWorkpieceLocation( WORK_PIECE, &vecOrient, &vecOldPos );
    TSStowDeployAllLegs(TRUE);
    MenuOnEditSuctionFoot();
    MenuOnEditLocation(m_bShowRedraws, FALSE);
    m_bDrawCandidates = TRUE;

    return m_posList.GetCount();
}

int CDocking::CheckForReach()
{
    double dX, dY, dZ, dInitialExtend, dInitialRotate;
    VECTOR vecOrient, vecCand, vecOldPos, vecNewPos;
    CVector *pvTemp;
    CVector vDistance;
    char szMsg[STD_STR_LEN], szBfr[STD_STR_LEN];

    SysGetExtendRotatePositions(&dInitialExtend, &dInitialRotate);
    WPGetWorkpieceLocation( WORK_PIECE, &vecOrient, &vecOldPos );
    m_bDrawCandidates = FALSE;

    POSITION pos = GetFirstCandidatePos();
    while( pos != NULL )
    {
        CCandidate* pCandidate = GetNextCandidate( pos );

        // check it:
        if( pCandidate->GetIsPossible() )
        {
            pvTemp = pCandidate->GetPos();
            pvTemp->GetAll( &dX, &dY, &dZ );
            MakeVector( (LPVECT)&vecCand, dX, dY, dZ );

```

## Appendix E: Docking Library Main Source Code

```

        CopyVector(&vecOldPos, &vecNewPos);
        // Set the workpiece location to be relative to candidate position:
        WorldCoords(&vecNewPos, FALSE);
        VectorSubtract(&vecNewPos, &vecCand, &vecNewPos);
        WPSetWorkpieceLocation( WORK_PIECE, &vecOrient, &vecNewPos );
        MenuOnEditLocation(FALSE, FALSE);

// Let's check at this location (every 1cm for now)
PathSetUpSteps(0.01, FALSE);

double dAccess = 0.0;
double dExtend = 0.0;
        BOOL bRotateIncreasing = TRUE;
        if (dTSMInDeployRotate > dTSMMaxDeployRotate)
                bRotateIncreasing = FALSE;

pCandidate->SetAccessValue(0.0, 0.0, 0.0);
while (dExtend <= dTSMMaxDeployDistance)
{
        double dRotate = dTSMInDeployRotate;
        while ( (bRotateIncreasing && dRotate <= dTSMMaxDeployRotate) ||
                (!bRotateIncreasing && dRotate >= dTSMMaxDeployRotate) )
        {
                SysSetExtendRotatePositions(dExtend, dRotate);
                GUISetExtendCollPosition();
                if (m_bShowRedraws)
                {
                        GUISetScreenUpdate(RECALC_ALL | REDRAW_ALL);
                        MenuOnEditLocation(TRUE, FALSE);
                }
                dAccess = TaskCheckPath(FALSE);
                // Only bother to check for collisions if it has good access
                if (dAccess > pCandidate->GetAccessValue())
                {
                        if (dExtend == 0.0 && dRotate == 0.0)
                                pCandidate->SetAccessValue(dAccess, dExtend, dRotate);
                        else
                        {
                                // We may have caused a boom collision
                                bJobCollisionDetected = FALSE;
                                // setup collision arrangement
                                // (includes ColCheckLinksForContact corrected for boom arrangement)
                                // (ignore actual arm collisions because configuration is unknown)
                                JobCheckForCollision(FALSE);
                                // Update if not causing collision with boom
                                if (!CollisionSituation(FALSE))
                                {
                                        // Note we only store the best access each time so we have
                                        // to redo every check each time
                                        pCandidate->SetAccessValue(dAccess, dExtend, dRotate);

                                        if (FALSE)//m_bShowRedraws)
                                        {
                                                // Display to user
                                                LoadString(hInst, IDS_PATH_REACHPOS, szBfr, STD_STR_LEN);
                                                sprintf(szMsg, "At %4.2lf,%4.2lf,%4.2lf (E=%4.2lf, R=%4.2lf)
access is %3.0lf%%.", dX, dY, dZ, dExtend, dRotate, dAccess * 100.0);
                                                AfxMessageBox(szMsg, MB_OK);
                                        }
                                }
                                else if (m_bShowRedraws)
                                        AfxMessageBox("Collision with ROV (boom)!");
                        }
                }
                if (bRotateIncreasing)
                        dRotate += m_dRotateStep;
                else
                        dRotate -= m_dRotateStep;
                dExtend += m_dExtendStep;
        }
        if (pCandidate->GetAccessValue() < m_dAcceptanceLimit)
                pCandidate->SetIsPossible(FALSE);
}
PruneList();

// Reset to start position:
SysSetExtendRotatePositions(dInitialExtend, dInitialRotate);
GUISetExtendCollPosition();
WPSetWorkpieceLocation( WORK_PIECE, &vecOrient, &vecOldPos );
GUISetScreenUpdate(RECALC_ALL | REDRAW_ALL);
MenuOnEditLocation(m_bShowRedraws, FALSE);
m_bDrawCandidates = TRUE;

return m_posList.GetCount();
}

void CDocking::Serialize( CArchive& ar )

```

```

{
    // This may be useful for logging
    if( ar.IsStoring() )
    {
    }
    else
    {
    }
}

void CDocking::Draw()
{
    double adPoint[2][3], adN[3], adS[3];
    CVector *pvTemp;
    CVector vTemp;
    int nStartCol, nEndCol;

    if (m_bDrawCandidates)
    {
        for( int i=0; i<2; i++ )
            for( int j=0; j<3; j++ )
                adPoint[i][j] = 0.0;

        // Set up normal and sliding 'vectors':
        adN[0] = 0.0; adN[1] = 0.0; adN[2] = 1.0;
        adS[0] = 0.0; adS[1] = 1.0; adS[2] = 0.0;
        m_vWeldCofG.GetAll( &adPoint[0][0], &adPoint[0][1], &adPoint[0][2] );

        PlotGetColourRange( (LPSTR)"BLUE", &nStartCol, &nEndCol );

        // Iterate through all positions, drawing possible ones:
        POSITION pos = GetFirstCandidatePos();
        while( pos != NULL )
        {
            CCandidate* pCandidate = GetNextCandidate( pos );

            // check it:
            if( pCandidate->GetIsPossible() )
            {
                pvTemp = pCandidate->GetPos();
                vTemp = *pvTemp;
                WorldCoords(&vTemp);
                vTemp.GetAll( &adPoint[1][0], &adPoint[1][1], &adPoint[1][2] );
                GUIDisplayCross( adPoint[1], adN, adS );
                GUIDisplayLine( adPoint, 2, nEndCol );
            }
        }
    }

    void CDocking::DeleteCandidateList()
    {
        // Delete the list of candidate positions:
        while (!m_posList.IsEmpty())
            delete m_posList.RemoveHead();
    }

    int CDocking::WriteDefinitionFiles()
    {
        int nErr;

        // Initialize();
        // CheckForCollisions();
        // CheckForReach();
        // CheckStickyFeetReach();
        nErr = WriteStrengthsFile();
        if(!nErr)
            nErr = WriteTemplateFile();
        if(!nErr)
            nErr = WriteWeightsFile();
        if(!nErr)
            nErr = WriteLayoutFile();
        if(!nErr)
            nErr = WriteNetworkFile();
        return nErr;
    }

    void CDocking::PruneList()
    {
        POSITION pos1, pos2;
        CCandidate* pCand;

        // This is based on example code from RemoveAt help
        for( pos1 = m_posList.GetHeadPosition(); ( pos2 = pos1 ) != NULL; )
        {
            pCand = (CCandidate*)m_posList.GetNext(pos1); //pos1 now set to next object
            if (!pCand->GetIsPossible())
            {
                m_posList.RemoveAt(pos2);
            }
        }
    }
}

```

## Appendix E: Docking Library Main Source Code

```
        delete pCand; // Deletion avoids memory leak.
    }
}

int CDocking::WriteStrengthsFile()
{
    CString sFileName, strData;
    int nErr=0;

    sFileName = "e:\\neuralnw\\data\\phd\\rov.str";
    CStdioFile fDef;
    CFileException e;
    if( !fDef.Open( sFileName, CFile::modeCreate | CFile::modeWrite | CFile::typeText, &e ) )
    {
        #ifdef _DEBUG
            afxDump << "File could not be opened " << e.m_cause << "\n";
        #endif
    }
    else
    {
        strData = "set dlevel 1\\nget network rov.net\\nget weights rov.wts\\n";
        fDef.WriteString( strData );
        strData = "set mode clamp 1\\nset param estr 1.0\\nset param istr 0.2\\n";
        fDef.WriteString( strData );
        strData = "set ncyc 50\\nget anneal 2 20 .05 end\\nget unames ";
        fDef.WriteString( strData );

        //-----
        // Iterate through all positions, writing name labels of possible ones:
        double daPoint[3];

        POSITION pos = GetFirstCandidatePos();
        while( pos != NULL )
        {
            CCandidate* pCandidate = GetNextCandidate( pos );

            // check it:
            if( pCandidate->GetIsPossible() )
            {
                VECTOR vecBasePos;
                CVector vBasePos;
                double dX, dY, dZ;

                // Get ROV position:
                WPGetBaseLocation( NULL, &vecBasePos );
                GetVector( &vecBasePos, &dX, &dY, &dZ );
                vBasePos.SetAll( dX, dY, dZ );

                CVector* pvTemp = pCandidate->GetPos();
                CVector vTemp;
                vTemp = *pvTemp;
                WorldCoords(&vTemp);

                vTemp += vBasePos;
                vTemp.GetAll( &daPoint[0], &daPoint[1], &daPoint[2] );
                StringSetDouble(&strData, daPoint[0], 3, 2);
                fDef.WriteString( strData+" " );
                StringSetDouble(&strData, daPoint[1], 3, 2);
                fDef.WriteString( strData+" " );
                StringSetDouble(&strData, daPoint[2], 3, 2);
                fDef.WriteString( strData );
                fDef.WriteString( " " );
            }
        }

        //-----
        // Write labels for extra criteria

        fDef.WriteString("Legs Access");

        //-----

        fDef.Close();
    }
    return nErr;
}

int CDocking::WriteTemplateFile()
{
    CString sFileName, strData;
    int nErr=0;

    sFileName = "e:\\neuralnw\\data\\phd\\rov.tem";
    CStdioFile fDef;
    CFileException e;
    if( !fDef.Open( sFileName, CFile::modeCreate | CFile::modeWrite | CFile::typeText, &e ) )
    {
        #ifdef _DEBUG
            afxDump << "File could not be opened " << e.m_cause << "\n";
        #endif
    }
```

## Appendix E: Docking Library Main Source Code

```

    }
    else
    {
        //-----
        strData = "define: layout\n\n"; fDef.WriteString( strData );
        strData = "
                                ROV Docking Definition Created by ARM\n";
                                fDef.WriteString( strData );
        strData = "
                                =====\n\n\n";
                                fDef.WriteString( strData );
        strData = "$      Possible                                Cycleno                $\n";
                                fDef.WriteString( strData );
        strData = "      Locations                                updateno                $\n";
                                fDef.WriteString( strData );
        strData = "      cuname                                $\n";
                                fDef.WriteString( strData );
        strData = "      goodness                                $\n";
                                fDef.WriteString( strData );
        strData = "      temperature                                $\n\n";
        fDef.WriteString( strData );
        strData = "      Manip                                Sticky\n";
                                fDef.WriteString( strData );
        strData = "      Access                                Foot\nend\n";
                                fDef.WriteString( strData );
        strData = "rovdock      look      1      $ 0      activation      1 10 1 rovdock\n";
                                fDef.WriteString( strData );
        strData = "cycleno      variable 1      $ 1      cycleno      7 1\n";
                                fDef.WriteString( strData );
        strData = "updateno      variable 1      $ 2      updateno      7 1\n";
                                fDef.WriteString( strData );
        strData = "uname      variable 1      $ 3      cuname      -10 1.0\n";
                                fDef.WriteString( strData );
        strData = "goodness      floatvar 1      $ 4      goodness      7 1.0\n";
                                fDef.WriteString( strData );
        strData = "temperaturefloatvar 1      $ 5      temperature      7 1.0\n";
                                fDef.WriteString( strData );
        strData = "weight      matrix      5      $ 0      weight      h      4 10.0 0 16 0
16\n";
                                fDef.WriteString( strData );
        strData = "weight      vector      5      $ 2      uname      v      6 1 0 16\n";
                                fDef.WriteString( strData );
        strData = "weight      vector      5      $ 5      uname      h      4 1 0 16\n";
                                fDef.WriteString( strData );
        //-----

        fDef.Close();
    }
    return nErr;
}

int CDocking::WriteWeightsFile()
{
    CString sFileName, strData, sTmp;
    int nErr=0;

    sFileName = "e:\\neuralnw\\data\\phd\\rov.wts";
    CStdioFile fDef;
    CFileException e;
    if( !fDef.Open( sFileName, CFile::modeCreate | CFile::modeWrite | CFile::typeText, &e ) )
    {
        #ifdef _DEBUG
            afxDump << "File could not be opened " << e.m_cause << "\n";
        #endif
    }
    else
    {
        //-----
        // Iterate through all positions, writing possible ones:
        POSITION pos1 = GetFirstCandidatePos();
        // Set up weights for each element
        while( pos1 != NULL )
        {
            CCandidate* pCand1 = GetNextCandidate( pos1 );
            POSITION pos2 = GetFirstCandidatePos();
            // Set up weights for each element
            while( pos2 != NULL )
            {
                CCandidate* pCand2 = GetNextCandidate( pos2 );
                if( pos1 != pos2 )
                    StringSetDouble(&sTmp, EXCLUSIVE_VALUE, FIELD_WIDTH, PRECISION_DP);
                else
                    StringSetDouble(&sTmp, NO_CONNECTION, FIELD_WIDTH, PRECISION_DP);
                strData += sTmp;
            }
            // Add pCand1 leg values
            StringSetDouble(&sTmp, pCand1->GetLegsAttachedRatio(), FIELD_WIDTH,
PRECISION_DP);
            strData += sTmp;
            // Add pCand1 access values
            StringSetDouble(&sTmp, pCand1->GetAccessValue(), FIELD_WIDTH,
PRECISION_DP);

```

## Appendix E: Docking Library Main Source Code

```

        strData += sTmp;

        strData += "\n";
        fDef.WriteString( strData );
        strData.Empty();
    }

    // Add pCand2 leg values
    POSITION pos2 = GetFirstCandidatePos();
    while( pos2 != NULL )
    {
        CCandidate* pCand2 = GetNextCandidate( pos2 );
        StringSetDouble(&sTmp, pCand2->GetLegsAttachedRatio(), FIELD_WIDTH,
PRECISION_DP);
        strData += sTmp;
    }
    for (int i=0; i<EXTRA_CRITERIA; i++)
    {
        StringSetDouble(&sTmp, NO_CONNECTION, FIELD_WIDTH, PRECISION_DP);
        strData += sTmp;
    }
    strData += "\n";
    fDef.WriteString( strData );
    strData.Empty();

    // Add pCand1 access values
    pos2 = GetFirstCandidatePos();
    while( pos2 != NULL )
    {
        CCandidate* pCand2 = GetNextCandidate( pos2 );
        StringSetDouble(&sTmp, pCand2->GetAccessValue(), FIELD_WIDTH,
PRECISION_DP);
        strData += sTmp;
    }
    for (i=0; i<EXTRA_CRITERIA; i++)
    {
        StringSetDouble(&sTmp, NO_CONNECTION, FIELD_WIDTH, PRECISION_DP);
        strData += sTmp;
    }
    strData += "\n";
    fDef.WriteString( strData );
    strData.Empty();

    // Set up biases for each element
    for (i=0; i<m_posList.GetCount()+EXTRA_CRITERIA; i++)
    {
        StringSetDouble(&sTmp, DEFAULT_BIAS, FIELD_WIDTH, PRECISION_DP);
        strData += sTmp;
    }
    strData += "\n";
    fDef.WriteString( strData );

    //-----
    fDef.Close();
}
return nErr;
}

int CDocking::WriteLayoutFile()
{
    CString sFileName, strData, sTmp;
    int nErr=0, nElements = m_posList.GetCount();

    sFileName = "e:\\neuralnw\\data\\phd\\rov.100";
    CStdioFile fDef;
    CFileException e;
    if( !fDef.Open( sFileName, CFile::modeCreate | CFile::modeWrite | CFile::typeText, &e ) )
    {
        #ifdef DEBUG
            afxDump << "File could not be opened " << e.m_cause << "\n";
        #endif
    }
    else
    {
        //-----
        // Iterate through all positions, writing possible ones:
        StringSetInt(&strData, LAYOUT_ROWS);
        strData += " ";
        StringSetInt(&sTmp, LAYOUT_COLUMNS);
        strData += sTmp;
        strData += "\n";
        fDef.WriteString( strData );
        strData.Empty();

        // Set up weights for each element
        for (int i=0; i<LAYOUT_ROWS; i++)
        {
            for (int j=0; j<LAYOUT_COLUMNS; j++)

```

## Appendix E: Docking Library Main Source Code

```

    {
        if( j>=LAYOUT_STARTPOS )
        {
            // Do main row of nodes
            if( i==0 && j<LAYOUT_STARTPOS+nElements )
                StringSetInt(&sTmp, j-LAYOUT_STARTPOS, 3);
            // Do first extra criteria
            else if ( i==LAYOUT_EXTRAPOS && j==LAYOUT_STARTPOS )
                StringSetInt(&sTmp, nElements, 3);
            // Do second extra criteria
            else if ( i==LAYOUT_EXTRAPOS && j==LAYOUT_STARTPOS+15 )
                StringSetInt(&sTmp, nElements+1, 3);
            else
                sTmp = " .";
        }
        else
            sTmp = " .";
        strData += sTmp;
    }
    strData += "\n";
    fDef.WriteString( strData );
    strData.Empty();
}

//-----

fDef.Close();
}
return nErr;
}

int CDocking::WriteNetworkFile()
{
    CString sFileName, strData, sTmp;
    int nErr=0, nElements = m_posList.GetCount();

    sFileName = "e:\\neuralnw\\data\\phd\\rov.net";
    CStdioFile fDef;
    CFileException e;
    if( !fDef.Open( sFileName, CFile::modeCreate | CFile::modeWrite | CFile::typeText, &e ) )
    {
        #ifdef _DEBUG
            afxDump << "File could not be opened " << e.m_cause << "\n";
        #endif
    }
    else
    {
        strData = "definitions:\n";
        fDef.WriteString( strData );
        strData.Format((LPCTSTR)"nunits %d\n", m_posList.GetCount()+EXTRA_CRITERIA);
        fDef.WriteString( strData );
        strData.Format((LPCTSTR)"ninputs %d\n", m_posList.GetCount()+EXTRA_CRITERIA);
        fDef.WriteString( strData );
        strData.Format((LPCTSTR)"nupdates %d\n", m_posList.GetCount()+EXTRA_CRITERIA);
        fDef.WriteString( strData );
        strData = "end\nnetwork:\n";
        fDef.WriteString( strData );
        //-----
        // Iterate through all positions, writing position for weights:
        for (int i=0; i<m_posList.GetCount()+EXTRA_CRITERIA; i++)
        {
            strData.Empty();
            for (int j=0; j<m_posList.GetCount()+EXTRA_CRITERIA; j++)
            {
                sTmp = " .";
                strData += sTmp;
            }
            strData += "\n";
            fDef.WriteString( strData );
        }
        strData = "end\nbiases:\n";
        fDef.WriteString( strData );
        // Iterate through all positions, writing position for biases:
        strData.Empty();
        for (i=0; i<m_posList.GetCount()+EXTRA_CRITERIA; i++)
        {
            sTmp = " .";
            strData += sTmp;
        }
        strData += "\nend\n";
        fDef.WriteString( strData );
        //-----

        fDef.Close();
    }
    return nErr;
}

```



## Appendix E: Docking Library Main Source Code

```
int CDocking::CloseDown()
{
    DeleteCandidateList();
    return 0;
}

void CDocking::WorldCoords(CVector * pVec, BOOL bForward)
{
    if (bForward)
        *pVec = *pVec + m_vWeldCofG;
    else
        *pVec = *pVec - m_vWeldCofG;
}

void CDocking::WorldCoords(LPVECT pVec, BOOL bForward)
{
    double dX, dY, dZ;
    VECTOR vecTemp;

    m_vWeldCofG.GetAll( &dX, &dY, &dZ );
    MakeVector( (LPVECT)&vecTemp, dX, dY, dZ );
    if (bForward)
        VectorAdd(pVec, &vecTemp, pVec);
    else
        VectorSubtract(pVec, &vecTemp, pVec);
}

BOOL CDocking::CollisionSituation(BOOL bReport)
{
    COBJECT ccTempCollisionObject;
    int nContactMade=0;
    char szContactText[96];
    char szContact1[32], szContact2[32];
    BOOL bCollision = FALSE;

    // for each object in workpiece (mostly cylinders)...
    for (int i=nBaseCollisionObjects; i<ColGetNCollisionObjects(); i++)
    {
        ColGetCollisionDetails(i, &ccTempCollisionObject);
        if (ccTempCollisionObject.nColObjectType == COLCUBOID_OBJECT)
            break; // we cant yet check 'other fixtures' against other cuboids

        // ...check against each object in ROV & toolskid
        for (int j=0; j<nBaseCollisionObjects; j++)
        {
            ColGetCollisionDetails(j, &ccTempCollisionObject);
            if (ccTempCollisionObject.nColObjectType == COLCUBOID_OBJECT)
            {
                nContactMade = ColCheckCylinderAgainstCuboid(i, j);
                if (nContactMade == 3)
                {
                    AfxMessageBox("Error in collision handling");
                    return 0;
                }
                else if (nContactMade == 1 || nContactMade == 2)
                    break;
            }
            else
            {
                nContactMade = ColCheckContactBetweenCylinders(i, j);
                if (nContactMade == 3)
                {
                    AfxMessageBox("Collision handling", MB_OK);
                    return 0;
                }
                else if (nContactMade == 1 || nContactMade == 2)
                    break;
            }
        }

        if( nContactMade==1 || nContactMade==2 )
        {
            bCollision = TRUE;
            if (bReport)
            {
                lstrcpy((LPSTR)szContact1,
(LPSTR)ccTempCollisionObject.szColObjectName);

                ColGetCollisionDetails(i, &ccTempCollisionObject);
                lstrcpy( (LPSTR)szContact2,
(LPSTR)ccTempCollisionObject.szColObjectName );

                wsprintf( (LPSTR)szContactText, (LPSTR)"%s and %s",
(LPSTR)szContact1,
(LPSTR)szContact2 );
                AfxMessageBox((LPCTSTR)szContactText, MB_OK);
            }
            break;
        }
    }
}
```

## Appendix E: Docking Library Main Source Code

```
    }
    return bCollision;
}

void CDocking::ToggleRedraws()
{
    if (m_bShowRedraws)
        m_bShowRedraws = FALSE;
    else
        m_bShowRedraws = TRUE;
}

void CDocking::ChangeSettings()
{
    CDockSettingsDlg dlg;

    dlg.m_dXStart = GMRound(m_dXStart, 3);
    dlg.m_dYStart = GMRound(m_dYStart, 3);
    dlg.m_dZStart = GMRound(m_dZStart, 3);
    dlg.m_dXEnd = GMRound(m_dXEnd, 3);
    dlg.m_dYEnd = GMRound(m_dYEnd, 3);
    dlg.m_dZEnd = GMRound(m_dZEnd, 3);
    dlg.m_dGridSpacing = m_dGridSpacing;
    dlg.m_dExtendStep = m_dExtendStep;
    dlg.m_dRotateStep = 180.0 * (m_dRotateStep/PI);
    dlg.m_dAcceptanceLimit = m_dAcceptanceLimit * 100.0;

    // Show dialog
    if (dlg.DoModal() == IDOK)
    {
        m_dXStart = dlg.m_dXStart;
        m_dYStart = dlg.m_dYStart;
        m_dZStart = dlg.m_dZStart;
        m_dXEnd = dlg.m_dXEnd;
        m_dYEnd = dlg.m_dYEnd;
        m_dZEnd = dlg.m_dZEnd;
        m_dGridSpacing = dlg.m_dGridSpacing;
        m_dExtendStep = dlg.m_dExtendStep;
        m_dRotateStep = PI * (dlg.m_dRotateStep/180.0);
        m_dAcceptanceLimit = dlg.m_dAcceptanceLimit/100.0;
    }
}

void CDocking::ShowResults()
{
    CDockResultsDlg dlg;
    double dRoll, dPitch, dYaw;
    CVector vTemp;
    int i=1;
    MATRIX matBase;

    GetVector(&m_vecBaseOrient, &dRoll, &dPitch, &dYaw);
    VectorGetRotationMatrix(dRoll, dPitch, dYaw, &matBase);

    POSITION pos = GetFirstCandidatePos();
    while( pos != NULL )
    {
        CCandidate* pCandidate = GetNextCandidate( pos );

        // check it:
        if( pCandidate->GetIsPossible() )
        {
            double dX, dY, dZ;
            CString strPosition;
            VECTOR vecTemp;

            vTemp = *(pCandidate->GetPos());
            // Convert to ROV coords
            WorldCoords(&vTemp);

            // Convert to rig coords
            vTemp.GetAll( &dX, &dY, &dZ );
            MakeVector(&vecTemp, dX, dY, dZ);
            MatrixVectorMultiply(&matBase, &vecTemp, &vecTemp);
            VectorAdd(&vecTemp, &m_vecBasePos, &vecTemp);
            GetVector(&vecTemp, &dX, &dY, &dZ);

            strPosition.Format("%2d: %5.2lf, %5.2lf, %5.2lf. E=%5.2lf, R=%d: Access = %d%%. Legs
= %5.2lf", i, dX, dY, dZ,
                pCandidate->GetExtend(),
                (int)(180.0*(pCandidate->GetRotate()/PI)),
                (int)(100.0*pCandidate->GetAccessValue()),
                pCandidate->GetLegsAttachedRatio());
            dlg.m_saPositions.Add(strPosition);
            i++;
        }
    }
    dlg.DoModal();
}
```

## *Appendix E: Docking Library Main Source Code*

```
BOOL CDocking::AttachmentLegsAvailable()
{
    BOOL bLegsAvailable = FALSE;
    for( nTSLegID=PORT; nTSLegID<=UPPER; nTSLegID++ )
    {
        if( bTSLegAvailable[nTSLegID] )
        {
            bLegsAvailable = TRUE;
        }
    }

    return bLegsAvailable;
}
```

## **Candidate.h**

```

/*****
/* FILE                CANDDATE.H                                */
/*-----*/
/* File Name          :  \Include\Canddate.h                    */
/* Class Name         :  CCandidate                             */
/* Purpose            :  Class header file for possible docking positions */
/* Author             :  T.Larkum                               */
/* Written on         :  08/12/95                                */
/*-----*/
/* Copyright (c) Technical Software Consultants Ltd. 1995.      */
/*-----*/
/* HISTORY                                                     */
/*-----*/
/* Changed by   Date       Reason                               */
/*-----*/
/*-----*/
/*****

#ifndef _INC_CAND
#define _INC_CAND

/*-----*/
/* INCLUDE FILES                                             */
/*-----*/

#include <vector.hpp>

/*-----*/
/* CLASS DEFINITION                                           */
/*-----*/

class CCandidate : public CObject
{
public:
    CCandidate( );
    CCandidate( CVector* pvTemp );
    ~CCandidate( );

protected:
    DECLARE_SERIAL( CCandidate )

// Attributes
protected:
    CVector m_vPos;
    BOOL    m_bPossible;
    double  m_dReachValue;          // Percentage of weld reachable
    double  m_dExtend, m_dRotate;   // at this arrangement
    double  m_dLegsAttachedRatio;

// Operations
public:
    void    Initialize();
    CVector* GetPos();
    BOOL    GetIsPossible();
    void    SetIsPossible( BOOL bFlag );

// Helper functions
protected:
public:
    double GetLegsAttachedRatio();
    void SetLegsAttachedRatio(double dRatio);
    double GetRotate();
    double GetExtend();
    double GetAccessValue();
    void SetAccessValue(double dValue, double dExtend, double dRotate);
    virtual void Serialize( CArchive& ar );
};

#endif // _INC_CAND

```

## **Candidate.cpp**

```

/*****
/*  EILE                                CANDDATE.CPP                                */
/*-----*/
/* File Name      :  \Docking\Canddate.cpp                                */
/* Class Name     :  CCandidate                                           */
/* Purpose        :  Implementation of the CCandidate class              */
/*                 :  - a Candidate is a possible docking position        */
/* Author         :  T.Larkum                                              */
/* Written on     :  11/12/95                                              */
/*-----*/
/* Copyright (c) Technical Software Consultants Ltd. 1995.                */
/*-----*/
/* HISTORY                                                */
/*-----*/
/* Changed by   Date      Reason                                          */
/*-----*/
/*-----*/
/*****

/*-----*/
/* INCLUDE FILES                                          */
/*-----*/

#include <stdafx.h>

#include <canddate.h>

/*-----*/
/* MFC MACROS                                            */
/*-----*/

IMPLEMENT_SERIAL( CCandidate, CObject, 1 );

/*-----*/
/* CLASS IMPLEMENTATION                                  */
/*-----*/

CCandidate::CCandidate()
{
    Initialize();
}

CCandidate::CCandidate( CVector* pvTemp )
{
    m_vPos = *pvTemp;
    Initialize();
}

CCandidate::~CCandidate()
{
}

void CCandidate::Initialize( )
{
    m_bPossible = TRUE;
    m_dReachValue = 0.0;
    m_dExtend = 0.0;
    m_dRotate = 0.0;
    m_dLegsAttachedRatio = 0.0;
}

CVector* CCandidate::GetPos()
{
    return (CVector*)&m_vPos;
}

void CCandidate::SetIsPossible( BOOL bFlag )
{
    m_bPossible = bFlag;
}

BOOL CCandidate::GetIsPossible()
{
    return m_bPossible;
}

void CCandidate::Serialize( CArchive& ar )
{
    // This may be useful for logging
    if( ar.IsStoring() )
    {
    }
    else
    {
    }
}

```

## *Appendix E: Docking Library Main Source Code*

```
void CCandidate::SetAccessValue(double dValue, double dExtend, double dRotate)
{
    m_dReachValue = dValue;
    m_dExtend = dExtend;
    m_dRotate = dRotate;
}

double CCandidate::GetAccessValue()
{
    return m_dReachValue;
}

double CCandidate::GetExtend()
{
    return m_dExtend;
}

double CCandidate::GetRotate()
{
    return m_dRotate;
}

void CCandidate::SetLegsAttachedRatio(double dRatio)
{
    m_dLegsAttachedRatio = dRatio;
}

double CCandidate::GetLegsAttachedRatio()
{
    return m_dLegsAttachedRatio;
}
```

### F.1. Mobilisation

The Shelf Supporter with the ROV crew and NICS toolskid on board (see Figure F.1) left port on 27<sup>th</sup> August to transit to NRA, arriving about midnight. The next day the Triton was tested in the water and suffered the first of a number of faults (TMS winch, system oil leak, compensator leak); the Supporter was also moved off station a number of times for other work. Since the FATs Covus had developed a new high pressure water cleaning system (to allow the removal of hard and soft marine growth from the weld areas before inspection) and this was initially fitted to both NICS manipulators.

Finally on 1<sup>st</sup> September the Triton was deployed carrying the full NICS skid configured for HP water cleaning and the first system check with the ARM Software was conducted; unfortunately this was cut short by another oil leak.



Figure F.1 – NICS on deck, fitted with twin HP water jets, awaiting deployment

On 2<sup>nd</sup> September the system was deployed again (see Figure 9.8) and this time successfully flew down to node 4E2, docked on, and successfully conducted automated weld cleaning from clock positions 1.00 to 6.30 and 8.00 to 10.00.

## **F.2. Node 4E2, Weld 8**

The next day, 3<sup>rd</sup> September, the system was reconfigured for ACFM inspection and launched overboard at 1400 (see Figure F.2). At 1515 it was in position, docked on 4E2, and began inspecting – a significant moment, the first operational use of the ARM Software after a decade in development and also believed to be the first operational robotic deployment of the ACFM array probe inspection system.



Figure F.2 – ARM NICS System deploying into the water

By 2000 it had successfully inspected from 2.30 to 6.00 on the chord toe (highlighted in Figure F.3) and 3.00 to 5.00 on the brace toe. Following a wrist rotation problem the system was recovered. After repair it was redeployed back onto the node at 0440 on 4<sup>th</sup> September, when it inspected from 6.00 to 10.30 on the chord toe and 6.30 to 9.00 on the brace toe. Early the next morning, 5<sup>th</sup> September, it inspected 8.30 to 10.30 on the brace toe, then 1.30 to 2.30 on the chord toe, then 1.30 to 3.00 on the brace toe, after which it was withdrawn and flown to node 4G2 to conduct manual cleaning.



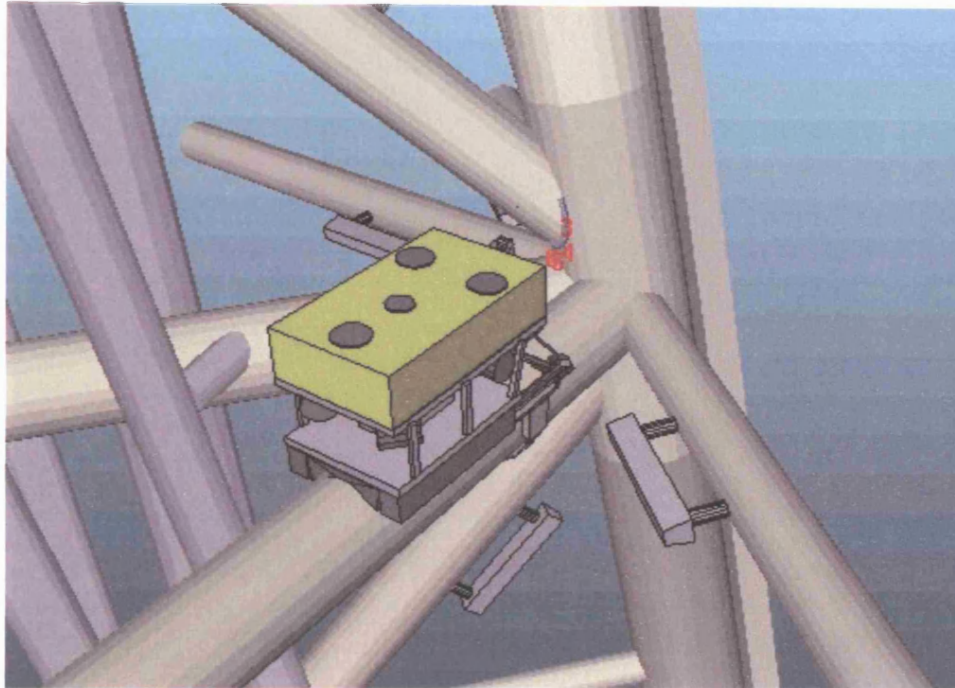


Figure F.3 – ARM view of node 4E2, 2.30 – 6.00 on weld highlighted

### F.3. Node 4G2, Weld 5

ACFM inspection on 4G2 began in the early afternoon of 7<sup>th</sup> September, with the toolskid clamped to an anode, achieving 7.00 to 9.00 on the chord toe and 7.30 to 8.30 on the brace toe, then the next day 2.30 to 5.00 on both toes (the final position is shown in Figures 9.9 and 9.10).

To get beyond 5.00 the right-angled probe mount was fitted on 10<sup>th</sup> September and this allowed inspection from 5.00 to 7.00 on the chord toe (the 5.30 position is shown in Figure F.4) and 5.00 to 7.30 on the brace toe.



Figure F.4 – Inspection of 5.30 position on node 4G2 using right-angled probe mounting

#### **F.4. Node 3C2, Weld 1**

Manual cleaning of Weld 1 (brace L317) on 3C2 took place on the morning of 9<sup>th</sup> September. Unfortunately while manoeuvring the ROV around with the claw open, it collided with the structure causing one of the claw hydraulic hoses to burst. This caused a massive oil leak, forcing an emergency recovery of the ROV which succeeded just before it shut down through lack of oil. Its recovery did, however, provide a rare opportunity to see the large docking claw in the open position and the ROV in its 'slid back' position (see Figure F.5).

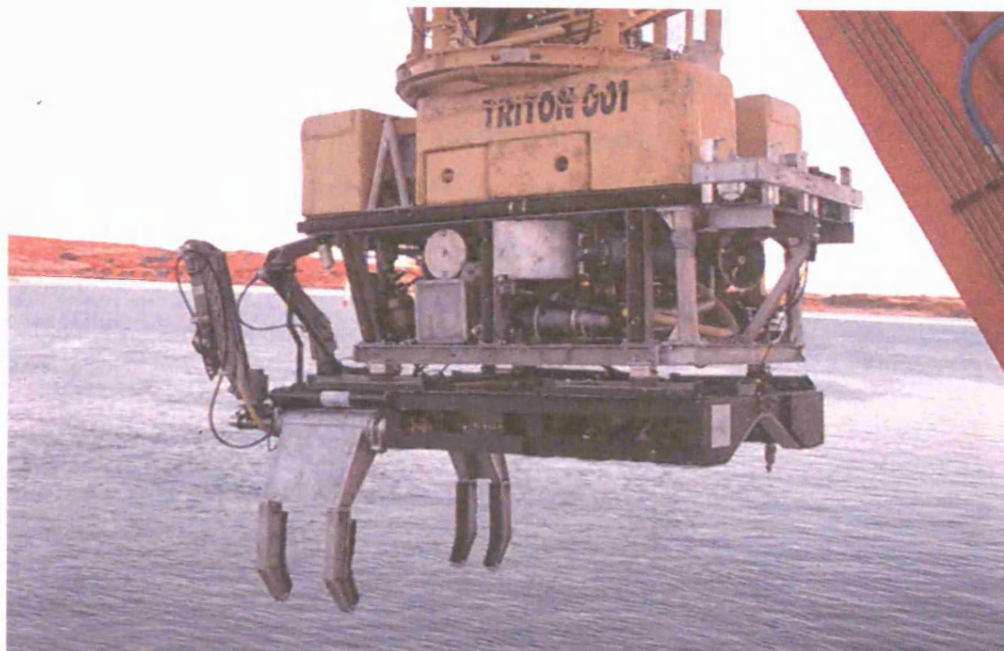


Figure F.5 – ARM NICS System showing the underside claw and ROV in rear position

ACFM inspection of the weld began late that night with the ROV sitting on the brace to the 'left' of L317 (as viewed when looking at the weld). By 3am the next day the weld had been inspected from 6.45 to 11.30 on the chord toe (see Figures F.6 and F.7), and 6.45 to 8.00 and 10.30 to 11.15 on the brace toe.



Figure F.6 – Inspection of 11 o'clock position on node 3C2



In the afternoon of the 10<sup>th</sup> the ROV was flown down again, with the ACFM probe swapped over to the left manipulator, and it docked onto the brace to the 'right' of L317. From there it was possible to inspect 11.30 to 1.30 on the chord toe; it was not possible to inspect the brace toe because of another brace intersecting the node just above the weld.

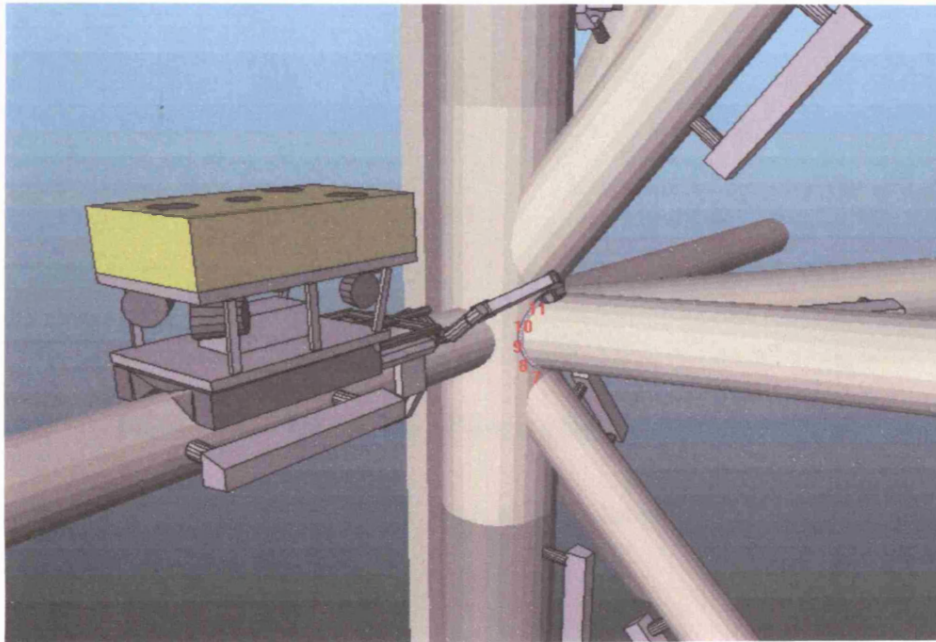


Figure F.7 – ARM view of inspection of 11 o'clock position

The interstitial segment of the weld between this brace and the next one to the 'right' was also inspected (see Figure F.8), approximately equivalent to 1.00 to 3.00 on the brace weld.



Figure F.8 – Inspection of interstitial weld on node 3C2

At 2200 on 10<sup>th</sup> September the NICS system was recovered from node 4G2 and the operational work came to a close.

## **Contents**

The CD-ROM contains the following items:

### Root directory:

- Setup.exe: a directly installable version of the NNW neural network software
- NeuralNW.exe: the NNW executable which can just be copied onto a PC to run

### Data directory:

- Manual test definition files
- ARM docking library test definition files
- Tic-tac-toe test definition files

### Docking directory:

- ARM docking library source code

### NeuralNW directory:

- Full source code to NNW (approximately 300 files)