COMPLIANT FORCE CONTROL FOR AUTOMATED SUB-SEA INSPECTION

by

Jason Tisdall  B.Eng.

A Thesis Submitted for the Degree of Doctor of Philosophy in the Faculty of Engineering

UNIVERSITY OF LONDON

August 1997

Department of Mechanical Engineering
University College London
Torrington Place
LONDON WC1E 7JE
I dedicate this to my wife Andrea who, although not always entirely sure of what I was doing, has always remained for me and in support of me.
This thesis describes a course of work aimed at developing practical methods of implementing compliant force control to enable the automation of sub-sea inspection of existing offshore structures. The overall system is based on a six degree-of-freedom (dof) hydraulic manipulator mounted on a two dof platform which is mounted on a remotely operated vehicle (ROV). For inspection tasks, the ROV attaches itself to the structure using three vacuum feet to maintain a reasonably stiff working platform. The manipulator may then be deployed to surface track the weld of two or more intersecting structural members whilst holding the appropriate inspection probe. For this project an Alternating Current Field Measurement (ACFM) probe is used. This does not need electrical surface contact but instead can operate with a constant elevation above a surface provided by the probe's own casing.

This thesis also considers the likely motions that will be experienced by the real system and examines the use of vision systems to compensate for unwanted movement. It further examines the design of an appropriate compliance to sit between the manipulator and the probe and describes the construction of this compliance device.

A novel method is offered that uses an artificial neural network (ANN) to simplify the problem of mapping a robot tool position in space when using compliant force control. For this case, the compliant device is a network of springs between plates situated between the manipulator end flange and the tool. A force sensor is mounted between the tool and the compliant device. To overcome the difficulties of accurately modelling the non-linear and highly coupled characteristics of the compliant device, a back propagating ANN was trained to relate the forces to positions for real data sets. The finished ANN was used as part of a computer model of the overall manipulator system.

The ANN was then used as part of a force control loop to enable surface tracking.
I would like to express my thanks to all those at UCL who have helped me technically, socially, emotionally and spiritually during the past four years of my life.

My particular thanks go to Professor David Broome, my supervisor, for his guidance, support and vision and to Alistair Greig, my other supervisor, for really detailed feedback on my work.

In the control laboratory, I want to thank Alex Fairweather, Susana Rivas, Duncan Wilson, Hugh Martindale, Patrick Ata, William Suen, Nader Safari and Szen Ong for their friendship and help with all aspects of my life. These are people who have helped keep a PhD fun. I also want to thank Linda Luck, Margaret Harrison, Charlotte Reisch and George Slater in the departmental office for helping make things work within the department and for caring.

Special thanks are due to Martin Hall and Trevor Larkum of Technical Software Consultants Ltd who, whilst working on parallel projects, have never failed to help me when I needed assistance.

For some of the particular sections of experimental work, I would like to acknowledge the help received from Mobil North Sea Ltd, Offshore Supplies Office, Slingsby Engineering Ltd. and Subsea Offshore Ltd.

My final thanks go to a very special friend, Hilary Goodman, who for the last year has taken on the role of supporting me with my PhD by regular meetings, discussions of progress and really lovely lunches. Without Hilary, I think I would not have finished with nearly the level of enjoyment that I have had.

Thank you all

26th August 1997
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTENTS</td>
<td>9</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>15</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>17</td>
</tr>
<tr>
<td>LIST OF ACRONYMS</td>
<td>19</td>
</tr>
<tr>
<td>LIST OF SYMBOLS</td>
<td>21</td>
</tr>
<tr>
<td>CHAPTER 1: INTRODUCTION</td>
<td>25</td>
</tr>
<tr>
<td>1.1. The Requirement for Oil</td>
<td>25</td>
</tr>
<tr>
<td>1.2. Getting Oil From An Offshore Oil Field</td>
<td>25</td>
</tr>
<tr>
<td>1.3. The Problem: Cracks Forming due to Fatigue and Corrosion of an Offshore Jacket</td>
<td>26</td>
</tr>
<tr>
<td>1.4. The Current Solution: Detection and Repair of a Fatigue Crack Using Divers</td>
<td>27</td>
</tr>
<tr>
<td>1.5. The Robotic Solution: Detection and Repair of a Fatigue Crack Using Unmanned Submersible Vehicles Carrying Robotic Arms</td>
<td>29</td>
</tr>
<tr>
<td>1.6. Summary of Missions</td>
<td>32</td>
</tr>
<tr>
<td>1.7. Conclusions</td>
<td>33</td>
</tr>
<tr>
<td>1.8. Summary of Chapters in the Ph.D.</td>
<td>33</td>
</tr>
<tr>
<td>CHAPTER 2: ROBOTICS AND FORCE CONTROL</td>
<td>35</td>
</tr>
<tr>
<td>2.1. Background to Robotics</td>
<td>35</td>
</tr>
<tr>
<td>2.2. Force Control</td>
<td>36</td>
</tr>
<tr>
<td>2.2.1. The Need for Force Control</td>
<td>36</td>
</tr>
<tr>
<td>2.2.2. Types of Force Control</td>
<td>37</td>
</tr>
<tr>
<td>2.2.3. Key Aspects of Force Control</td>
<td>39</td>
</tr>
<tr>
<td>2.3. Conclusions</td>
<td>43</td>
</tr>
<tr>
<td>CHAPTER 3: DEFINING THE PROBLEM</td>
<td>45</td>
</tr>
<tr>
<td>3.1. Chapter Outline</td>
<td>45</td>
</tr>
<tr>
<td>3.2. The Automated Remote Manipulator (ARM) System</td>
<td>45</td>
</tr>
<tr>
<td>3.2.1. Remotely Operated Vehicle (ROV)</td>
<td>47</td>
</tr>
<tr>
<td>3.2.2. Manipulator Arm</td>
<td>48</td>
</tr>
<tr>
<td>3.2.3. Tool Skid</td>
<td>48</td>
</tr>
<tr>
<td>3.2.4. Control System</td>
<td>48</td>
</tr>
<tr>
<td>3.3. ROV Motion</td>
<td>49</td>
</tr>
<tr>
<td>3.3.1. ROV Motion Due to Sea Currents</td>
<td>49</td>
</tr>
<tr>
<td>3.3.2. ROV Motion Due to Wave action</td>
<td>51</td>
</tr>
<tr>
<td>3.3.3. ROV Motion Due to Contact Forces</td>
<td>51</td>
</tr>
<tr>
<td>3.4. Required Surface Contact to Achieve a Weld Inspection</td>
<td>51</td>
</tr>
<tr>
<td>3.5. The Requirement for Compliant Force Control</td>
<td>53</td>
</tr>
<tr>
<td>3.6. Conclusions</td>
<td>55</td>
</tr>
<tr>
<td>CHAPTER 4: SUPERVISORY CONTROL OF THE PUMA 560 MANIPULATOR</td>
<td>57</td>
</tr>
<tr>
<td>4.1. Chapter Outline</td>
<td>57</td>
</tr>
<tr>
<td>4.2. General Description</td>
<td>57</td>
</tr>
<tr>
<td>4.3. PUMA 560 Manipulator Geometry</td>
<td>59</td>
</tr>
<tr>
<td>4.4. PUMA 560 Manipulator Controller</td>
<td>61</td>
</tr>
<tr>
<td>4.4.1. Hardware</td>
<td>61</td>
</tr>
<tr>
<td>4.4.2. Software</td>
<td>62</td>
</tr>
<tr>
<td>4.5. Supervisory Controller</td>
<td>63</td>
</tr>
</tbody>
</table>
CHAPTER 8: USING ARTIFICIAL NEURAL NETWORKS TO MODEL THE COMPLIANT DEVICE

8.1. Introduction..................................................................................................................127
8.2. Choosing an Artificial Neural Network Structure .........................................................127
  8.2.1. Artificial Neural Networks.......................................................................................127
  8.2.2. Back Propagating Artificial Neural Networks .......................................................128
8.3. Experimental Procedure For Data Gathering .................................................................131
  8.3.1. Experimental Method ............................................................................................131
  8.3.2. Experimental Equipment .......................................................................................131
8.4. Training the Artificial Neural Network ..........................................................................133
  8.4.1. Minimising the Training Time for Artificial Neural Networks ....................... 133
  8.4.2. Pre Processing of Data .........................................................................................134
  8.4.3. Selection of the Number of Hidden Layer Neurons .......................................... 135
  8.4.4. Training ..............................................................................................................138
8.5. Verifying the Artificial Neural Network .........................................................................138
  8.5.1. Comparison Between Training and Verification Data Sets ............................... 138
  8.5.2. Interpretation of the Error Distributions .............................................................. 140
  8.5.3. Comparison With A Simple Spring Model ..............................................................141
8.6. Conclusions ..................................................................................................................143

CHAPTER 9: KINEMATICS: EULER ANGLES AND HOMOGENEOUS TRANSFORMATION MATRICES

9.1. Introduction..................................................................................................................145
9.2. Task Kinematics ..........................................................................................................145
  9.2.1. Manipulator Kinematics .......................................................................................145
  9.2.2. Euler Angles ........................................................................................................146
  9.2.3. Rotation Matrices .................................................................................................147
  9.2.4. Homogeneous Transformation Matrices ..............................................................149
9.3. Application of Transformation Matrices in this Thesis ................................................151
  9.3.1. Use of Transformation Matrices to Map the Probe Position ...............................151
  9.3.2. Extraction of the Data from the Artificial Neural Network ...............................152
9.4. Conclusions ..................................................................................................................156

CHAPTER 10: EXPERIMENTAL USE OF AN ARTIFICIAL NEURAL NETWORK MODEL TO ENABLE COMPLIANT FORCE CONTROL

10.1. Introduction................................................................................................................157
10.2. Experiment ................................................................................................................157
  10.2.1. Apparatus ...........................................................................................................157
  10.2.1. Operation of Hardware .......................................................................................158
10.3. Method .......................................................................................................................159
10.4. Results .......................................................................................................................160
  10.4.1. Recorded Toe and Heel Co-ordinates ................................................................ 160
  10.4.2. Surfaces Mapped by the Artificial Neural Network ........................................... 161
10.5. Conclusions ................................................................................................................164
  10.5.1. Sources of Error .................................................................................................164
  10.5.2. Error Due to Calibration .....................................................................................165
  10.5.3. Summary ...........................................................................................................166

CHAPTER 11: CONCLUSIONS

11.1. Introduction................................................................................................................169
11.2. Thesis Summary ........................................................................................................169
11.3. The PUMA 560 and Associated Control System .......................................................170
11.4. Applicability of Existing Sub-Sea Intervention Systems ...........................................171
11.5. ROV Motions and Disturbances .................................................................................172
11.6. Alternative Means of Minimising Manipulator Disturbances ....................................172
11.7. Design of a Novel Compliant Device ..........................................................................173
11.8. Artificial Neural Network Modelling of the Compliant Device ................................173
11.9. Experimental Conclusions .........................................................................................174
11.10. Summary ..................................................................................................................177
APPENDIX A: TECHNICAL SPECIFICATION FOR THE SLINGSBY ENGINEERING MRV ... 181

APPENDIX B: TECHNICAL SPECIFICATION FOR THE SLINGSBY ENGINEERING A206 MANIPULATOR ................................................................. 183

APPENDIX C: PUMA 560 SUPERVISORY CONTROL SOFTWARE ........................................... 185
  C.1. Supervisory Control Software .................................................. 185
    C.1.1. all_def.h ............................................................................. 185
    C.1.2. track4.cpp ......................................................................... 188
    C.1.3. JCDaq100.h ....................................................................... 193
    C.1.4. JCDaq100.cpp ................................................................... 194
    C.1.5. JCContrl.h ......................................................................... 197
    C.1.6. JCContrl.cpp ..................................................................... 198
    C.1.7. JCAnn.h ............................................................................. 202
    C.1.8. JCAnn.cpp ......................................................................... 203
    C.1.9. JCMatrix.h ......................................................................... 208
    C.1.10. JCMatrix.cpp .................................................................... 209
    C.1.11. JCPuma.h ......................................................................... 213
    C.1.12. JCPuma.cpp ...................................................................... 214

APPENDIX D: CALIBRATION OF THE CAMERA TRACKING SYSTEM ........................................... 223

APPENDIX E: RESOLVED APPLIED LOADS TO THE DOCKED ROV ........................................ 225

APPENDIX F: ROV MOTION SIMULATION PROGRAM ............................................................... 227

APPENDIX G: ATI FORCE/TORQUE TRANSDUCER ................................................................. 229

APPENDIX H: MATLAB MODEL PROGRAMS TO CALCULATE LOCAL STIFFNESS FOR TRANSVERSE SPRING COMPLIANT DEVICE ........................................... 231
  H.1. Without Probe Offset .............................................................. 231
    H.1.1. Program main3.m ............................................................ 231
    H.1.2. Program var_mod3.m ....................................................... 233
    H.1.3. Program normdat.m ......................................................... 236
  H.2. With Probe Offset ................................................................... 237
    H.2.1. Program mainreal.m ....................................................... 237
    H.2.2. Program var_real.m ....................................................... 238
    H.2.3. Program normreal.m ...................................................... 241
    H.2.4. Program plotgraf.m ......................................................... 242

APPENDIX I: ENGINEERING DRAWINGS FOR THE TRANSVERSE SPRING COMPLIANT DEVICE .................................................................... 245

APPENDIX J: ENGINEERING DRAWINGS FOR THE TRANSVERSE POLYMER SPRING COMPLIANT DEVICE AND MOULD ........................................... 247

APPENDIX K: SOFTWARE DEVELOPED TO COLLECT NEURAL NETWORK TRAINING DATA 251
  K.1. PUMA Manipulator Control Programs ..................................... 251
  K.2. Data Logging Software ............................................................ 254

APPENDIX L: MATLAB SOFTWARE USED TO TRAIN THE ARTIFICIAL NEURAL NETWORK 257
  L.1 - PUMA Manipulator Control Programs .................................... 257
APPENDIX M: MATLAB SOFTWARE USED TO CALCULATE RMS ERROR OF THE
ARTIFICIAL NEURAL NETWORK ................................................................. 259

APPENDIX N: MATLAB LINEAR SPRING MODEL ........................................... 261
N.1. Program inv_spr3.m .............................................................................. 261
N.2. Program def_geom.m ........................................................................... 262
N.3. Program converg3.m .......................................................................... 263
N.4. Program sol_vec3.m ............................................................................ 264
N.5. Program trans.m ................................................................................. 265
N.6. Program retrans.m .............................................................................. 266
REFERENCES .................................................................................................. 269
### LIST OF FIGURES

| FIG. 1.1 | SOME TYPICAL OFFSHORE STEEL JACKET DESIGNS [FROM PATEL 89] | 26 |
| FIG. 1.2 (A) | ROV DEPLOYMENT FOR OIL-RIG INSPECTION | 30 |
| FIG. 1.2 (B) | ROV DEPLOYMENT FOR OIL-RIG INSPECTION | 31 |
| FIG. 1.3 | COMPUTER GENERATED IMAGE OF AN ROV DEPLOYED TO INSPECT A WELD | 31 |

| FIG. 2.1 | SIMPLIFIED STIFFNESS AND FORCE CONTROL STRUCTURE | 43 |
| FIG. 3.1 | SCHEMATIC OF TOOL SKID ATTACHED TO ROV | 46 |
| FIG. 3.2 | REAL ROV AND TOOL SKID ASSEMBLY | 46 |
| FIG. 3.3 | TETHER MANAGEMENT SYSTEM ON TOP OF THE ROV (TOP HAT STYLE) | 47 |
| FIG. 3.5 | TYPICAL WELD GEOMETRY FOR AN ACFM PROBE TO INSPECT | 52 |
| FIG. 3.6 | SIMPLIFIED SCHEMATIC OF THE MECHANICS OF COMPLIANT FORCE CONTROL | 53 |

| FIG. 4.1 | PUMA 560 INDUSTRIAL MANIPULATOR, CONTROLLER AND TEACH PENDANT | 58 |
| FIG. 4.2 | PUMA 560 INDUSTRIAL MANIPULATOR JOINTS | 60 |
| FIG. 4.3 | PUMA POSITION AMBIGUITY | 62 |
| FIG. 4.4 | PUMA GLOBAL CO-ORDINATE SYSTEM | 65 |
| FIG. 4.5 | FORCE CONTROL STRUCTURE FOR SURFACE TRACKING | 66 |
| FIG. 4.6 | FLOW DIAGRAM FOR SUPERVISORY CONTROL SOFTWARE | 68 |

| FIG. 5.1 | EXPERIMENTAL LAYOUT - PLAN AND ELEVATION | 72 |
| FIG. 5.2 | MEASUREMENT OF RELATIVE CAMERA POSITIONS | 73 |
| FIG. 5.3 | RAW AND FILTERED DATA FOR IMPULSE LOAD | 74 |
| FIG. 5.4 | ROV FRAME OF REFERENCE FOR APPLIED TEST FORCES | 76 |
| FIG. 5.5 | TEST 22 ROV MOTION LOCUS - SIDE LOAD, ROV LEGS PARALLEL | 77 |
| FIG. 5.6 | TEST 34 ROV MOTION LOCUS - SIDE LOAD, ROV LEGS SPREAD | 77 |
| FIG. 5.7 | ROV RAW DATA TRANSIENT FROM DROP WEIGHT TEST | 78 |
| FIG. 5.8 | ROV RAW DATA TRANSIENT: FILTERED WITH 2HZ BUTTERWORTH | 79 |
| FIG. 5.9 | ROV RAW DATA TRANSIENT: FILTERED 2HZ: FIRST 2 SECS | 79 |
| FIG. 5.10 | ROV RAW DATA TRANSIENT: FILTERED 2HZ AND 1HZ | 80 |
| FIG. 5.11 | MOTION RESPONSE OF LINEAR ROV MODEL TO FORCE "IMPULSE" | 81 |

| FIG. 6.1 | MANIPULATOR, TARGET AND CAMERA SET-UP | 86 |
| FIG. 6.2 | FIXED TARGETS MOUNTED ON A WORKPIECE | 90 |
| FIG. 6.3 | RELATIVE ERROR MEASUREMENT PROBE IN ELEVATION | 91 |
| FIG. 6.4 | VISION EXPERIMENTS DIVIDED BY SYSTEM | 92 |
| FIG. 6.5 | RMS ERROR VS RMS SPEED | 95 |
| FIG. 6.6 | RMS ERRORS VS DESCENDING ORDER OF AVERAGE DISTANCE TRAVELLED | 98 |

| FIG. 7.1 | FOUR DEGREE OF FREEDOM COMPLIANT DEVICE | 102 |
| FIG. 7.2 | LOCATION OF COILS WITHIN THE ACFM PROBE | 103 |
| FIG. 7.3 | SECTION THROUGH BRACE AND CHORD SHOWING KEY DIMENSIONS | 104 |
| FIG. 7.4 | PROBE/COMPLIANT DEVICE/MANIPULATOR ASSEMBLY | 105 |
| FIG. 7.5 | POSSIBLE FORCE/TORQUE TRANSDUCER LOCATIONS | 107 |
| FIG. 7.6 | POSSIBLE LOCATIONS OF ACFM PROBE FORCE TRANSDUCERS | 108 |
| FIG. 7.7 | ACTUAL MOCK PROBE/COMPLIANCE ASSEMBLY | 109 |
| FIG. 7.8 | DIAMOND SHAPED POLYURETHANE COMPLIANT DEVICE | 110 |
TABLE 4.1 - PUMA 560 MANIPULATOR SPECIFICATIONS [UNIMATION 85a].......................59
TABLE 4.2 - PUMA 560 JOINT ANGLE RANGE..................................................................................59
TABLE 4.3 - PUMA 560 LINK LENGTHS..............................................................................................60
TABLE 4.4 - SINGLE CYCLE MESSAGE TRANSFER BETWEEN PUMA AND COMPUTER......64
TABLE 4.5 - SUPERVISORY SOFTWARE CLASS MAP........................................................................69

TABLE 5.1 - MAXIMUM IMPULSIVE AND STATIC LOADS APPLIED DURING ALL TESTS.....76
TABLE 5.2 - MAXIMUM DISPLACEMENTS OCCURRING DURING ALL TESTS.............................76
TABLE 5.3 - MAXIMUM IMPULSIVE AND STATIC LOADS APPLIED DURING TEST 34...........76

TABLE 6.1 - GRADIENT, LAG AND INTRINSIC MEASUREMENT ERROR.................................95

TABLE 7.1 - RELATIVE STIFFNESSES REQUIRED BY THE COMPLIANT DEVICE...............114
TABLE 7.2 - NORMALISED RELATIVE STIFFNESS OF TRANSLATIONAL COMPLIANT
DEVICE AXES WITH VARYING GEOMETRY COMPLIANT ELEMENTS.................................117
TABLE 7.3 - NORMALISED RELATIVE STIFFNESS OF ROTATIONAL COMPLIANT DEVICE
AXES WITH VARYING GEOMETRY COMPLIANT ELEMENTS...............................................117
TABLE 7.4 - VARIATION OF LOCAL STIFFNESS AS TRANSLATION IN Y AND ROTATION IN X
TAKE PLACE FOR A RANGE OF INNER AND OUTER PLATE GEOMETRIES......................121
TABLE 7.5 - VARIATION OF LOCAL STIFFNESS AS TRANSLATION IN Y AND ROTATION IN X
TAKE PLACE FOR A RANGE OF INNER AND OUTER PLATE GEOMETRIES......................122

TABLE 8.1 - PRINCIPAL ARTIFICIAL NEURAL NETWORK TYPES..............................................128
TABLE 8.2 - TRANSLATIONAL AND ROTATIONAL MOTION RANGE USED FOR DATA
GATHERING FOR NEURAL NETWORK.........................................................................................132
TABLE 8.3 - ABSOLUTE MAXIMUM ERROR AND RMS ERROR FROM
APPLICATION OF THE ARTIFICIAL NEURAL NETWORK TO THE TRAINING
AND THE VERIFICATION DATA SET.........................................................................................139
TABLE 8.4 - MEAN AND STANDARD DEVIATION OF THE MAGNITUDE OF PROBE TOE AND
HEEL POSITION ERRORS........................................................................................................141
TABLE 8.5 - RELATIVE ERROR OF THE SPRING AND NEURAL NETWORK MODELS OF THE
COMPLIANT DEVICE................................................................................................................142
TABLE 8.6 - MEAN AND STANDARD DEVIATION OF THE MAGNITUDE OF PROBE TOE AND
HEEL POSITION ERRORS........................................................................................................142

TABLE 10.1 - TOE AND HEEL CO-ORDINATES AT THE START AND END OF TRACKING...161
TABLE 10.2 - TRUE AND MEASURED LENGTHS BETWEEN HEELS AND TOE.....................161
TABLE 10.3 - COMPONENT AND COMBINED ERROR OF CALCULATED PROBE POSITION 163

TABLE B.1 - SLINGSBY A206 MANIPULATOR GENERAL SPECIFICATIONS ....................183
TABLE B.2 - SLINGSBY A206 MANIPULATOR GEOMETRY SPECIFICATIONS..................183
TABLE B.3 - ACCURACY AND REPEATABILITY OF THE SLINGSBY A206 MANIPULATOR 184

TABLE E.1 - MAXIMUM IMPULSIVE LOAD APPLIED DURING IMPULSE TESTS...............225
TABLE E.2 - FINAL STATIC LOAD APPLIED DURING IMPULSE TESTS.................................225
TABLE E.3 - LOADS APPLIED DURING STATIC TESTS.........................................................226
### LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACFM</td>
<td>Alternating Current Field Measurement (crack detection probe)</td>
</tr>
<tr>
<td>ADC</td>
<td>Analogue to Digital Converter</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>ARM</td>
<td>Automated Remote Manipulator</td>
</tr>
<tr>
<td>BASIC</td>
<td>A simple computer language</td>
</tr>
<tr>
<td>CCTV</td>
<td>Closed Circuit TeleVision</td>
</tr>
<tr>
<td>CD</td>
<td>Compliant Device</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal Oxide Semiconductor</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital to Analogue Converter</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>DDCMP</td>
<td>Digital Data Communications Message Protocol (PUMA)</td>
</tr>
<tr>
<td>DEC LSI-11</td>
<td>A computer on a card forming part of PUMA controller</td>
</tr>
<tr>
<td>dof</td>
<td>degree of freedom</td>
</tr>
<tr>
<td>EPROM</td>
<td>Erasable Programmable Read Only Memory</td>
</tr>
<tr>
<td>LVDT</td>
<td>Linear Variable Differential Transducer</td>
</tr>
<tr>
<td>MRV</td>
<td>Multi-Role Vehicle (type of ROV)</td>
</tr>
<tr>
<td>OAT</td>
<td>Orientation, Attitude and Twist (angles defining PUMA orientation)</td>
</tr>
<tr>
<td>PAL</td>
<td>European TV transmission protocol</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional, Integral and Differential controller</td>
</tr>
<tr>
<td>PUMA</td>
<td>Programmable Universal Machine for Assembly (manipulator arm)</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RCC</td>
<td>Remote Centre Compliance</td>
</tr>
<tr>
<td>RMS</td>
<td>Root of the Mean of the Square</td>
</tr>
<tr>
<td>ROV</td>
<td>Remotely Operated Vehicle</td>
</tr>
<tr>
<td>RS 232 (c)</td>
<td>A standard serial communication protocol</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>TMS</td>
<td>Tether Management System</td>
</tr>
<tr>
<td>TSC</td>
<td>Technical Software Consultants (Software company)</td>
</tr>
<tr>
<td>VAL</td>
<td>Variable Assembly Language (control language for PUMA)</td>
</tr>
<tr>
<td>VAL II</td>
<td>Variable Assembly Language (version 2)</td>
</tr>
<tr>
<td>VDU</td>
<td>Visual Display Unit</td>
</tr>
<tr>
<td>wd</td>
<td>water depth</td>
</tr>
</tbody>
</table>

In this thesis, variables are denoted by italic symbols, physical quantities by normal type and matrices or vectors by bold type.

**Variables:**

<table>
<thead>
<tr>
<th>symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \nu )</td>
<td>kinematic viscosity (of seawater) (m(^2)/s)</td>
</tr>
<tr>
<td>( \phi )</td>
<td>phase angle (deg)</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>minimisation function</td>
</tr>
<tr>
<td>( \theta )</td>
<td>chord/brace internal angle (deg)</td>
</tr>
<tr>
<td>( \alpha, \beta, \varphi )</td>
<td>Euler rotation angles (deg)</td>
</tr>
<tr>
<td>( \omega_x )</td>
<td>frequency of motion along a cartesian axis (( x ) or ( y )) (rad/s)</td>
</tr>
<tr>
<td>( \tau_x )</td>
<td>period of motion along a cartesian axis (( x ) or ( y )) (s)</td>
</tr>
<tr>
<td>( A_x )</td>
<td>amplitude of motion along a cartesian axis (( x ) or ( y )) (mm)</td>
</tr>
<tr>
<td>( a_s )</td>
<td>rotation matrix element</td>
</tr>
<tr>
<td>( B_{1n} )</td>
<td>bias vector element</td>
</tr>
<tr>
<td>( B_{2n} )</td>
<td>bias vector element</td>
</tr>
<tr>
<td>( d )</td>
<td>cylinder diameter (m)</td>
</tr>
<tr>
<td>( D )</td>
<td>constant</td>
</tr>
<tr>
<td>( e_i )</td>
<td>( i )(_{th} ) calculated error (mm)</td>
</tr>
<tr>
<td>( f )</td>
<td>vortex shedding frequency (Hz)</td>
</tr>
<tr>
<td>( F )</td>
<td>force (N)</td>
</tr>
<tr>
<td>( f_i )</td>
<td>( i )(_{th} ) recorded error (mm)</td>
</tr>
<tr>
<td>( F_x )</td>
<td>force acting along a cartesian axis (( x, y ) or ( z )) (N)</td>
</tr>
<tr>
<td>( h )</td>
<td>weld access height (mm)</td>
</tr>
<tr>
<td>( L )</td>
<td>weld depth (mm)</td>
</tr>
</tbody>
</table>
\( l \) chord wall thickness (mm)

\( n_x \) rotation matrix element

\( O, A, T \) Euler angles defining manipulator orientation (deg)

\( O_m \) output vector element (position or orientation in mm or deg)

\( P_i \) input vector element (force or torque in N or Nm)

\( R_n \) intermediate solution vector element

\( s_x \) rotation matrix element

\( t \) time (s)

\( T \) total time duration of a test (s)

\( T_x \) torque acting about a cartesian axis (x, y or z) (Nm)

\( V \) fluid velocity (m/s)

\( V_x \) velocity of motion along a cartesian axis (x or y) (mm/s)

\( W_{1_{i},i} \) weight vector element

\( W_{2_{m,i}} \) weight vector element

\( x, y, z \) cartesian co-ordinate axes

\( X, Y, Z \) cartesian co-ordinate axes

**Vectors and Matrices:**

<table>
<thead>
<tr>
<th>symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>bias vector</td>
</tr>
<tr>
<td>B2</td>
<td>bias vector</td>
</tr>
<tr>
<td>F</td>
<td>force and torque (N and Nm)</td>
</tr>
<tr>
<td>F_{a}</td>
<td>actual force and torque (N and Nm)</td>
</tr>
<tr>
<td>F_{d}</td>
<td>desired force and torque (N and Nm)</td>
</tr>
<tr>
<td>F_{e}</td>
<td>error force and torque (N and Nm)</td>
</tr>
<tr>
<td>I</td>
<td>identity matrix</td>
</tr>
<tr>
<td>K</td>
<td>stiffness (N/mm and Nm/deg)</td>
</tr>
<tr>
<td>O</td>
<td>output vector</td>
</tr>
<tr>
<td>p</td>
<td>point in cartesian space</td>
</tr>
<tr>
<td>P</td>
<td>position and orientation (mm and deg)</td>
</tr>
<tr>
<td>P</td>
<td>input vector</td>
</tr>
<tr>
<td>R</td>
<td>intermediate solution</td>
</tr>
<tr>
<td>R</td>
<td>rotation matrix</td>
</tr>
</tbody>
</table>
true probe position data (mm and deg)
transformation
calculated probe position data (mm and deg)
constant transformation
velocity (mm/s)
vector
constant vector
weight vector
weight vector
actual position (mm)
desired position (mm)
desired position for required force control (mm)
desired position for required tracking (mm)

Physical Quantities:

<table>
<thead>
<tr>
<th>symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>°</td>
<td>degrees</td>
</tr>
<tr>
<td>cm</td>
<td>centimetre</td>
</tr>
<tr>
<td>Hz</td>
<td>hertz (frequency)</td>
</tr>
<tr>
<td>kg</td>
<td>kilogram</td>
</tr>
<tr>
<td>K</td>
<td>( x 10^3 )</td>
</tr>
<tr>
<td>m</td>
<td>metre</td>
</tr>
<tr>
<td>mm</td>
<td>millimetre</td>
</tr>
<tr>
<td>ms</td>
<td>millisecond</td>
</tr>
<tr>
<td>ms(^{-1})</td>
<td>velocity</td>
</tr>
<tr>
<td>ms(^{-2})</td>
<td>acceleration</td>
</tr>
<tr>
<td>M</td>
<td>( x 10^6 )</td>
</tr>
<tr>
<td>N</td>
<td>newton</td>
</tr>
<tr>
<td>Nm</td>
<td>newton-metre</td>
</tr>
<tr>
<td>Re</td>
<td>reynolds number</td>
</tr>
<tr>
<td>s</td>
<td>second</td>
</tr>
</tbody>
</table>
1.1. The Requirement for Oil

For more than a century, oil has provided cheap and easy to use energy for domestic and industrial operations. An abundance of affordable energy is critical for any activity to grow and so the economic extraction of oil from sites the world over has been an ever growing demand to fuel the growth of industries and economies. Oil was first discovered and extracted from land based wells but, at the end of the last century, offshore oil extraction began. This activity is still growing today with the technological and geographical boundaries continually being pushed back by technology and demand.

With these advanced technologies come serious engineering problems and this thesis examines new ways to tackle one of the key problems on steel offshore oil rigs: How do you detect fatigue cracks on the oil rig structure?

1.2. Getting Oil From An Offshore Oil Field

The first significant offshore drilling operation was off the California coast in 1887. The problem in all oil exploration and recovery is to be able to guide a drill into the ground to cut a well. If oil or gas is discovered, it is necessary to control the extraction of these materials and transport them on to their next port of call such as a refinery. To this end, a vast range of structures (known as jackets) have been built and fixed to the seabed in the last century to act as platforms from which the drilling and pumping operations can take place. Figure 1.1 shows some typical offshore structures with their year of manufacture and the depth of water in which they operate.

The early jackets were built of wood, but this quickly gave way to steel to give stiffer and more corrosion resistant structures for deeper water. Steel reinforced concrete is also used but the majority of new jackets are steel as these are significantly cheaper. This is because the concrete structures still require almost the same mass of
steel, as reinforcement, as well as the enormous tonnage of concrete required [Patel 89]. The advantage of concrete structures is that the concrete itself protects the steel from the corrosive effects of sea water.

Since the first offshore jacket, the number of oil and gas jackets and the depth of water in which they work has continually increased. In 1975, technology reached a level sufficient for the first jacket to be placed in the deep and extremely inhospitable waters of the North Sea. Now there are about 180 fixed installations in the UK sector alone and over 50 of these are more than 15 years old [Stacey 95]. These are generally found in depths of up to 120 metres although the European oil search is now moving into the even deeper waters of the North Atlantic [Edwards 95].

1.3. The Problem: Cracks Forming due to Fatigue and Corrosion of an Offshore Jacket

It is well established that cracks in a material can grow slowly, at stresses well below a material's yield strength, if the component in question is in a corrosive environment or if it is subjected to repeated stress cycles [Ashby 83]. In large structures, such as bridges or jackets, cracks always exist to some degree. If they are left to grow, as they will under continued loading, these cracks can result in a catastrophic failure called a fatigue failure.
Once placed at its drilling site, each of these jackets undergoes repeated cyclic loading and considerable static loading over its lifetime due to wind, waves and sea currents. Of these, by far the largest forces are due to waves [Patel 89].

In addition, as novel drilling techniques are developed enabling new oil reserves to be reached from the same platform, they are often required to stay in service for longer than was originally intended thus increasing the exposure to stress cycles and corrosion.

Finally, as the height of jackets increase, their natural, or resonant, frequencies of vibration decrease and approach more closely the frequencies of waves. This increases the effective stresses and further increases the likelihood of a catastrophic fatigue failure.

These three factors create a perfect situation for the development of fatigue cracks.

Research shows that the stress concentrations and thus the highest rate of crack propagation, occurs within the weld material joining intersecting steel cylinders that make up the jacket. These are found at nodes; the name given to the section of a main supporting tubular member where one or more other tubular members are joined by welding. There is, therefore, a requirement from the operators and their insurance companies for regular weld inspections of the jacket to ensure that growing fatigue cracks may be detected and repaired in good time.

1.4. The Current Solution: Detection and Repair of a Fatigue Crack Using Divers

Currently crack detection and repair is achieved using divers who deploy hand held probes to detect cracks. A typical mission profile for a crack detection task at a depth of 100 metres might be as follows:

1. On the surface vessel, the divers enter a compression chamber. They are then pressurised to the pressure of their working depth at which they will remain for several weeks, a process known as saturation diving. During any diving task, gases (particularly nitrogen) dissolve in the bloodstream at the elevated pressures underwater. If a diver moves towards the surface too quickly, the rapid pressure drop causes nitrogen to re-appear as gas bubbles in the bloodstream. This causes great pain and is extremely dangerous and so, to avoid it, divers return to the surface slowly allowing the nitrogen to separate at a low enough speed for the body to get rid of it harmlessly through the lungs. At depths below 36 metres (pressure equivalent to about 3.6 atmospheres), the decompression times required after even a
short dive are too long for effective work to be carried out. Thus for deep work, the divers are pre-pressurised to their working depth in a mobile high pressure (hyperbaric) chamber. They will remain at their working pressure for several weeks as they are lowered, in the chamber, to various work sites where they can work shifts; working, resting and sleeping at the elevated pressure. On completion of their work period, they are brought to the surface and decompressed at an appropriate speed. This is an extremely slow and expensive process.

2. The pressure chamber will be placed near the work site. On the surface there will be a support vessel and crew present throughout this work period. Initially divers will be required to swim around looking for visual clues to confirm that they are at the right node. They are helped to do this by the surface support team using cameras.

3. The first task is to clean the weld area. This is generally done by grit blasting and, depending on the type of marine growth, may take several hours to do. A clean surface is required 50 mm either side of the weld. The final surface quality required depends on the inspection method to be used.

4. The diver will now mark out the node being inspected to have some sort of metric as to where any detected cracks are in space. This will be done with a centre punch to mark some of the cardinal clock positions around the node. i.e. 12 O’clock.

5. The diver will then take a probe to the node being inspected. e.g. an alternating current field measurement (ACFM) probe that measures the magnetic field intensity induced in the steel surface by an alternating current. The probe placement is critical in terms of surface lift off and alignment and so this is a time consuming and arduous task. It is also difficult to record exactly where, on a large node, the crack was detected. The actual detection interpretation equipment will be remote from the diver and so they are relying on intercom communications. The normal practise is to mark the crack with a crayon and then photograph a tape measure stretched from the cardinal mark to the crack start and finish.

6. The diver may then remove any equipment used and return to the compression chamber ready to be moved to the next inspection site.

It may be seen that this is a time consuming and expensive task. An additional problem is the medical effects on divers. Disorientation is a common problem leading to slow identification of the divers position in space. It is further likely that, as medical
knowledge increases and the long term effects are more clearly identified, the health and safety limits on where and for how long divers can work will be reduced. Norway is already taking the lead with maximum depths set at 200 metres.

The crack position measurement techniques in particular, are slow and cumbersome but vital to enable a follow up team to grind out and repair the located crack.

In summary, there is a very significant danger and a high cost incurred using divers for sub-sea inspection tasks. These may both be substantially reduced by unmanned intervention. Also, there are an ever growing number of complex inspection tasks that will require machine intervention if they are to be performed at all as they are beyond the depths that divers can reach.

1.5. The Robotic Solution: Detection and Repair of a Fatigue Crack Using Unmanned Submersible Vehicles Carrying Robotic Arms

To enable continued maintenance and repair in ever deeper water and to reduce the hazard to divers, there is a trend toward the use of robotic arms (manipulators) mounted on remotely operated vehicles (ROV's).

A typical overall system is shown in figure 1.2 (a and b). The contents of these drawings may best be explained by considering the mission profile of an inspection operation using an ROV:

1. The ROV and tether management system are lowered straight down from the deck of the oil rig using the heavy duty main lift umbilical cable. This cable contains all communications and power transmission to and from the ROV. At the correct depth, the ROV is flown away towing a lighter, neutrally buoyant, umbilical cable that unreels from the tether management system. This system is used because, to fly a mission at any great distance from the deployment crane, an enormous load due to cable would be imposed on the ROV if a single umbilical cable was used. This load arises from drag induced by tidal currents.
2. The ROV deployment area comprises a cabin containing an operator who 'flies' the vehicle to the required node. The operator knows the structural layout of the jacket and uses depth and heading sensors and visual cues from video cameras mounted on the ROV to ensure that he reaches the correct target.

3. Once the node is reached, three legs with vacuum 'sticky' feet are extended and attached to tubular steel members to give a stiff working platform. These legs may be moved independently. When attached to the jacket, adjusting these legs can be used as a method of altering the position and orientation of the ROV without undocking.

4. At this point, the manipulator arm confirms the position and orientation of the node by touching the node at a range of points with a contact sensitive switch.

5. When the node position is confirmed, the manipulator is used to clean the weld using a grit blasting tool.
6. Finally, force feedback is used to enable a probe to be moved around the weld. A high level graphical control system is used to control the manipulator. Figure 1.3 shows a typical screen image from the Automated Remote Manipulator (ARM) controller that gives a rendered image of where the ROV, manipulator and jacket are in space [Langrock 94].
7. The position of any crack detected is automatically recorded in a world model to a high accuracy.

8. When complete, the ROV can leave the node and fly to any other node at any other depth to continue inspection work.

1.6. Summary of Missions

There are several key areas in which the two missions may be usefully compared:

1. Cost. This is critical for any business venture. As the tendency is for extraction to take place from smaller fields in more difficult conditions, it is imperative to save costs wherever possible. For diving, a surface support vessel is required that costs tens of thousands of pounds every day. A remote ROV intervention uses fewer personnel directly and so, if a task can be done remotely, the cost savings are considerable.

2. Safety. The risk to divers lives is ever present and the long term health risks of saturation diving are starting to be realised as severe. Using remote intervention techniques, there is no direct risk of personnel. Safety, of course, has a further impact on cost as there are considerable legal liabilities for employers in the face of injury to their employees. Surface operators of ROV's can work shifts to keep the ROV working continuously. To work diving shifts requires many more divers in saturation with a far greater unit personnel cost. In brief, if the task can be done remotely, the safety to personnel is improved.

3. Speed. The inspection by divers is a slow task made worse by the ‘dead’ time in decompression afterwards. ROV inspection is even slower, but this is not a serious drawback as it is still much cheaper. In addition, the ROV can rapidly deploy to a new working depth, in either direction, without coming to harm due to pressure changes. This is not the case for a diving team. In brief, if the task can be done remotely, it is likely to be as fast as a diver based inspection and the changeover to new tasks is likely to be considerably quicker.

4. Accuracy. As was described earlier, the measurement of the precise position of a crack is a rather laborious task for a diver. By the nature of a robotic system, the probe position is continually measured and so it is straightforward procedure to record this whenever a crack is detected. This has the result that the reliability and
repeatability of an inspection may be improved. In brief, if the task can be done remotely, it is likely to be more accurate.

5. Depth. Currently, divers are limited to maximum depths of about 300m and that is being reduced by safety standards. The ROV described in chapter three of this thesis is rated for 1000m. In brief, at depths greater than 300m, only the unmanned solution will work.

1.7. Conclusions

In conclusion, if the task can be done remotely it will be faster, cheaper, safer, more accurate and deeper. This thesis is concerned with making the remote inspection automatic once the target has been reached and the ROV attached to the jacket. The title “Compliant Force Control for Automated Sub-Sea Inspection” refers to the use of a sprung tool to enable the weld tracking to be automated.

As a measure of success for this thesis, the following key tasks will need to be achieved:

1. The motions of an ROV attached to a sub-sea structure must be estimated.

2. An effective design for a compliant device must be developed giving the required stiffness in different degrees of freedom.

3. A means of establishing the probe position from contact force data must be developed to allow motion commands to be given in the probe co-ordinate frame (local modelling) reducing force disturbances due to motion into the workpiece. The desired accuracy is to have the commanded direction within 2° of true direction.

4. A means of globally mapping where the crack is on the weld is required. The desired overall accuracy is ±10mm.

1.8. Summary of Chapters in the Ph.D.

The automated inspection of sub-sea welds is a major task and this work deals with specific areas within it. These are (by chapters):

1. The introduction.

2. Robotics and Force Control. A survey of the current state of the art and its particular relevance to this application.
3. Problem Definition. This gives detail on the hardware (ROV and manipulator) in the real sub-sea application and on the stability of the ROV as a working platform. It describes the geometrical constraints, the nature of the required contact between the probe and the weld under inspection, and the effect of compliance on this contact.

4. Supervisory Control of the PUMA 560. This describes the experimental manipulator used to prove this thesis and the software developed to control it.

5. ROV motion. This describes an experiment to measure the motion of the ROV when subjected to loads, whilst docked onto the oil-rig structure.

6. Motion Compensation. This describes work done testing a stereo vision system to enable the manipulator to compensate for a moving working platform. It is known that the ROV can still move when attached to the jacket due to wave motion, sea currents and reaction from the manipulator contact forces.

7. Design and Implementation of the Compliant Device. This describes the design process that delivered a complex spring construction (the compliant device), to sit between the manipulator and the probe, which was able to give an appropriate stiffness along different axes.

8. Artificial Neural Networks used to Model the Compliant Device. This outlines the choice of this non-linear modelling tool, the development work that went into creating the real application and the verification of the tool as sufficiently accurate for the task.

9. Kinematics. This describes the task kinematics used to define the motion commands of the manipulator.

10. Use of the Compliant Device Model to Enable Compliant Force Control. This details the experimental method by which the Artificial Neural Network is used to enable real time control of the manipulator and surface mapping of the workpiece.

11. Conclusions from the thesis work.

In summary, this work has succeeded in simplifying the modelling of a highly complex compliant shape so that position data could be accurately extracted from force data at a rate of thirty six times a second. Furthermore, it has incorporated that model into a practical surface following algorithm for a manipulator deployed probe.
2.1. Background to Robotics

A robot is defined in the Collins Concise Dictionary as "Any automated machine programmed to perform specific mechanical functions in the manner of a man". A more useful paraphrase of this, applied to a modern industrial robot, is that a robot is a re-programmable, general purpose manipulator that can be applied to many different tasks.

A manipulator consists of a chain of several rigid links joined by revolute or prismatic joints. One end is anchored to a base whilst the other end may hold a tool to perform an assigned task. By controlling the position or angle of all the joints, relative motion of the individual links takes place and thus the position and angle of the tool may be controlled.

For the robot to be re-programmable implies some level of computer supplied 'intelligence'. In industry this has the great significance that a single machine may perform a large number of different tasks without expensive hardware alterations.

The real beginnings of robotics and, in particular, manipulators, were in the development of master-slave manipulators for handling radioactive materials in the 1940's. The first commercial computer-controlled robot manipulator was introduced by Unimation in the 1950's. More recently, the real push towards the development of commercial robots was to improve productivity in industrialised nations.

The early manipulators were solely position controlled and relied on their tools or materials being placed in precisely the correct position in order for a task to be correctly executed. If this was not the case then collisions between the manipulator and the materials could occur. The position errors can arise from a number of sources. The main ones are manipulator inaccuracy, workpiece location errors and disturbances (for example, a powerful rotating tool will exert a force on the manipulator as it comes into contact with a surface). Position control is still extremely common in the 1990's and many industrial applications, such as spray painting, are non-contacting positional tasks.
Such tasks assume that the environment is known and thus, if the manipulator can move
with the required repeatability to a pre-defined position and orientation, it may be
assumed that the sprayed paint will reach the workpiece as required. This type of task
has been achieved successfully since such manipulators were first developed.

Similarly, simple contact tasks may be achieved using only position control of a
manipulator. For example, automated spot welding involves contact between a
workpiece and the electrode which is held by the manipulator. The contact forces can
reasonably be disregarded from a control sense provided that the repeatability of
positioning of the end effector and the workpiece is high and that the manipulator or
workpiece has sufficient passive compliance such that one or the other will give as a
result of excessive contact force. Obviously, the greater the compliance used to absorb
the dimensional errors the greater the degradation in the precision of the manipulator
tool position. Compliance may be defined as the inverse of stiffness.

There are many situations in which pure position control is insufficient. For
example, if the position of a manufactured component lying on a production line were to
vary from one component to the next, then a manipulator applying adhesive to a
particular area may make errors resulting in costly failures at a later stage of production.
Similarly, if a component is to be press fitted into an aperture and the aperture is not
where the manipulator expects it to be, both parts may be damaged by the contact
pressure. To address these types of problem, in the 1960’s, researchers began
experimenting with a wide array of external sensory devices using media such as vision,
pressure, force and sound [Ernst 62] [Tomovic 62] [McCarthy 68]. In the 1970’s the
pace of research stepped up and the focus was heavily on the use of external sensors to
augment and facilitate the use of manipulators. This is still the case today.

2.2. Force Control

2.2.1. The Need for Force Control

The need for active robot force control arises in many situations. The following list
highlights some of the more typical ones:

- An unstructured environment where a manipulator must feel its way around. e.g. a
  sub-sea environment where poor visibility precludes the extensive use of vision
  systems.
• A controlled contact force task must be undertaken. e.g. grinding or polishing.
• Assembly tasks where components may jam. In fact this may also be achieved by a passive remote centre compliance (RCC) for some special cases [Whitney 86].

These may all be considered as tasks where it is necessary to eliminate positional uncertainty whether arising from the lack of accuracy of manipulators or from a less than perfect world model. All these problems have proved difficult to solve and are still very current.

At the level of a manipulator and workpiece in contact, some sort of force feedback is vital to limit the development of enormous contact forces. Currently, the most widely used form of sensory feedback is vision. This can be extremely effective for scene interpretation and task planning and is generally used as a means of improving the world model. Vision systems could be used to sense force but only indirectly by seeing the deflection of objects. If the objects are very stiff (steel tubes supporting an oil rig for example) then it will take enormous and damaging forces before a deflection is visible. Vision is, therefore, rarely used in this capacity.

Control methodologies tend to be a trade off between accuracy/cost and speed. For example, an early hybrid position-force controller was implemented in cartesian space and was at the very limits of their computational power [Raibert 81]. Simplifications of their techniques were necessary to achieve usable solutions [Zhang 85]. Nowadays, computer speeds have increased so enormously that much larger computational burdens may be taken on as standard; the trade off between cost and speed still exists but the goal posts have moved.

2.2.2. Types of Force Control
As stated by Whitney in his 1987 review paper, most control analyses deal with manipulators moving in free space and, in general, focus on strategy rather than control issues [Whitney 87]. As a result of this, the basic ideas required for force control were relatively late appearing. The ideas covered in that review summarise the following basic force control techniques (These may sometimes be called by other names in other papers and the distinctions between them can become blurred with some variants):
• Stiffness control - sensed forces give rise to position modifications.
• Damping control - sensed forces give rise to velocity modifications.

• Impedance control - sensed forces give rise to position and velocity modifications [Hogan 85].

• Force control - has a desired force as the input reference rather than a desired position.

• Hybrid control - a hybrid position and force control method in which orthogonal axes may be controlled by one or the other [Raibert 81].

• Implicit force feedback - no sensor is used. The joint servos are adjusted to give a particular stiffness matrix for the manipulator.

Underlying many practical force control schemes is the strategy of splitting the controlled space into orthogonal position and force controlled directions, listed above as hybrid force control [Mason 81]. Mason's ideas basically state that, for directions in which the motion of the manipulator is physically constrained, a motion in that direction will generate contact forces and therefore those contact forces may be used as the controlling feedback data. For motions in unconstrained directions, no forces are generated and so the position of the manipulator is the controlling feedback data instead. Throughout the mid-eighties, a great deal of work was put into hybrid position/force control [Backes 85] [Wu 82] [Leininger 84] [Chung 84]. However, there is growing opposition to the pure application of this idea. It has been shown that the simple orthogonality of hybrid force control is highly dependant upon the units and the axes used and may yield unstable results [Lipkin 88] [Duffy 89] [Kerr 96] [Hunter 96]. Duffy shows that there is a symmetric bilinear form that can be used, based on twists and wrenches, and so the idea is not lost but merely refined. Recent hybrid force/position control systems and adaptive variations of these have been successfully applied to hydraulic underwater manipulators [Dunnigan 96] [Lane 97].

More complex methods of applying these same hybrid control methodologies were developed for the case where a constraint may lie anywhere along a manipulator [West 85] rather than just at the end.

Kazerooni et al developed a theory in which specifications in the frequency domain ensure the desired compliant motion in the presence of bounded uncertainties [Kazerooni 86a].
The operational space formulation uses generalised task specification matrices for force/position control. These enable the dynamic coupling between different dofs to be described and so improve dynamic performance over joint space dynamic models [Khatib 87].

2.2.3. Key Aspects of Force Control

Some of the key considerations for any force control system are:

1. Adaptation - An important class of problem is that in which, as the environment or task changes, the nature of the contact force changes and so some form of adaptive response is implemented to enable the manipulator to respond with a near optimal dynamic response. These commonly use some form of real-time parameter estimation for a simple, local process model [Zhou 91] [Wang 91].

2. Logic Branching or Continuous Feedback. [Whitney 87]. A logic branching feedback system is of the variety that states: first move in \( x \) until \( F_x > \text{limit} 1 \). Then twist about \( z \) until \( T_z < \text{limit} 6 \) etc. A continuous feedback system typically takes data from a six axis sensor and creates simultaneous six axis motion commands. This is a primarily dynamic strategy whereas logic branching is primarily static.

3. Model based or Performance based Control. An extremely common and effective approach to a control problem is that of forming a mathematical model of the system under consideration. A control system may then be designed, using the model, to meet certain design criteria and subsequently be applied to the real system. Sufficient and relevant modelling is extremely important. Certain primary dynamic limitations of robotic systems have been pin-pointed [Eppinger 89]. In addition, a common problem is that, although a control system may deal adequately with the (usually) rigid body model, higher modes of vibration in the real system will often still cause instabilities [Eppinger 87].

Performance based controllers assume that an accurate dynamic model is not available, for example, the case of a rapidly varying payload. These controllers include sliding mode control, robust control and adaptive control.

In recent years artificial neural networks have achieved success as a method of modelling non-linear systems [Venkatamaran 93] [Horne 89] [Colina 92] [Elsharkawe 91]. The basic characteristics of artificial neural networks as system
models are that they learn, require careful preparation of input and target data and that they tend to provide an output that is statistically excellent, but is never numerically precise in the way that a classical model would be. i.e. a noisy dynamic output is the norm. For example, artificial neural networks have been successfully used to extract stress information from the noisy and ill-posed strain data of a tactile sensor [Pati 92]. As is generally the case, a range of smoothing and conditioning functions were applied to the data before it could be usefully applied to the learning network. The data pre-processing is often the most time consuming element of using a neural network and is still very much a 'black art'.

Effectively, most artificial neural networks act as a ‘black box’. In all but the simplest cases, the resulting trained network cannot be proven to always give an output within the correct region. As there is always an element of doubt as to the validity of an output they are potentially dangerous to use in safety critical systems. Despite this, artificial neural networks have been used in real time control structures [Irwin 95] [Tao 93]. Even in the case of the more understandable and tractable fuzzy logic, real applications will bound the output by using definite software limits that effectively filter out unacceptable results [Buckman 96].

4. Making Contact With the Environment. In most force control problems, there are two key areas that need addressing. These may be separated into making contact and maintaining stable contact. In terms of making contact, prior knowledge of the surface to be contacted may be used to good advantage to pre-shape the feed-forward signal [Hyde 93]. Various clever local schemes to limit or pre-shape particular values have been successfully applied [Wilfinger 93]. Even such a simple modification as capping an error signal can eliminate ‘bounce’ on contact. In all cases, the final desired result is to achieve a stable, continuous surface force. It may be that some initial ‘bounce’ or ‘chatter’ is acceptable [Marth 93] if it is bounded.

5. Active or Passive force/motion interactions (compliance). The active case is where forces arise from energy supplied to actuators and the supply actuation varies according to the measured forces. Passive interaction arises from deformation of elastic bodies and the actuator inputs are not altered as a result. An example of an active (programmed) compliance is shared compliance control [Kim 90] in which force data is low pass filtered before being used to adjust the end effector motion. This has the same effect as having a damped spring (a real passive compliance)
attached to the manipulator. This has found particular use in tele-robotics where the technique of using force reflection to enable the operator to 'feel' when contact is made brings significant stability problems.

6. Stiff (hard) or Compliant (soft) Contact. When maintaining stable contact between a manipulator and the environment, the key division is between stiff and compliant contact. The exact definition of the dividing stiffness varies according to the source. A sensible division between hard and soft contact is that a contact is soft if the tool deflection is measurable and greater than the positioning accuracy of the manipulator [Zhou 90]. Any deflection of lesser magnitude would then be considered a hard contact. The practical outcome of this choice is a trade off between high bandwidth stiff contact to give precise positioning or the less precise compliant contact. The stiff (high bandwidth) solution requires more rapid sampling of data as any given manipulator motion against a rigid environment will result in higher forces than for the compliant contact and so it must respond more rapidly. Basically, as the sampling rate drops so the end effector velocity must drop [Goddard 92].

This thesis is concerned entirely with compliant contact. As explained in chapter one, a fast manipulator response is not a high priority requirement in this application and so lower computational performances are acceptable. This means that it is possible to use slower, and hence cheaper, hardware.

In general, most compliant systems are passive. That is, the compliance of any element is determined only by its intrinsic spring stiffness and no actuators are employed to artificially alter its effective stiffness. Possibly the best known passive system is the Remote Centre Compliance (RCC) [Whitney 86], which actually rotates and translates (is compliant) about a virtual centre and so contact forces work to align the workpiece with the task instead of simply pushing the workpiece out of line. Compliant systems have also been used to identify surface properties through touch [Sinha 93] [Tsujimura 88] [Fearing 91].

There is a considerable body of work that demonstrates the trade off between compliance and feedback gain [Whitney 87] [Roberts 85]. To be able to use a high control gain and thus have a reasonable performance, requires a low environment stiffness. This can be most easily guaranteed by using a compliant device between the manipulator and the environment and considerable research into the stability of such a system has been carried out [Waibel 91] [Whitney 87]. Using an arbitrarily small time step, Ts, cannot solve the problem as the fundamental limits are set by the
manipulator bandwidth. That is to say that the elements of the manipulator with the lowest natural frequency will define the speed at which the manipulator can work. One problem arising from such a passive compliance is the difficulty of knowing precisely where the contacted surface is, relative to the manipulator, since the passive compliance will have deformed. This was addressed in a very simplistic one dof manner [Roberts 85] in which the measured force was used to calculate the deflection of a beam (the force transducer was a beam with a strain gauge on it) and the manipulator was then moved to compensate for it. A full six dof compliant device used a complex six jointed geometry with each joint angle measured to enable the environment position to be known [Sinha 93]. This enabled the surface to be mapped but was bulky and vulnerable. One of the key problems solved in this thesis was to produce a compliant device with which the surface could be accurately measured and which was both compact and robust.

Stiff contact systems have been slower developing, largely due to the cost and performance limitations mentioned above. Active damping has been one of the successful methodologies [Quian 92]. Before this stable stiff contact was achieved by direct torque control [An 87] and by hybrid impedance control [Liu 91]. Early work in the field was conducted at the Jet Propulsion Laboratory [Craig 79].

Precision, rather than accuracy or repeatability, is one of the key elements for stiff control as very small motions can generate large contact forces. By precision is meant the ability of the manipulator to make a distinct motion down to a lowest dimensional unit. Indeed, some grinding solutions have used additional, very precise actuators, attached to the manipulator end effector to move the tool relative to the manipulator [Bone 88]. Others have used additional actuators to move the workpiece relative to the manipulator [Kazerooni 86b]. Both methods have been designed to enable the necessary degree of precision.

Accuracy is an issue for position controlled industrial manipulators that interact with some other system and careful calibration is required to achieve this [Greig 95]. Accuracy is also important in force control when trying to gather position data to improve a world model. However, due to the conforming nature of all force control schemes, accuracy is not the greatest problem in this field. Force controlled motion must always take place relative to the true position of the workpiece since it only works when manipulator and workpiece are in contact. Therefore, whether the
workpiece is precisely mapped globally or not, it is still defined locally relative to the manipulator and thus whatever task is planned may be carried out.

2.3. Conclusions

A range of different force control methodologies have been outlined in this chapter along with some of the key advantages and disadvantages of each. Chapter three gives additional reasons for the choice of using a passive-compliance force-control scheme.

For the purposes of this thesis, a controller is selected that looks something like the force control loop closed about the stiffness control loop. The force loop incorporates an Artificial Neural Network to calculate the desired position for the inner position loop as shown in figure 2.1. This is a reasonably common basic structure [De Schutter 88a,b] not least because most manipulators (including the PUMA 560 used in these experiments) are primarily designed as position controlled devices.

As manipulator control matures, there is a tendency for programming and control to become separate issues [De Schutter 88a,b] [Langrock 94]. In other words, the user will specify a task and the control aspects are handled automatically. This is often described as using a high level command structure where the low level commands may be assumed to be handled automatically - they are hidden from the user.

"Compliant motion of a manipulator occurs when the manipulator position is constrained by the task geometry" [Mason 81]. This compliance may be passive or it...
may be implemented using force control. For the work in this thesis both have been used. The control structure for the system described in this thesis is therefore a hybrid force/position controller.
3.1. Chapter Outline

The purpose of this chapter is to more fully define the inspection problem of detecting cracks in welds on offshore structures. To achieve this it will:

- Describe the Automated Remote Manipulator (ARM) system in some detail. This is the actual hardware that is used offshore for various sub-sea tasks and basically comprises an ROV, a hydraulic manipulator arm and a control system.
- Describe possible instabilities or disturbances to the ROV as a working platform when it is attached to the oil rig jacket. These are due to sea currents, waves and forces arising from the manipulator itself.
- Describe the nature of the surface contact required to do a weld crack inspection.
- Describe the reasons for selecting compliant force control as the optimal solution to achieve this form of contact.

3.2. The Automated Remote Manipulator (ARM) System

The ARM system consists of an advanced hydraulic robotic manipulator with six dof (six independent axes of rotation). This is mounted on an extend/rotate mechanism which provides an additional two degrees of freedom (one rotational and one translational). This is a steel boom which will extend from a tool skid and rotate about its own axis. The tool skid also has three extensible legs with suction feet on the end to enable attachment of the tool skid to a jacket. In addition it carries the controllers and power supplies for the manipulator and any tools.

The tool skid may be carried by a standard, work class ROV to deliver the tools to the inspection site. This configuration is shown in figure 3.1. A real configuration is seen in the photograph in figure 3.2.
A personal computer (PC) based control system on the surface supplies a full three-dimensional graphical interface to the user and communicates with the control electronics mounted on the tool skid.

**FIG. 3.1 - SCHEMATIC OF TOOL SKID ATTACHED TO ROV**

**FIG. 3.2 - REAL ROV AND TOOL SKID ASSEMBLY**
3.2.1. Remotely Operated Vehicle (ROV)

The ROV in figure 3.2, for which the tool skid was originally designed, is built by Slingsby engineering. It was designed as a flexible vehicle to which additional units can be added and is called the Multi-Role Vehicle (MRV). It uses a top hat design of tether management system (TMS) as illustrated in figure 3.3. As mentioned in chapter one, the tether management system is necessary to prevent the ROV having to pull the load of the much heavier lifting cable. Instead, the heavy lifting cable is used to lower the ROV to its operating depth. At this point, the ROV disconnects from its tether management system and flies to its work site towing a cable that is only for power and communications but not lifting. It is, therefore, a shorter cable of smaller cross section. As a result, the drag forces due to sea currents and the motion of the ROV relative to the other end of the cable are less as these are roughly proportional to the projected area of cable as seen by the current flow (diameter x length).

The core vehicle measures 2.7m by 1.5m and is 1.56m tall. It uses six hydraulically powered thrusters and is rated to work at depths of up to 1000m. The full specification is included in Appendix A.
3.2.2. Manipulator Arm

The manipulator arm, also built by Slingsby engineering, was designed to be able to reach a high percentage of the welds around a node from a single base position. With its offset cranks and large angular range in many joints, it is highly dextrous. It has a large work envelope with a reach of 2.5 metres. The data sheet is in Appendix B.

3.2.3. Tool Skid

The tool skid, also built by Slingsby engineering, was designed to be joined to the MRV or any other similarly sized work class ROV. The tool skid extend/rotate mechanism allows an extension of up to 2 metres and a rotation of up to 360 degrees. This considerably increases the reachable workspace and, for most nodes, allows the manipulator to reach almost the entire weld without the need to move and re-attach the ROV to the jacket. The tool skid carries electronics to control the extend/rotate mechanism, the manipulator and any attached tools as well as a hydraulic pump to drive the vacuum feet and the manipulator.

It is common to have some additional cameras fixed to the front of the tool skid as well as some flood lighting.

3.2.4. Control System

The control system, built by Technical Software Consultants, was designed to enable control of the manipulator in a number of different ways. These are:

1. Position Feedback Mode. A manual control method. A miniature model of the real arm is moved by the operator. The real manipulator follows its motions - known as 'master/slave' control.

2. Supervisory Control Mode. The manipulator may now be moved in a variety of standard methods such as position control, joint control, cartesian control. The operator is giving relatively high level commands which are interpreted by the supervisory computer which, in turn, controls the manipulator. For cartesian control, the operator may select whether to use global co-ordinates (the ROV co-ordinate frame), tool co-ordinates or workpiece co-ordinates. For this type of inspection work, the workpiece co-ordinate frame is particularly useful. If the workpiece model has been created in terms of radius and angle from a reference line
along a cylinder centre (polar co-ordinates), then to follow the circumference of that cylinder, only the angle need be adjusted. This is considerably more intuitive than calculating the surface position in cartesian co-ordinates and also calculating the orientation required of the probe.

3. Fully Automatic Mode. In this mode a very high level command such as ‘perform inspection’ is given. The manipulator will then automatically move to the workpiece and perform its task.

The most important aspect of the ARM control system as a solution to the problem of sub-sea inspection, is the presence of low level intelligent modules that relieve the surface operator of certain difficult tasks. That is, the operator is able to specify a task, such as tracking a 20 degree arc of weld, with only a few simple commands. The commands are interpreted by the low level modules and converted into commands, in an appropriate co-ordinate frame, that can actually drive the manipulator. The high level commands enable tasks such as modelling the structure, envisaging complex three-dimensional paths upon that structure and moving the manipulator along the surface of the structure.

3.3. ROV Motion

When the ROV has flown to the target node, it attaches to the jacket using three vacuum feet. At this point, the ROV becomes a stiff working platform from which the manipulator can be deployed for a variety of tasks.

It is known from trials in the water tank at the National Hyperbaric Centre in Aberdeen that this platform is not perfectly stiff [Broome 96]. The ROV moves under load due to distortion of the elastomeric vacuum feet. In the North Sea it is envisaged that loads may arise due to the following reasons:

3.3.1. ROV Motion Due to Sea Currents

Sea currents acting on the ROV structure and the umbilical cable. In the North Sea, currents of several knots (1 knot = 0.49 m/s) are commonplace. These exert static loads on the ROV. For Reynolds numbers from $10^2$ to $10^7$ in a fluid flow past a cylinder, vortex shedding may take place which can create dynamic loads. The equations
explaining the relationship between fluid velocity, cylinder diameter, Reynolds number and frequency of vortex shedding are, for an idealised infinite length cylinder [Massey]:

\[ \text{Re} = \frac{V \cdot d}{\nu} \]  

(3.1)

\[ f = \frac{V}{d} \left( 1 - \frac{19.7}{\text{Re}} \right) \]  

(3.2)

where:  
- Re - Reynolds number.  
- V - fluid velocity.  
- d - cylinder diameter.  
- \( \nu \) - kinematic viscosity of seawater.  
- f - vortex shedding frequency

Figure 3.4 illustrates how the frequency of vortices shed from a cylinder vary as the cylinder diameter increases. This is shown for a range of fluid velocities from 0.5 to 2.0 m/s. The red line shows the natural frequency of the attached ROV. It may be clearly seen that, as the fluid velocity increases, the required cylinder diameter to create vortices at a frequency matching the natural frequency of the ROV, increases [Broome 96]. When the vortex shedding frequency is near the structural vibration frequency of a body, resonance can occur resulting in such dangerous behaviours as galloping of sub-sea cables [White 79].

The power of these vortices is a function of the water velocity and so the powerful effects will take place at higher velocities. Hence, figure 3.4 illustrates that powerful vortex shedding at a dangerous frequency will take place in rapid flow past greater diameter cylindrical steel members.
3.3.2. ROV Motion Due to Wave action

ROV motion may occur due to waves acting on the ROV at shallow depths (less than 10m). Near the sea surface, this is a powerful effect and the resultant forces on the ROV will cause it to oscillate.

3.3.3. ROV Motion Due to Contact Forces

Reactions due to contact forces between the manipulator and the structure may cause the ROV to oscillate. Inspection requires surface contact and so there will inevitably be a reaction force of some description acting on the ROV which will cause motion. Some other tasks requiring large contact forces such as grinding could generate large motions of the ROV.

A possible additional problem could be a situation where it is not possible to attach more than two vacuum feet to the jacket or the feet are not well spread due to geometrical constraints (all in a line). This will cause a greatly reduced ROV stability as the ROV will be able to rotate about the straight line joining the vacuum feet. This will be constrained to a degree as the leg and vacuum feet joints are all locked off once the ROV is attached.

3.4. Required Surface Contact to Achieve a Weld Inspection

By surface contact is meant the contact between the crack detection probe and the metal surface being inspected. The precise task will vary according to the type of probe being used and the geometry of the weld being inspected.

For this thesis, we are concerned with an alternating current field measurement (ACFM) probe. This operates by using a set of coils to induce a current in the steel surface. Other coils then detect the near surface magnetic fields created by the induced current [Lugg 96]. Any crack that is present in the steel will distort the detected field and so the crack start and finish points may be clearly pinpointed.

It is a non-contact process as neither the induction nor the detection coils require surface contact. However, for an ACFM probe inspection, the coils need to be a known distance above the conducting surface. This distance is provided by the probes' own casing and so the probe casing is required to contact the surface. Since surface contact is required, it is a logical progression to use that surface contact to guide the process.
This may be done by using contact force monitoring to make decisions as to the path the manipulator must take.

In addition, some very constrained spaces must be reached such as that shown in figure 3.5. A tight angle between a chord (main steel member) and a brace (intersecting steel member) may be as low as 22 degrees [API 93]. The illustration shows 30 degrees. This constraint requires that the probe be attached at its back as there is not room for the robot manipulator to reach right into the tightest angle of the crack. This has repercussions for the design of the compliant device as is discussed in chapter seven.

The weld has both depth and width. Either side of the weld, the intersection with the chord or brace is called the toe of the weld. The probe must track along both toes of the weld and possibly along individual beads if the weld is large enough. These are illustrated in figure 3.5 which shows a section through a large weld. This is built up of multiple small beads. The weld beads introduce a high surface roughness or bumpiness. Additional roughness is added by there being weld spatter on the surface. That is lumps of molten metal that have landed randomly during the welding process. This constraint requires any probe to follow the rapidly changing surface contours.

For the purposes of this Ph.D., it is assumed that our goal is to be able to inspect the two weld toes of a weld within a 30° cylinder intersection.
3.5. The Requirement for Compliant Force Control

In this thesis, compliant force control was chosen as an appropriate method to automate the inspection of sub-sea welds. This choice is justified in this section.

What is meant by compliant force control may be explained by examining the simplified physical system illustrated in figure 3.6. This shows how, as a probe moves across a surface (3.6a), it meets a sudden change in surface height (3.6b). This ‘disturbance’ to its trajectory will compress the compliant device (here shown as a simple spring) and transmit a force to the force sensor. The force magnitude is then used to adjust the manipulator trajectory such that the force returns to its original level (3.6c).

The key point of compliant force control is that a large change in surface height can happen rapidly without damaging the probe or force transducer as the motion is absorbed by the compliance. This has the effect of moderating the force change that takes place.

In brief, we are using force feedback to modify the manipulator trajectory, and a passive compliance to allow for rapid changes in surface height. The justification for this approach is:

1. As was mentioned in chapter one, sub-sea inspection is a slow task. Whether carried out by divers or ROV’s, the process is slow at every stage from deployment, through inspection to completion. Hence, the speed with which the inspection itself is carried out has a relatively small impact on the overall time taken. Compliant force control, therefore, may be used even though it results in slower manipulator motions than a stiffer, higher bandwidth system would give.

FIG. 3.6 - SIMPLIFIED SCHEMATIC OF THE MECHANICS OF COMPLIANT FORCE CONTROL
It has the advantage that the manipulator may be less precise and operate at a lower bandwidth saving considerable expense.

2. The accuracy with which the structure geometry is known will be severely limited due to structure mis-modelling. This may arise from the control system having a simplified model of the true environment and will cause errors as a result of:

   - Non-registration. The known relative positions of the ROV and structure are incorrect.
   - Incorrect data. The engineering drawings of the structure contain nominal dimensions. The true structure will differ slightly from these in both size and geometry.
   - Noise. The structure surface will not be smooth mainly due to weld spatter, marine growth and corrosion.

These inaccuracies may be taken up by the movement of a compliant device. Non-registration and incorrect data will not matter once contact has been made as force control will ensure the correct stand-off between manipulator and work piece. The noise will matter. Marine growth will have been removed prior to inspection but weld spatter may result in disturbances to the probe position of up to 4mm.

3. As discussed in section 3.3, there will be ROV motion relative to the structure. These motions will increase the problem of non-registration and make it a dynamic problem. These motions need to be compensated for and are up to 15mm (see chapter 5). With a compliant device a slower response time may be accommodated.

4. Manipulator imprecision. The manipulator will not go precisely where it is asked and so the position errors may be taken up by the compliant device. The repeatability is 1mm (see Appendix B).

The resulting errors are cumulative with noise adding 4mm, ROV motion adding up to 15mm and manipulator imprecision adding 1mm. This gives potential errors of 20mm and so, for the real sub-sea application, a compliant device would need travel of up to ±20mm with contact forces of up to 100N max. This will result in constrained direction stiffnesses of 5000 N/m.

An additional advantage of using a compliant device between the probe and the manipulator arm is that it simplifies tele-operation of the arm. When the manipulator
position is being directly controlled by a surface operator using camera views, the
deformation of the compliant device gives the operator good visual cues as to the
contact made. It is normally very hard to judge the nature of a contact just from camera
views as the view points tend to be fixed and the visibility poor.

3.6. Conclusions

The key conclusions from this chapter are:

1. There is considerable hardware in existence to carry out sub-sea inspection work.

2. A specific problem that is identified as requiring a solution is that of running a probe
   along a weld toe at the junction of two cylinders when the angle between the
cylinders is 30°.

3. Compliant force control is chosen as an appropriate solution to the problem because
   inspection speed is not critical, model errors are expected and it is known that the
docked ROV is unlikely to be a rigid platform.

The design of the compliant device is detailed in chapter seven.
4.1. Chapter Outline

The purpose of this chapter is to describe the PUMA 560 industrial manipulator and the way in which it may be controlled by a supervisory computer. In addition it details the structure of the supervisory software. The code may be found in Appendix C. This is dealt with by detailing:

- General manipulator background.
- Manipulator geometry.
- Manipulator controller.
- Supervisory controller.
- Manipulator co-ordinate systems.
- Supervisory software development.

4.2. General Description

The manipulator used throughout this work is a Unimate PUMA 560 Mark II as illustrated in figure 4.1. This is a popular research robot originally developed as a position controlled industrial manipulator for factory assembly work. It was developed by the General Electric Company in 1975 [GREIG 92] [GREIG 96]. The manipulator used in this thesis was purchased in 1986.

The manipulator specifications as supplied by the manufacturer are given in table 4.1 [UNIMATION 85a]. The PUMA 560 has six revolute joints giving it six degrees of freedom (6 dof). Each joint is driven by a geared DC motor using an incremental optical encoder to feedback the joint position.
The control and power electronics are all housed in the box labelled PUMA control cabinet. The user may program the controller via the keyboard and visual display unit (VDU) standing on top of the electronics housing. In addition there is a teach pendant which enables the user to move the manipulator to a position manually and then teach that position to the controller so that it can return there automatically. This might be used, for example, to teach a paint spraying robot to hold the paint tool in the correct position and orientation.

An additional component of the system is a fifteen core, shielded cable running inside the manipulator arm to take signals from the end flange to the base. The end flange is the name given to the flange plate at the end of the manipulator arm that connects the manipulator to any required tooling. A pneumatic internal link also runs between the end flange and the base to provide power for a gripper or other tool. The air supply is controlled by a solenoid. Neither of these additional components are used in this work.

The PUMA manipulator is mounted on a steel plate that is mounted on two parallel ‘U’ channel beams. There are a number of spare mounting holes on the beams and so it is fairly straightforward to slide the manipulator, on its plate, to a new position or orientation [GREIG 96].
### Item Specification

<table>
<thead>
<tr>
<th>Item</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axes</td>
<td>six revolute</td>
</tr>
<tr>
<td>Work envelope (max. clearance required)</td>
<td>spherical volume radius 0.92m from shoulder</td>
</tr>
<tr>
<td>Mass</td>
<td>62 kg</td>
</tr>
<tr>
<td>Drive</td>
<td>Electric DC servomotor</td>
</tr>
<tr>
<td>Maximum load</td>
<td>2.3 kg</td>
</tr>
<tr>
<td>Maximum moments of inertia:</td>
<td></td>
</tr>
<tr>
<td>Joint 5</td>
<td>0.042 Ncms²</td>
</tr>
<tr>
<td>Joint 6</td>
<td>0.0041 Ncms²</td>
</tr>
<tr>
<td>Maximum static force at tool</td>
<td>60 N</td>
</tr>
<tr>
<td>Positional repeatability</td>
<td>±0.1 mm</td>
</tr>
<tr>
<td>Maximum tool acceleration</td>
<td>9.8 m/s²</td>
</tr>
<tr>
<td>Maximum tool velocity</td>
<td>1.0 m/s (with maximum load)</td>
</tr>
<tr>
<td>Maximum straight line tool velocity</td>
<td>0.5 m/s</td>
</tr>
</tbody>
</table>

**TABLE 4.1 - PUMA 560 MANIPULATOR SPECIFICATIONS [UNIMATION 85a]**

### 4.3. PUMA 560 Manipulator Geometry

The PUMA 560 joint travel geometry is listed in table 4.2, along with the actual angular ranges found by moving the PUMA to its limits [UNIMATION 85a]. The joint axes are illustrated in figure 4.2. In addition, the link lengths between joints are shown in table 4.3. This shows the nominal link lengths and those measured by various researchers [GREIG 96], [LOVELL 90], [FU 87]. The measurements taken by Lovell and Greig were on the actual manipulator used in this work.

The reduction in angular range of joint 4 and the increase of joint 3 from the stated values has not proven a problem in this work.

#### Joint Angular Range

<table>
<thead>
<tr>
<th>Joint</th>
<th>Stated angular range</th>
<th>Actual angular range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waist (joint 1)</td>
<td>320°</td>
<td>same</td>
</tr>
<tr>
<td>Shoulder (joint 2)</td>
<td>250°</td>
<td>266°</td>
</tr>
<tr>
<td>Elbow (joint 3)</td>
<td>270°</td>
<td>284°</td>
</tr>
<tr>
<td>Wrist (joint 4)</td>
<td>300°</td>
<td>280°</td>
</tr>
<tr>
<td>Wrist (joint 5)</td>
<td>200°</td>
<td>same</td>
</tr>
<tr>
<td>Wrist (joint 6)</td>
<td>532°</td>
<td>same</td>
</tr>
</tbody>
</table>

**TABLE 4.2 - PUMA 560 JOINT ANGLE RANGE**
The precision link length measurement carried out by Greig and others was as part of a calibration exercise to ascertain the repeatability and accuracy of the manipulator and thereby improve upon it. The repeatability of a machine is its ability to repeat any defined motion. Its accuracy is defined by its ability to attain a given point in space measured absolutely.

It was found that the true repeatability of this manipulator was 0.13mm compared to the factory specification of 0.1mm. This was found to be a function of the encoder resolution and so could not be improved upon. The manipulator accuracy was found to be ±0.32% and, by calibrating each joint independently, error curves for each joint could be produced. Using these curves to compensate when commanding joint positions increased the accuracy fourfold to ±0.08% [GREIG 96].
For this thesis, it is assumed that the PUMA link lengths and joint angle offsets have not materially changed in the five years since Greig’s calibration work. Given the relatively light work duties of a research robot (it is not on a production line 24 hours a day) and the knowledge that no structural components have been replaced, this seems reasonable. It is therefore intended that, should highly accurate work be required, use will be made of Greig’s compensation data.

4.4. PUMA 560 Manipulator Controller

4.4.1. Hardware
The controller module, housed in the PUMA control cabinet, contains a DEC LSI-11 computer comprising the processor, memory and communications hardware. The system software is stored in Erasable, Programmable, Read Only Memory (EPROM). User software is stored in volatile Random Access Memory (RAM) and non-volatile CMOS memory. The RAM capacity of a standard PUMA controller is 16K.

The LSI-11 acts as a user interface allowing instructions to be entered using the keyboard or disc drive. Instructions are entered using the PUMA language VAL (see section 4.4.2) and interpreted to determine scheduling priorities such as:

1. Co-ordinate systems transformations between world, joint and tool co-ordinates.
2. Trajectory planning in either joint or cartesian co-ordinates to ensure smooth path following. This requires the calculation of interpolated joint positions.
3. Communicating with each joint (via a Rockwell 6503 microprocessor) every 28 ms (approx. 36 Hz) to confirm their status and update their demanded position.

At a lower hierachical level, the six Rockwell 6503 microprocessors (one for each joint) are each connected to a joint encoder, a digital to analogue converter (DAC) and a current amplifier. The microprocessors form part of the servo loops for each joint drive. The communication between the LSI-11 and the microprocessors occurs every 28 ms (approx. 36 Hz) whilst the inner servo loop between each microprocessor and its encoder is interrogating every 0.875 ms (approx. 1143 Hz).

The PUMA manipulator control is a conventional PID scheme (proportional plus integral plus derivative) with each joint treated as a separate servo. The feedback gains are pre-set for a certain conservative performance at a nominal speed. This means that,
for operation under varying loads or speeds, the control gains will give a stable system but performance will frequently degrade as the manipulator is a highly non linear system.

4.4.2. Software

The software used to operate the PUMA is a proprietary system language called VAL (variable assembly language). The current version is VAL II.

This is a BASIC like language with a library of powerful, simple command words. It has the facility to store manipulator end flange locations in two ways:

1. Precision points are absolutely defined by the angle of each joint. There can be no ambiguity as to the configuration of the manipulator, i.e. it is possible to have a manipulator in the two configurations shown in figure 4.3 and still have the end flange at exactly the same position and orientation in space. The disadvantage of precision points is that they are manipulator specific. To calculate the end flange position in space requires an accurate model of the manipulator. It is also much less intuitive for people to visualise the manipulator location.

2. Transformations record the cartesian co-ordinate \((x,y,z)\) and orientation \((OAT - an Euler angle representation described in chapter nine)\) of the end flange with respect to the base. This has the advantage that a position thus identified is not manipulator dependant. It is also simpler to visualise. The drawbacks of this system are that the manipulator configuration is not defined, as was seen in figure 4.3., and also an inverse kinematic solution is required to calculate the joint angles.

![FIG. 4.3 - PUMA POSITION AMBIGUITY](image.png)
The PUMA manipulator has an analogue to digital converter (ADC) for sensor input. For any sensor fixed to the manipulator end flange, if it gives a DC voltage output, the ADC enables the signal to be measured. This can be very useful. For example, when a force signal is measured, if a force threshold is exceeded, the manipulator motion can be arrested to prevent damage to the tool or workpiece. More complex software may also call subroutines on contact or adapt the end effector position in proportion to the end flange force. The limitations are set by system memory and processing speed. To perform more complex tasks such as additional transformation calculations requires an external computer as a supervisory controller.

4.5. Supervisory Controller

4.5.1. Hardware Link

The PUMA controller has the facility to connect to an external computer via an industry standard RS232C specification serial link. This is capable of running at speeds of up to 38400 baud (equivalent to approx. 4000 bytes/second). The link uses the DDCMP (digital data communications message protocol) specification developed by DEC [DEC 78] [UNIMATION 85b].

The external computer (for this work an IBM clone is used) enables several things otherwise not possible:

- Use of high speed ADC cards for data sampling.
- Large scale data storage.
- Code development in a quality programming environment. In this case the Microsoft Visual C++ compiler.
- Off-line processing and manipulation of sampled data.

The nature of the communication between the PUMA controller and the computer is shown in table 4.4 illustrating the typical time allocation for the different elements during a single 28 ms cycle. For a fast PC such as the 66 MHz machine used in this work, the 21 ms of free processing time between communications allows up to 346500 floating point operations to be performed (assuming four clock cycles per floating point operation). The 28 ms cycle is set by the PUMA controller and limits the update rate at which the computer can update the manipulator demanded position to just under 36 Hz.
4.5.2. Nature of Messages

During normal communication, the PUMA controller sends limited information on the current manipulator status as well as some error indicators. It can, on demand, also send the current end flange transformation. However, this requires 24 additional bytes to be sent and so slows down the protocol by an additional 6.0 ms, reducing the free processing time available to the computer by almost one third.

The computer sends a co-ordinate change \((x,z,y)\) and an orientation change \((OAT)\) by which amount the PUMA controller will adapt its current position. In practice, the most effective mode of operation requires the PUMA controller to keep telling the manipulator to move nowhere. This (non) motion is then modified by the signal from the computer. Effectively, high level control is now given over entirely to the computer because the internal PUMA motion commands are to move nowhere.

4.5.3. E-Slave Communication Protocol

An alternative communications protocol does exist which effectively bypasses the VAL II part of the controller. E-slave, developed by AEA Technology, enables an external computer to directly control position or torque demand to each joint with a clock rate of 3.6 ms between command signals (approx. 278 Hz). This uses an ethernet link between the computer and the Unimate controller. This was not chosen for this work because the high clock speed was not necessary. The e-slave system is also expensive with the entire system being added for over £15000. A further problem is that the VAL software cards are no longer manufactured and so parts may be hard to replace.

4.6. PUMA 560 Co-ordinate Systems

The Joint co-ordinate system is straightforward. Each joint is measured from some datum point in degrees. This is precise but seldom used for the reasons explained in section 4.4.2. In practice, the homogeneous transformation is used to describe robot
positions and orientations [Denavit 55]. This uses a cartesian co-ordinate frame as shown in figure 4.4, and the Euler representation of angles to define the end effector position and orientation. This is explained in more detail in chapter nine.

![PUMA GLOBAL CO-ORDINATE SYSTEM](image)

**FIG. 4.4 - PUMA GLOBAL CO-ORDINATE SYSTEM**

### 4.7. Supervisory Software

#### 4.7.1. Previous Software Development

The software inherited from previous research in the Automatic Control Group was adequate for the key computational tasks [Wang 92]. It enabled serial communications between the supervisory computer and the PUMA controller. In addition, it allowed key calculations to be performed between each communication cycle and enabled diagnostic and experimental data to be written to file.

However, the style was not modular in terms of functionality and so changes to the code were both cumbersome and unreliable. It was therefore decided to rewrite the code using a more advanced class structure. The purpose of this was to create units of code to carry out particular functions at various levels of abstraction. For example, in the communications module, the transmission of an individual byte was a functional part of the transmission of a particular message type. Therefore, the main program would call a function called GetMessage which in turn would make use of a function called GetByte. This type of abstraction had been used in the earlier software but to a much lesser degree and with the functional divisions less clearly defined.
4.7.2 Guidance Strategy and Control Structure

The supervisory computer needs to achieve a range of guidance and control tasks in order to be able to assess the effectiveness of the artificial neural network. These are:

1. Moving from free space to make contact with the workpiece.
2. Make the transition to stable force contact in the required inspection position.
3. Track along a prescribed path whilst maintaining stable contact with the workpiece.
4. Move safely from the workpiece to a neutral position at the end of the task.

To achieve these steps, an overall control structure as shown in figure 4.5 was used. The strategy of the controlling software was to divide the overall task into three sub-tasks as they broke down in points 1 to 3 above. Thus the first sub-task was to move in a pre-defined direction until the force sensor indicated that contact had been made.

The second sub-task was to move and rotate the manipulator end effector until the forces in all six degrees of freedom were equal to their target values (or close enough to
lie within a tolerance). The third sub-task was then to track in the desired probe co-
ordinate direction whilst maintaining the forces at their correct set points.

The safe return to a neutral position at the end of the tracking task (point number 4
above) was in fact achieved by the PUMA controller once the supervisory PC had
indicated that the task was complete.

4.7.3. Class Based Supervisory Software Structure

The flow diagram for the supervisory structure is detailed in figure 4.6. The basic
structure is very simple although some of the modules are very complex, e.g. the box
called ‘Plan new manipulator position according to Contact algorithm’ crosses several
classes. This task includes Neural Network modelling of the compliant device,
calculation of the current tool position and transformation of a tool motion into the
manipulator co-ordinate frame.

4.7.4. Class Map

The class map shows the functional decomposition of the main software modules into
class member functions that perform specific functions. The main modules are:

1. Track4.cpp - this is the main program and in appearance resembles the flow diagram
   of figure 4.6.

2. JCAinn.cpp - this is the class containing all functions relevant to the solution of the
   artificial neural network used to model the behaviour of the compliant device.

3. JCContr1.cpp - this is the class containing the software pertaining to the control
gains and other devices necessary to achieve stable surface tracking.

4. JCDaq100.cpp - this is the class containing all functions necessary to read the force
torque sensor using the analogue to digital converter (ADC).

5. JCMat1x.cpp - this is the class containing a range of matrix and vector
   multiplication functions.

6. JCPuma.cpp - this is the class containing all the functions enabling serial
   communications between the PUMA and the supervisory computer.
The member functions of each class and their purpose are listed in table 4.5. No further description of the software functionality is given in this thesis. The source code is included in Appendix C for further reference.
### TABLE 4.5 - SUPERVISORY SOFTWARE CLASS MAP

<table>
<thead>
<tr>
<th>Class</th>
<th>Member functions</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>JCAnn</td>
<td>CDTransform</td>
<td>Given a force reading from the sensor, this returns a transformation of the probe centre position and orientation relative to the manipulator end flange.</td>
</tr>
<tr>
<td>JCCtrl</td>
<td>CtrlFreeSpace</td>
<td>Returns the desired end flange transformation based on desired position.</td>
</tr>
<tr>
<td></td>
<td>CtrlForce</td>
<td>Returns the desired end flange transformation based on desired contact force.</td>
</tr>
<tr>
<td>JCDa100</td>
<td>DaqReadCh0</td>
<td>Each will read a single value from the single channel indicated by the function name and scale it to the correct voltage.</td>
</tr>
<tr>
<td></td>
<td>. DaqReadCh7</td>
<td>Reads six channels required for the force sensor and scales them to the correct voltage.</td>
</tr>
<tr>
<td></td>
<td>DaqReadAllSix</td>
<td></td>
</tr>
<tr>
<td>JCMatrix</td>
<td>MatXM4</td>
<td>Multiplies two 4x4 matrices.</td>
</tr>
<tr>
<td></td>
<td>MatXM3</td>
<td>Multiplies two 3x3 matrices.</td>
</tr>
<tr>
<td></td>
<td>HomInv</td>
<td>Inverts a 4x4 homogeneous transformation.</td>
</tr>
<tr>
<td></td>
<td>Trans</td>
<td>Creates a transformation from a 6x1 vector.</td>
</tr>
<tr>
<td></td>
<td>Decompose</td>
<td>Creates a vector from a transformation.</td>
</tr>
<tr>
<td>JCPuma</td>
<td>PumaGetByte</td>
<td>Reads a single byte from the PUMA.</td>
</tr>
<tr>
<td></td>
<td>PumaSendByte</td>
<td>Sends a single byte to the PUMA.</td>
</tr>
<tr>
<td></td>
<td>PumaGetInitialMessage</td>
<td>Reads an initial message from the PUMA.</td>
</tr>
<tr>
<td></td>
<td>PumaSendInitialMessage</td>
<td>Sends an initial message to the PUMA.</td>
</tr>
<tr>
<td></td>
<td>PumaGetMessage</td>
<td>Reads a standard message from the PUMA.</td>
</tr>
<tr>
<td></td>
<td>PumaSendMessage</td>
<td>Sends a standard message to the PUMA.</td>
</tr>
<tr>
<td></td>
<td>PumaGetTransformation</td>
<td>Reads the current manipulator transformation.</td>
</tr>
<tr>
<td></td>
<td>PumaGetStartPosition</td>
<td>Reads the initial manipulator transformation.</td>
</tr>
<tr>
<td></td>
<td>PumaSetBytes</td>
<td>Ensures bytes transmitted to PUMA are non-zero.</td>
</tr>
<tr>
<td></td>
<td>PumaFakeMessage</td>
<td>Diagnostic to test for communications.</td>
</tr>
</tbody>
</table>

### 4.8. Conclusions

It can be seen that the PUMA 560 manipulator is a versatile tool enabling many motion controlled tasks to be carried out to a high accuracy.

It does have some limitations of which the most serious is the low clock rate at which it will receive commands from an external computer. Another drawback is the relatively slow serial link with which it talks to the supervisory PC thus reducing the available processing time per clock cycle for the PC.

The PUMA 560 is used for all the robot experiments described in this thesis. In chapter six, it is used to evaluate the use of a stereo vision system to compensate for ROV motion. In chapter eight it is used as the input device to gather data to train an artificial neural network and in chapter ten it is used to carry out some mock inspection tasks to evaluate the overall system.
5.1. Introduction

In March 1996, trials of an ROV equipped with a hydraulic ARM and a crack measurement probe were carried out in the freshwater tank at the National Hyperbaric Centre in Aberdeen. This equipment comprised the ROV attached to the Slingsby tool skid which was carrying the Slingsby ARM and the TSC controller as was specified in chapter three. Three legs with a vacuum foot at the end of each one were used to anchor the ROV to an underwater structure (in this case, the tank wall).

In operation, the ROV and its umbilical cable will be subject to sea currents and, at shallow depths, wave induced motions even whilst attached to the structure. In addition, the ARM itself can induce large forces as a result of deploying high pressure jet cleaners, drilling, grinding or other tools. These forces are estimated at about 200N [Broome 96]. It was found that with currents of 0.375 ms\(^{-1}\) (about 0.75 knots) the drag force on the ROV amounts to about 400N while a similar current acting on a 30m umbilical cable results in a force of about 650N [Broome 96]. It is thus necessary to know the static and dynamic response of the attached ROV in order to develop appropriate control software to counter these disturbances. This experiment was the first practical attempt to do this.

Experiments were performed to measure the motion of the ROV when it was attached to the water tank wall and a range of static and ramped loads were applied. Typical static loads of 25kg resulted in a deflection of the ROV of about 5mm, and the maximum movement during the transient response to an impulsive load was of the order of 10 to 15mm depending on the arrangement of the docking legs.
5.2. Experimental Set-up

As was previously stated, the ROV was mounted on the Slingsby tool skid which was carrying the Slingsby ARM and the TSC controller. The tool skid was also equipped with three telescopic legs with suction cups on the end. After the ROV was immersed in the freshwater tank, these were used to anchor the ROV to the tank wall as illustrated in figure 5.1.

Once established in its correct position, a range of static and ramped loads were applied to the rear and the side of the ROV using a cable passing over a pulley and carrying weights at its end. The static loads were increased from zero in increments of 10kg up to 30kg and reduced back to zero. An impulsive load was obtained by dropping the weight from a height of 1m over the side of the tank. These load levels were considered realistic compared with the environmental loading seen by a docked ROV, and this is justified in a later section. The tests were repeated for a number of docking arrangements, and results for a 'legs together' case (parallel) and a 'legs spread' case are quoted in this thesis.

To measure this motion, a mast was mounted on the ROV which protruded above the water surface. At the top of the mast were mounted three targets; each consisting of a ball bearing with a light source directed at it. These were then tracked by an analogue camera system. Each camera would track the brightest object in its field of view and output two voltages proportional to the $x$ and $y$ co-ordinate of the object within its focal
plane. This meant that each camera could only track one of the targets. It also meant that the experiment had to take place at night because the targets were not bright enough to ensure consistent tracking. The arrangement was as shown in figures 5.1.

5.3. Measurements Taken

Several measurements were taken during each test:

1. The camera positions and orientations both relative to one another and in a global frame of reference. In use, each camera was mounted on a long brace. The front and back of each brace was measured relative to that of each other camera as shown in figure 5.2.

2. Each camera position was also measured on a scaled site diagram for reference.

3. The relative positions of the targets and the ROV centre in a local frame of reference. These were careful measurements taken of the ROV dimensions to within ±5mm and measurements of the three targets relative to one another to within ±0.5mm.

4. A range of camera calibration measurements was taken as is explained in Appendix D. These were to calculate the transform from the output x and y voltages to image plane displacements.

5. The point of application, direction and magnitudes of the forces applied. These are shown in component form in the results section and are listed in Appendix E.

6. The camera output as the loads were applied.

FIG. 5.2 - MEASUREMENT OF RELATIVE CAMERA POSITIONS
5.4. Post Processing of Data

To decompose the data it was necessary to resolve it into an ROV motion frame of reference. This involved several steps.

First the data was filtered. The data had the ROV motion superimposed with several types of noise as shown in Fig. 5.3. These noise types were:

1. Outliers. These are anomalous signals far outside the range of possible motion and can be seen in the first 2 seconds of data 1) in figure 5.3. This occurred when the camera had failed to pick up the target and instead found another light source which was the brightest in the field of view. This occurred when the ROV target motion revealed street lights in the background.

2. Medium frequency (2 - 3 Hz) vibration which is presumed to be due to the compliance of the elastomeric suction cup on the end of the docking legs. This did not always appear and was partly dependant on the wave form, direction and magnitude of the load applied.

3. High frequency vibration of the order of 20Hz which is due to the high wind prevailing on the top of the exposed test tank exciting flexural vibration of the support masts for the targets. These masts were made out of stiff Aluminium U-sectioned channelling and so this high a frequency is entirely possible.

The filtering consisted of a first pass to remove the outliers and a second pass to apply a first order butterworth filter at a cutoff frequency of 2 Hz. A third pass (not shown in figure 5.3) used the data from the camera calibration process to convert each data point (an x-y voltage) into a set of unit direction vectors from each camera.

![FIG. 5.3 - RAW AND FILTERED DATA FOR IMPULSE LOAD](image-url)
The second step was to convert the camera x-y data into ROV position and orientation at each moment in time by developing a model of the entire system and then designing and running minimisation software such that the target position errors were minimised. This was a large task and was broken down into several main sections.

1. The cameras had to be positioned within the global model. For this it was possible to use the camera calibration measurements combined with some more general measurements, to move the cameras in space within the model until the measurement error criteria were met. This resulted in accurate camera translation placement to within ±5mm, although the orientations were still somewhat unreliable. This was because cameras 1 and 2 were almost in alignment and so large changes in relative angle resulted from small variations in the measured distances apart. These variations resulted from errors in measuring such large distances.

2. From site measurements, an approximate initial position of the ROV with respect to the global frame could be ascertained to within ±10mm. Using this initial estimate of ROV position, the target positions in space were calculated. It was thus possible to take the initial camera data values and transform them into direction vectors in the camera co-ordinate frame, and then rotate the cameras in the model such that the data vector pointed directly at the relevant target. Next the ROV was moved in space such that the direction vectors of subsequent data values intersected the targets in space. In this way, the global motion of the ROV was ascertained.

3. By varying the start position of the ROV, the solution could be found with a minimum of error thus indicating a more accurate start position.

5.5. Results

The resolved forces applied during most of the 47 tests are shown in Tables E1, E2 and E3 in Appendix E. The tests not shown are those for which there was no useful gathered data. In précis, we see that the maximum impulsive and static forces encountered are of the order shown in Table 5.1. These are not necessarily occurring simultaneously but are the maximums occurring across the whole range of tests, and maximum displacements are given in Table 5.2, again not necessarily corresponding to the maximum load.
TABLE 5.1 - MAXIMUM IMPULSIVE AND STATIC LOADS APPLIED DURING ALL TESTS

<table>
<thead>
<tr>
<th>load type</th>
<th>$F_x$ (N)</th>
<th>$F_y$ (N)</th>
<th>$F_z$ (N)</th>
<th>$T_x$ (Nm)</th>
<th>$T_y$ (Nm)</th>
<th>$T_z$ (Nm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>impulsive</td>
<td>1236</td>
<td>-1089</td>
<td>758</td>
<td>-1647</td>
<td>-1727</td>
<td>1595</td>
</tr>
<tr>
<td>static</td>
<td>274</td>
<td>-241</td>
<td>168</td>
<td>-364</td>
<td>-382</td>
<td>354</td>
</tr>
</tbody>
</table>

TABLE 5.2 - MAXIMUM DISPLACEMENTS OCCULRING DURING ALL TESTS

<table>
<thead>
<tr>
<th>load type</th>
<th>$D_x$ (mm)</th>
<th>$D_y$ (mm)</th>
<th>$D_z$ (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>impulsive</td>
<td>7.9</td>
<td>14.7</td>
<td>18.3</td>
</tr>
<tr>
<td>static</td>
<td>4.0</td>
<td>3.7</td>
<td>5.8</td>
</tr>
</tbody>
</table>

TABLE 5.3 - MAXIMUM IMPULSIVE AND STATIC LOADS APPLIED DURING TEST 34

The particular motion occurring during impulse tests 22 and 34 (side forces) are plotted in some detail. Figures 5.5 and 5.6 show the response of the ROV to a side force. In figure 5.5, the ROV legs are parallel to one another. In figure 5.6, the ROV legs are spread apart. The frame of reference of the applied force is shown in figure 5.4 and the values of the applied forces are shown in table 5.3. These forces are resolved into force and torque about the geometric centre of the ROV. For ease of interpretation of the results the plot show the motions resolved into the three principal global planes. In each case, the locus of motion begins at (0,0).

FIG. 5.4 - ROV FRAME OF REFERENCE FOR APPLIED TEST FORCES
FIG. 5.5 - TEST 22 ROV MOTION LOCUS - SIDE LOAD, ROV LEGS PARALLEL

FIG. 5.6 - TEST 34 ROV MOTION LOCUS - SIDE LOAD, ROV LEGS SPREAD
5.6. Correlation With a Linear 'Damped-Mass-Spring' Model

The raw position data (Volts) of the ROV subsequent to the 'drop weight test' has a typical transient characteristic as shown in figure 5.7, which shows 10s of motion data (sampled at 50Hz). It comprises a fairly noisy signal showing the movement of the ROV in both directions taking about 3.5s and returning to equilibrium with a slight offset due to the small added static load.

The noise is at approximately 20Hz and is most likely due to the vibration induced by the wind in the 3m long mast carrying the camera targets, or possibly in the camera mounts themselves. The data was filtered using a first order Butterworth filter with a cutoff frequency of 2Hz. This resulted in the plot given in Figure 5.8.

The filtering has cleaned up the signal and removed the 20Hz 'noise', but another vibration can now be seen superimposed on the main signal. This can be more clearly seen if it is sampled over a shorter time as is illustrated in figure 5.9, showing the first 2s of the ROV transient.

This parasitic vibration is assumed to be due to the flexure of the elastomeric suction pads which attach the docking legs to the structure, and is of the order of 2.5 to 3.0Hz. It is more noticeable when the ROV is stationary before and after the transient.
FIG. 5.8 - ROV RAW DATA TRANSIENT: FILTERED WITH 2HZ BUTTERWORTH

FIG. 5.9 - ROV RAW DATA TRANSIENT: FILTERED 2HZ: FIRST 2 SECS
During the transient the motion of the ROV caused the suction pads to take up a steady, presumably deflected, shape. A further filtering of the full 10s of data with another Butterworth filter, this time set at a cutoff frequency of 1Hz, results in the final clean motion transient shown in Figure 5.10.

The final filtered data shows an ROV motion with a period of about 3.5 secs, or a natural frequency of 0.286Hz. The characteristic is unexpected in two respects:

1. It is not obvious why the ROV appears to move in both directions to a change in load level
2. The second 'half period' of the transient is much larger then the first half, which infers some nonlinear stiffness or damping - or different values according to direction - or both these effects.

To examine the transient further, a mathematical simulation was created using Matlab software. This consisted of a simple linear damped-mass-spring with a view to including non-linear elements to perform a match to the experimental data. The listing is included as Appendix F and it was based on an ROV mass of 1.5 tonnes with a 0.286Hz natural frequency and a damping ratio of 0.5.
Initially the simulation was tested using a simple step input which only resulted in a unidirectional response. However the dynamics of the subsystem of the 'drop weight' were further considered, and it became obvious that this acted more as an impulsive force input system. The cable tension would rise rapidly, probably to 5 to 10 times the weight value, as the weight decelerated to a stop at the end of the cable, and it would then fall away again to the static value. A simple model of this impulsive force input was developed, and a typical response is shown in Figure 5.11.

Here we can see that the motion of the ROV is very similar to that measured. It does move in one direction under the influence of the fast rising force, but once the force is rapidly ramped back down the under damped ROV overshoots on its return to rest as observed in the trial. This also gives the slower second half period as observed.

FIG. 5.11 - MOTION RESPONSE OF LINEAR ROV MODEL TO FORCE "IMPULSE"
5.7. Discussion

The ROV displacement under average steady loads of approximately 23kg was found to have a mean value, averaged over the $x$, $y$ and $z$ directions of 4.5mm (with a standard deviation of 0.9mm), representing a docking stiffness of about 50kN/m. The docking system seems to be of similar stiffness in the three main directions. The high stiffness implies that normal operating loads will not disturb the ROV by more than 15mm, and so the motion of the ARM end effector will also be of this order - and so remain within the range of about 25-30mm that it is envisaged would be designed into an underwater compliant wrist, tool-holding system.

Under dynamic loading, as shown in Figures 5.5 and 5.6 for Test 22 'legs parallel' and Test 34 'legs spread' respectively, there is a marked increase in the deflection during the transient, and it is not the same in the different directions. The maximum deflection, averaged across all three directions is increased to 13.6mm (with a standard deviation of 4.3mm). Here there is a marked increase in the $y$ (14.7mm) and $z$ (18.3mm) directions, but $x$ is still held to only 7.9mm. This is due to the geometry of the elastomeric suction cup which will deflect more readily to side loads than to axially applied loads. Also the transient dynamics are much more oscillatory in the $x$-direction, exciting the stiffer vibration of the docking cups in this direction. The loci in each plane show total excursions of 15mm in $x$ and $y$ and 12mm in $z$ for the poorer docking geometry, Figure 5.5, which is improved to 11mm in $x$, 8mm in $y$ and 13mm in $z$ for the wider separated attachment of figure 5.6.

A more detailed analysis, and perhaps some other measurements of cable 'stiffness', etc., of the 'drop weight' excitation subsystem would result in a better model to use in the simulation. The predicted motion response with this improved excitation model could then be compared with the observed response and the model tuned to fit the experimental data. This would give a further insight into the nature of the attachment system's viscoelastic properties, any hysteresis or nonlinear stiffness effects.

It has been estimated that the normal practice of only using one or two docking arms, instead of the three used in ARM, would result in much greater errors, easily of an order of magnitude bigger - 50 to 100mm under similar static loading, and 2 to 3 times greater under dynamic loading. This poor docking performance would make simple deployment tasks such as inspection very difficult, and surface contact tracking or tasks involving impulsive loading would not be possible.
5.8. Conclusions

A series of docking assessment trials has been carried out with the ARM underwater inspection system. The deflections obtained under loads which are representative of those imposed on a ROV performing tasks on an oil rig structure are of the order of 15mm, and could easily be accommodated by the compliant suspension system used to mount the tool. Three docking legs are essential for automating tasks such as manipulator based grind repair, or bolt tightening which are the likely future tasks of this system.

These results confirm the requirement for compliant force control for sub-sea inspection.
6.1 Introduction

This chapter describes an experiment to test a stereo vision tracking system. This system, developed by Subsea Offshore Ltd, enables a manipulator to compensate for motion of its target relative to itself.

The path generation algorithms being developed for the ARM project assume that the shoulder of the manipulator remains fixed with respect to the workpiece. In practice, the ROV will be subject to sea currents and, at shallow depths, wave induced motions even whilst attached to the structure. From the work described in chapter five, it is envisaged that there will be real translations of the ROV centre of the order of ±10mm. It is assumed that these translations result from a rotation more or less centred on a plane joining the three vacuum feet. Even with a full reach of 2500mm on the hydraulic manipulator, this would indicate that the manipulator tip motion would be no more than the ±10mm of the ROV. However, it is possible that in many real situations, such even placement of the ROV legs will not be possible and so give rise to much larger displacements of the manipulator tip; possibly an order of magnitude greater. Such motions would certainly cause problems for the 'making contact' phase of the inspection and possibly during the inspection as well. To address this problem, a stereo vision motion compensation system was investigated.

The aim of motion compensation is to convert a tool tip position demand, expressed in the workpiece co-ordinate system, to robot manipulator shoulder co-ordinates. In other words, we know where we want to travel on the workpiece. In order to give the correct motion command to the manipulator, we need to know where the workpiece is relative to the manipulator. In this case, that relationship is constantly varying.
This chapter will thus:

- Describe a system developed to track targets using stereo vision.
- Describe the software addition to the ARM control software that allowed the vision system output to control the PUMA manipulator.
- Describe the test-bed developed to test the vision system and explain the experimental procedure.
- Describe the calculations used to process the experimental data.
- Describe and illustrate the results achieved.
- Draw conclusions from and summarise this aspect of the work.

### 6.2. Premise for the Experiment

It was required to simulate, in the laboratory, the situation in which the base of the manipulator arm was moving relative to a fixed target. One approach to the problem was to mount the entire manipulator on a moving platform such as a Stewart platform and move it to simulate the ROV motion as has been done previously [Nicolodi 90]. This is, however, an expensive method and it was decided instead to move the workpiece holding the targets in a controlled manner whilst the robot and camera system remained still as illustrated in figure 6.1. This arrangement can produce identical relative motions between the target and the PUMA shoulder for the purposes of assessing the tracking performance.
The PUMA manipulator was thus required to follow a moving workpiece in two dimensional space. The stereo vision system was developed by Subsea Offshore Ltd. to track the workpiece as it moved relative to the cameras and thus relative to the manipulator shoulder. The spatial data gathered by the vision system was then used to modify the path of the manipulator such that the manipulator end flange remained fixed relative to the target.

6.3. Vision Based Target Tracking System

The stereo vision system used was based on the Camera Alive (now Subsea Offshore Ltd.) NCS2 stereo viewing and measurement system with additional software to provide real-time tracking of targets. This system uses two PAL CCTV video cameras (Panasonic WV350CL) and the Matrox IM1280 image processing system mounted in a 33MHz 486 based PC.

Software was developed to identify the individual targets in each image, to track these targets as they move, and to combine data from the two images to calculate the 3D co-ordinates of the centroid of each target. These co-ordinates were then transmitted to the ARM Vision system computer by an RS232 serial link.

6.4. ARM Vision Software

The ARM control software is a windows based manipulator control package developed by Technical Software Consultants (TSC). This system comprises a front end IBM PC clone giving the user a graphical representation of the manipulator being controlled and a set of virtual controls. This sends high level commands to a 68030 microprocessor control system that actually controls the manipulator in real time. For these tests, an addition to the ARM software is the ARM Vision software module. This is an additional module to take the camera data, apply motion compensation algorithms, and produce high level commands for controlling the PUMA robot. These high level commands were interpreted by the TSC ARM controller and used to directly control the PUMA joints. As this was a first trial, the software module was incorporated in the PC and not the microprocessor to allow rapid debugging. As will be seen in section 6.6, this had the implication of slowing the system update speed. In a real application, this could be minimised by transferring this module to the microprocessor.
6.4.1. Theory

A number of white targets were attached to a dark workpiece. This gave a number of highly contrasted targets, in a fixed pattern relative to one another, to be used for the motion compensation trials. The geometrical information describing the arrangement of targets on the workpiece was stored in the ARM workpiece data file. This information fully defined the workpiece co-ordinate system. Motions of the workpiece modify the demanded position in manipulator shoulder co-ordinates even when the required absolute position (in workpiece co-ordinates) is unchanged. i.e. while holding station above a single point on the workpiece (fixed workpiece co-ordinates) the manipulator can still move to keep station as the entire workpiece moves.

Target movements also alter the camera view of the workpiece. The transformation between the manipulator and workpiece varies as the targets move and must be calculated from the effective motion of the targets measured by the stereo camera system (transformations are described in detail in chapter nine). It can be calculated by considering a chain of transformations. The relationship between the manipulator and camera co-ordinate systems is determined by the position of the camera in the manipulator co-ordinate system. This is then combined with the workpiece transformation matrix to form a transformation relating the manipulator co-ordinate system to the workpiece co-ordinate system.

The method determined for calculating the required transformation matrices was used as the basis for producing a software library of functions for carrying out the various algorithms. A new Windows application was written, ARM Vision. It used this library, a communications library, and various other components to carry out the processing required. Data from the vision system was received on an RS232 serial line and processed by ARM Vision. The resulting compensation transformations were sent out along another serial line to the ARM controller; a 68000 processor system. This used the information to calculate the required PUMA positions and then sent the appropriate move commands to the PUMA controller.
6.4.2. Calibration

For motion compensation, the compound transformation relating the camera to the manipulator co-ordinate systems is used extensively. In theory the two component transformations can be measured directly but these measurements are unlikely to be sufficiently accurate. For this reason, it is advantageous that a self-calibrating method to determine this transformation is used. In an operational system, this procedure would be used to set up the system before it is launched, and may be repeated at any time on request, e.g. if the ROV has collided with the structure the camera or manipulator system may move slightly.

With the laboratory system, the self-calibration procedure was carried out before the trials took place. The target array was removed from the workpiece and fixed to the end flange of the PUMA. Since the PUMA tip position and the relative arrangement of targets were known, the positions of the targets as seen by the camera were used to create the camera-manipulator transformation matrix. This method allowed the control system to be calibrated to the accuracy of the camera system.

6.5. Experimental Test Rig and Procedure

As was stated in section 6.2, the PUMA manipulator was required to follow a moving workpiece in two dimensional space. The stereo vision system tracked the workpiece as it moved relative to the cameras and thus relative to the manipulator shoulder. The spatial data gathered by the vision system was then used to modify the path of the manipulator such that the manipulator end flange remained fixed relative to the target.

An inertial system comprising an extensible 'finger' connecting the manipulator to the target was used to measure errors between the desired (target) trajectory and the actual (manipulator) trajectory.

6.5.1. Hardware

A vertically mounted x-y plotter was used to move the workpiece. The targets consisted of contrasting circles fixed at varying depths on the workpiece, thus creating a three dimensional moving target as shown in figure 6.2. The workpiece was attached to the pen holder of the plotter.
The relative position between the target and the PUMA was measured with the moveable 'finger' shown in figure 6.3. This was mounted between the PUMA end flange and a rotary joint connected to the plotter pen holder. It comprises two potentiometers able to move ±30° in a rigid housing and giving a voltage according to the $x$ and $y$ displacement of the PUMA relative to the target. Any $z$-axis motion is taken up by a linear variable differential transducer (LVDT) with a stroke of 44mm.

These three voltages were read by a Data Translation 2801-A analogue-to-digital converter mounted in an Elonex 386 PC. The output voltages to the $x$-$y$ plotter were generated in the same PC by an Amplicon PC24 digital-to-analogue converter. Prior to each set of tests a calibration program was run, moving the target whilst holding the PUMA motionless. This gave a relationship between the voltages recorded and the length of misalignment that this represented.
The experimental equipment was logically divided into the four systems shown in figure 6.4. These systems are:

- **TARGET SYSTEM**: The target system PC drives the x-axis and y-axis of the x-y plotter independently. Each axis is driven in a separate sinusoidal pattern, varying in frequency and amplitude between tests. There is, additionally, a random phase difference between the start points of each axis. As each new motion is commanded, the PC logs the error voltage signals from the measurement probe and the requested current target position. All data is stored as ASCII text matrices in *.m files for subsequent off-line manipulation using the MATLAB mathematical software [Matlab UG 92]. For test one (section 6.5.2 details the tests), it also sends the new motion to the PUMA system controller, via an RS 232 serial link running at 19200 baud. For tests two and three, it does not send any information to directly control the PUMA system.

- **PUMA SYSTEM**: This is the PUMA manipulator and the Unimate controller. This controls the manipulator position according to the external information received via an external cable link either from the TARGET system PC or from the ARM system PC.
• ARM SYSTEM: This sends the position data to the PUMA system for tests two and three (section 6.4.2 details the tests). For test two it receives the true plotter position from the TARGET system. For test three, it receives the target position from the VISION system.

• VISION SYSTEM: This sends three dimensional target position information to the ARM system for test three. Although the targets move in a two dimensional plane, the cameras are set up such that they are not directed along an axis perpendicular to this plane. Thus, relative to the cameras, the targets are moving in three dimensions.
6.5.2. Experiments

The experiments were aimed at establishing the performance of the vision based motion compensation system. Using the system designations shown in figure 6.4, the following three tests were carried out:

1. Errors due to the PUMA system. This involved only the PUMA and TARGET system. The method was to send the identical position data from the plotter control PC to both the plotter and the PUMA manipulator and measure the position errors.

2. Errors due to the ARM and PUMA system. This involved sending the identical position data to the plotter and the ARM system PC and measuring the position errors.

3. Errors due to the VISION, ARM and PUMA system. This involved sending the position data to the plotter only. The VISION system then calculated the target positions, transmitted these to the ARM system PC which commanded the PUMA accordingly. Again the position errors were measured.

6.6. Calculations

As explained in Section 6.4.1., the x-y plotter was driven using independent sinusoids in x and y at a variety of different amplitudes (0 - 50mm), periods (8 - 45s) and relative phases (0 - 360°), whilst the plotter x-y positions and the PUMA errors were logged at a constant rate. Given these variations, it was necessary to find a means of normalising the data so that like could be compared to like. To this end it was decided to first normalise the various target output signals such that a single value of average speed could be calculated for the combined x and y signals over the duration of each test.

Given two perpendicular sinusoids in the x and y direction of amplitudes $A_x$ and $A_y$, periods $\tau_x$ and $\tau_y$ and with a phase lag of $\phi$ between them we can say that the distance travelled in the x and y direction is:

$$x = A_x \sin(\omega_x t)$$
$$y = A_y \sin(\omega_y t + \phi)$$

(6.1)

where $\omega_x = \frac{2\pi}{\tau_x}$ and $\omega_y = \frac{2\pi}{\tau_y}$ and $t =$ time in seconds
After differentiation, we get the velocities:

\[ V_x = A_x \omega_x \cos(\omega_x t) \quad V_y = A_y \omega_y \cos(\omega_y t + \phi) \]  \hspace{1cm} (6.2)

We can then get the instantaneous root mean square (RMS) velocity output signal:

\[ RMS_v = \sqrt{V_x^2 + V_y^2} \]  \hspace{1cm} (6.3)

\[ RMS_v = \sqrt{[A_x \omega_x \cos(\omega_x t)]^2 + [A_y \omega_y \cos(\omega_y t + \phi)]^2} \]  \hspace{1cm} (6.4)

It is apparent that, to get a single value representative of the average RMS speed, we need to integrate this instantaneous value to find:

\[ RMS_{\text{average}} = \lim_{T \to \infty} \frac{\int_0^T RMS_v \, dt}{T} \]  \hspace{1cm} (6.5)

Where \( T \) is the total time duration of the test (typically from 2 to 3 minutes). This is an elliptic equation and so is not analytically soluble. Instead, a computational method was used to calculate this solution for each test configuration.

In a similar (computational) manner, the average RMS error for each data set was calculated by averaging all the instantaneous RMS position errors.

For additional information, the lag between the target and PUMA was found by a simple minimisation of the function \( \lambda \):

\[ \lambda = \sum_{i=1}^{n} (f_i - e_i)^2 \]  \hspace{1cm} (6.6)

where \( f \) is the instantaneous recorded error signal.

The instantaneous calculated error signal, \( e \), is found from:

\[ e = A \sin \alpha t - D \sin(\omega t + \phi) \]  \hspace{1cm} (6.7)

where:

- \( A \sin \alpha t = \) target path;
- \( D \sin(\omega t + \phi) = \) PUMA path.

This was minimised in a least squares sense to solve for \( \phi \) and \( D \) where \( \phi \) is equivalent to a time lag in proportion to the period of the sine wave.
6.7. Results

The results are well illustrated by plotting the average RMS error of the PUMA end flange position relative to the plotter against the average RMS speed of the plotter during that test as shown in figure 6.5. The gradients and $y$-intercepts for this graph, using a first order least squares polynomial fit to this data, are shown in table 6.1. Also shown are the average lag figures calculated from the phase lag analysis. The main points of interest are the error due to lag, the intrinsic system error and the steady state error.

![Graph showing RMS error vs RMS speed](image)

**FIG. 6.5 - RMS ERROR VS RMS SPEED**

<table>
<thead>
<tr>
<th>System</th>
<th>Gradient (s)</th>
<th>Lag (s)</th>
<th>$y$-intercept (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUMA system</td>
<td>0.026</td>
<td>0.033</td>
<td>0.618</td>
</tr>
<tr>
<td>ARM/PUMA system</td>
<td>0.259</td>
<td>0.253</td>
<td>1.682</td>
</tr>
<tr>
<td>VISION/ARM/PUMA system</td>
<td>1.842</td>
<td>2.369</td>
<td>6.712</td>
</tr>
</tbody>
</table>

**TABLE 6.1 - GRADIENT, LAG AND INTRINSIC MEASUREMENT ERROR**

6.7.1. Error due to Lag

The line gradients in figure 6.5 are in units of seconds and relate directly to the lag of the PUMA position relative to the plotter. This was confirmed by the separate lag analysis (equation 6.6) which gave the values in table 6.1 that confirm the gradient values. It is
thought that the lag values are more accurate than the gradient figures, as it is slightly arbitrary what angle is drawn on the graph, and so these are the values discussed here.

The PUMA system results show a lag of 33ms between the plotter reaching a position and the PUMA arriving at that position. This lag is partly due to the serial communication delay between the PUMA and PC but also due to the physical limits of the system. i.e. the PUMA is updated every 28ms with a data package that takes 19ms to transmit. The remaining 14ms delay is due to physical acceleration and velocity constraints on the PUMA motion requiring a finite time to respond to a commanded position.

The ARM/PUMA system results show an increased lag of 253ms between the plotter reaching a position and the PUMA arriving at that position. This lag is partly due to an increased serial communication delay as there are more serial links (2 for the ARM/PUMA system, 3 for the VISION/ARM/PUMA system) each with its attendant transmission time from 19ms to 50ms. In addition, there is a great deal of calculation delay in the windows based ARM software amounting to 180ms. This would not normally be the case for this system using sensors. As the data handling was a quick add on, it passes the position data via the ARM system PC with its windows overhead. In a normal application, the sensing would go directly to the 68030 processor much reducing the lag - probably by more than 200 ms.

The VISION/ARM/PUMA system results show a hugely increased lag of 2369ms between the plotter reaching a position and the PUMA arriving at that position. Again, this lag is partly due to the increased serial communication delay and includes the ARM system delay mentioned before. In addition, there is a large computational overhead, with the VISION system taking over 2000ms to extract target position information from a stereo video image. The VISION system was originally designed as simply a viewing system and the target acquisition was an add on for this project. It is certainly possible to have much faster visual target acquisition systems.

It can be seen that the derived lag figures in table 6.1 (the gradient) tally closely with the directly calculated figure for lag giving confidence in the data analysis methods. The exception is for the Vision case when the data becomes much noisier due to the stepwise manner in which the PUMA is directed to move. This arises because the resolution of the vision system causes apparent steps in the target position of about 7mm as explained in section 6.7.2.
6.7.2. Intrinsic error of the systems

The y-intercepts from figure 6.5 indicate the intrinsic error of the various systems. This gives a resolution for the VISION system of 6.7mm and, for the measurement system (PUMA system), 0.6mm i.e. they are an order of magnitude apart.

For the measurement system, this error would appear to be largely due to noise as the potentiometers are fairly low precision. When the calibration data is tested on itself (that is when the calibration polynomials are refitted to the calibration data), the mean errors are recorded as varying from ±0.4 to ±0.6mm for the various calibration data sets collected at different times. This is ample for the measurement of a vision system that has a resolution an order of magnitude greater.

The intermediate value of 1.68mm for the situation of the ARM system may, in part, be due to increased calculation leading to rounding errors. However, the bulk of the difference is because the ARM system was receiving updates at only an 8 Hz sampling rate. Given that the ARM system was sampling at only a quarter of the rate of the PUMA system, then, for all but the motionless case, there will be an intrinsically larger error as the targets will have moved further before the PUMA manipulator can respond.

6.7.3. Steady state error

Figure 6.6. shows the RMS errors of the recorded data (PUMA - test one, ARM - test two and ARM-VISION - test three) Vs descending order of average distance travelled. In each case a calculated value has been included to show the RMS error that would have been recorded had the PUMA remained motionless. i.e. the error with no feedback system. This is referred to as NO-MOTION.

We see that as the plotter travels further from the centre point, all the errors generally get larger. This has to be true for the NO-MOTION case as this has been used to define the order in which the data are shown.

In the other cases the errors also grow larger as distance travelled by the plotter increases, but they appear to grow larger at a lesser rate than the overall distance moved. It seems likely that, if the average distance moved was much larger than the system resolution (not true for any of these cases), then the trend would continue of vision errors remaining more or less constant at a particular speed, but becoming a much smaller proportion of the target motion. i.e. there is a maximum steady state RMS error which will be reached.
6.7.4. ARM system operation

Results of the tests showed that the complex transformation algorithms in ARM work successfully and that, despite their complexity, they could be calculated at a reasonably high speed although slowing down the system. In an operational system the overheads could be reduced further by integrating the control system into the manipulator controller (i.e. on the ROV in a sub-sea system) as outlined in section 6.7.1. This would remove delays due to communication along serial lines and it would be practical to carry out the calculations at a rate between 10 and 100Hz, as required to achieve stable operation.

6.7.5. VISION system operation

The experimental system was limited in performance only by the very slow update of the vision system (over 2s). As an interim measure, the large PUMA moves produced by the slow camera update were smoothed by the addition of special demand shaping algorithms within the ARM controller, producing more controlled transitions between tip positions and preventing fast and potentially damaging transitions of the PUMA end flange from point to point.
6.8. Conclusions

The ARM VISION Control System worked successfully and at the speed required of it. The main conclusions drawn are:

1. The VISION system and ARM transformation algorithms can successfully calculate, at the speed required, the manipulator modifications required to hold a target position with a stationary or slow moving base. For these cases, tip errors can be of the order of 6-7mm due mainly to propagation of vision system resolution errors (currently 1-2mm in \(x\) and \(y\), 3-4mm in \(z\)).

2. The cameras for this test were 2 metres from the target. To be able to double this distance without creating worse tip errors will require a doubling of the camera resolution. i.e. twice the resolution in both the \(x\) and \(y\) axis resulting in four times the number of pixels. In the real environment, the likely target distance will be from two to four metres. This distance will be affected by visibility limitations due to murky water.

3. The errors occurring are consistently linear (figure 6.5) as test speed increases. This indicates that there is a constant (speed independent) time delay between target motion and manipulator response due to information processing and serial communications. Thus, to reduce errors at higher target speeds, requires faster information processing.

4. Within the scope of these tests (of limited amplitude) it can be seen (figure 6.6.) that the average RMS error is reduced by a factor of about a half by the addition of the vision/motion compensation system. To achieve larger amplitude or higher speed tests, a more robust test bed would be required than the \(x-y\) plotter which was working near its limits in terms of both dimension and load carrying.

This series of tests has shown that a stereo vision based system has the potential to significantly reduce motion-induced manipulator-tip errors, and therefore forms the basis of a practical motion compensation system. However, improvements would be required to the vision system resolution and update rate to be able to track more distant targets at an acceptable rate. A revised system architecture is required to reduce the number of serial communication links. Even with these improvements, a real system is unlikely to improve on errors of several mm and so a compliant force solution will still be required.
7.1. Introduction

This chapter details the design process leading to the current (1997) version of the compliant device (CD). This involves:

- A description of the previous design for a compliant device by Dr. Q. Wang (prior to the start of this project).
- A description of the physical task constraints.
- A description of the initial design choices examined.
- A description of a new design based on the need for directional stiffness within the compliant device.
- A conclusion of the current state of the compliant device.

7.2. Previous Design of a Compliant Device

7.2.1. Physical Design

In 1992, Dr. Q. Wang completed his Ph.D. work which included the design and build of an instrumented, multiple dof, compliant device developed for sub-sea inspection [Wang 92]. This was a four dof system with one translational dof and three rotational dof as illustrated in figure 7.1.

The compliance consists of two circular disks held apart by springs attached to the plungers of linear potentiometers. As the disks move together, the plungers retract and the recorded voltage from the potentiometer changes. By combining the readings from all four potentiometers, the relative angle and displacement of the lower plate relative to the upper plate may be calculated. A mock probe was attached beneath the lower of the two plates.
In the co-ordinate system shown in figure 7.1, the compliance will heave along the $z$ axis and roll, pitch and yaw about the $x$, $y$ and $z$ axes respectively. In use, roll and pitch allow compensation for misalignment between the probe base and the workpiece. Yaw allows rotation of the probe to enable alignment with a weld edge. Heave compensates for error in the probe height above the workpiece.

N.B. The co-ordinate system used here is not the same as any used for the compliant devices developed in this thesis.

**7.2.2. Limitations of the Previous Design**

Although proving the concept of the instrumented compliant device, there were several limitations to this design:

1. It had only four dof and so is rigid in the $x$ and $y$ translations. As will be seen in section 7.5, this is insufficient for the real task of inspection.

2. The geometry of the attachment of the manipulator to the compliant device does not allow it to reach into the tightest angles as the height from the probe base to the top of the manipulator wrist (the black section in figure 7.1) is too great. To do this would require the manipulator to be attached to the side of the compliance (see section 7.4).

3. The measurement devices (linear potentiometers) would not survive the corrosive marine environment. This is generally true of laboratory sensing devices and any move to the marine environment requires an investment in robust sensors that are generally much more expensive and bulky than their land based counterparts.
7.3. Physical Task Constraints on the Compliant Device Design

As was detailed in section 3.4, the ultimate goal of this work is to be able to inspect the two sides (roots) of a weld within a 30° cylinder intersection. The physical constraints are the physical geometry of the weld to be inspected and the performance of the ACFM crack detection probe.

7.3.1. Constraints of The ACFM Crack Detection Probe

The ACFM probe operates by using coils to induce and detect a surface magnetic field that fluctuates according to the surface properties. The probe coils are positioned as shown in figure 7.2. The probe has an optimum scanning speed of about 10 mm/s and will scan a width of approximately 20mm [Lugg 96]. The probe is capable of scanning in the directions shown by the arrows in figure 7.2. In general, scans are made along the toes of the weld (the intersection between weld metal and original metal) although, for a weld wider than 20mm, one or more intermediate scans should be made along the weld surface.

![FIG. 7.2 - LOCATION OF COILS WITHIN THE ACFM PROBE](image-url)
7.3.2. **Constraints of Weld Geometry**

The geometric constraints of this situation are illustrated in figure 7.3 and are based on:

1. From an examination of oil rig construction data, it is apparent that the larger tubular steel member, the chord, will have a thickness, $l$, in the range $20\text{mm} \leq l \leq 50\text{mm}$ [API 93].

2. The American Welding Standards determine the weld size appropriate for the joining of the chord and the brace (the smaller tubular member) [API 93]. For this acute angle configuration, these standards define the weld depth, $L$, as $L = 2l$.

3. The worst case, from the inspection point of view, exists when $L$ and the chord/brace internal angle, $\theta$, are a minimum ($\theta = 30^\circ$). This results in the smallest possible value for the access depth, $h$. From points 1 and 2, it is apparent that $L_{\text{min}} = 2 \times 20\text{mm} = 40\text{mm}$.

As figure 7.3 illustrates, these factors give a weld depth of 40mm. Therefore, for the probe to be able to reach either side of the weld, the probe must enter a truncated wedge (the red box in figure 7.3) only 23 mm high at the narrowest point. This requires a low profile probe with the manipulator arm located behind it and not above it. Clearly, a more complex compliant device is required along with a different method of attachment.

![FIG. 7.3 - SECTION THROUGH BRACE AND CHORD SHOWING KEY DIMENSIONS](image-url)
7.4. Design of the Compliant Device

7.4.1. Choice of the Compliant Device Assembly Layout

Figure 7.4 shows an intuitive probe/compliant device/manipulator assembly that can enable such a tight angle weld to be reached. This is the basic pattern used in all the subsequent compliance designs. This enables:

1. A large diameter manipulator arm to inspect a weld toe where the frontal clearance is less than 23mm.

2. The manipulator arm to be centred within the gap created between the chord and the brace.

7.4.2. Choice of a Force/Torque Transducer

The method of measuring force and torque is that of measuring the displacement of a system of known stiffness. For this project, it was decided that a full six dof force/torque transducer would be required to enable the range of inspection that was envisaged. This required a device that could touch down on a surface and then track either forwards or sideways until limits were detected. This requires all three translational dof. To enable angular alignment with all three axes also requires all three angular dof.
There are two main routes for measuring six degrees of force and torque:

1. This can be done on a macro scale, as illustrated by Dr Wang's compliant device in section 7.2, in which the relatively large displacement of a compliant material (a spring in that case) by a force is measured by an electrical transducer. From the measured displacement and the known stiffness of the compliance, the force may be calculated. This puts limitations on the design of the compliance as it has to accommodate the measurement devices. It is also difficult to build such a device that may measure all six dof as the compression of a compliance in a particular direction should not be affected by a simultaneous compression taking place to another compliance along an orthogonal direction. A device that achieves this state is referred to as a 'de-coupled' device. Attempts by another group resulted in a bulky and complex device that gave de-coupled force and torque measurements by this method [Sinha 93]. It would, however, be very hard to use for an underwater application because of its bulk and the difficulty of making it sufficiently durable for the harsh environment.

2. The other effective method of force measurement is to measure the displacement of a stiff material, such as steel, using strain gauges. This results in a force measurement device that is very stiff and so an additional compliant part is required to produce the compliant device. Separating the force transducer from the compliant device enables a much wider choice of compliant design. This is generally the method used by commercial force sensors and so there is a wide range of available devices.

The second option of a separate force/torque transducer was chosen because this allowed more freedom in the design of the compliance and because very small transducers were commercially available that offered the required force range and overload protection. The chosen device is not designed for sub-sea work and development work would need to take place before it could be used underwater. The chosen transducer is manufactured by the American company Associated Technologies Incorporated and the specifications are in Appendix G.
7.4.3. Location of the Force/Torque Transducer

With the assembly of the probe/compliant device as for figure 7.4, there are only really two possibilities for the placing of the force/torque transducer. It can either go downstream or upstream of the compliant device with respect to the manipulator. This is illustrated in figure 7.5.

The key considerations that arise when considering the relative merits of these two positions are:

1) If the force/torque transducer is downstream of the compliant device (figure 7.5.a) then the compliance sits between the actuator (the manipulator arm) and the sensor (the force/torque transducer) and it is said to be dynamically non-co-located [Eppinger 89]. This means that "the actuator and sensor can vibrate out of phase." This can lead to control instability and is likely to reduce the system bandwidth. To have the transducer upstream (dynamically co-located with the manipulator) would not guarantee stability, but would be likely to increase the frequency at which an out of phase vibration would occur. Other factors of interest, with regard to a downstream sensor, are:

- A downstream sensor will affect the compliance with its own mass or buoyancy.
- In the event of the compliance failing, the sensor is more likely to be damaged or lost.
- The sensor cable will run in parallel to the compliance and add its own stiffness properties to those of the compliance.

FIG. 7.5 - POSSIBLE FORCE/TORQUE TRANSDUCER LOCATIONS
2) If the force/torque transducer is upstream of the compliant device (figure 7.5.b) then it is likely to be outside of the normals to the footprint of the ACFM probe as illustrated in figure 7.6. The probe in this figure has three "feet" which actually make contact with the surface under inspection and from which surface normals are drawn. The ramifications of this are two-fold. First, although the full six degrees-of-force and torque may be calculated in either case, only in the downstream case will the torque reading directly relate to the type of contact giving a positive or negative reading according to which probe "foot" touches first. In the upstream case, a few calculations are necessary to decouple the forces and torques into true contact forces. When the contact is at more than one foot at once, the calculations become more complex and the effects of signal noise and friction more pronounced. This leads to the second ramification of the downstream/upstream choice. To decouple the contact forces with any degree of accuracy requires accurate geometric knowledge of where the force/torque transducer is with respect to the contact feet. When the transducer is downstream of the compliant device, this geometry is fixed. When it is upstream, the geometry varies as the compliant device is compressed. A key problem in the control of this system is knowing these geometrical variations. To have the transducer upstream will create a vicious circle which arises because, to know the geometrical variation of the compliant device requires good knowledge of the contact forces. However, to know the contact forces requires good knowledge of the geometry.

FIG. 7.6 - POSSIBLE LOCATIONS OF ACFM PROBE FORCE TRANSDUCERS
As a result of these factors, the decision was taken to use the downstream position of the force/torque transducer. The potential problems of sensor non-co-location were seen as less problematic, especially at low manipulator velocity, than the difficulty of de-coupling noisy force/torque data.

The use of three feet on the probe enables full contact to be ensured on a curved surface regardless of the orientation of the probe relative to the surface. The height of the feet was selected so that the base of the probe would remain clear of the cylinder surface in the worst case scenario; a cylinder of only 400mm diameter.

Figures 7.6 and 7.7 illustrate another design feature; the use of return legs at the back of the probe. This extended the length of the probe and enabled the force/torque transducer to be more centrally positioned with respect to the contact feet. The increase to the base contact dimensions of the probe enabled greater sensitivity to contact torque due to the increased moment arm.

For the experimental work in this thesis, a full scale aluminium mock up of the probe was constructed with two brass hemi-spherical feet at the back of the return legs and a brass contact plate at the front of the probe. To the back of this was attached the force/torque transducer. Between the transducer and the manipulator arm could be fitted a range of different compliant devices. This assembly is illustrated in figure 7.7.
7.4.4. Initial Compliant Device Shapes

At this stage, a compliant device was required that enabled other design work to continue such as developing the artificial neural network method of modelling the compliant device (see chapter eight). The first devices were solely designed as being relatively simple to manufacture and the properties that the compliant device required to perform an inspection were not addressed.

7.4.4.1. Diamond Shape Compliant Device:

The first compliant device developed consisted of two simple diamond shapes that were attached between the force/torque transducer and the manipulator attachment plate. These were chosen simply because they were so simple to manufacture and meant that a real compliant device could be fitted while early work on other sections of the project were also carried out. The diamonds were cast from 60% shure hardness polyurethane and bonded to steel interface plates (shure hardness is a standard measure of the hardness of a polymer). One of these plates was attached to the force/torque transducer and the other became the manipulator attachment plate.

Figure 7.8 illustrates the arrangement of the compliant device and its approximate dimensions. The tool co-ordinate frame is attached to the centre of the force transducer and is used in all subsequent designs.

![FIG. 7.8 - DIAMOND SHAPED POLYURETHANE COMPLIANT DEVICE](image)
Some time was spent mapping the position/stiffness characteristics of this device. The first diamond device split rapidly. The sharp corners throughout the diamond shape led to high stress concentrations under load and hence to rapid failure. A second moulding also failed rapidly and so this form was discontinued. In addition, it was noticed that the rotation about the $y$ axis, was not stiff enough when the probe was dragged sideways along a weld edge in the $x$ direction. This led to large deflections of the compliant device and would have required large motions of the manipulator to keep the same aspect of the probe in line with a weld.

7.4.4.2. Hollow Spherical Compliant Device:

A hollow sphere with a neck at each end had been produced by Technical Software Consultants as a possible shape to offer a useful compliant device. This was relatively simple to manufacture and is robust as it has gentle curves and hence little stress concentration. The bolts running through the ends of the extended sphere (figure 7.9) join the compliant device to the attachment plates. The bolts themselves have a hollow through their centres to allow for pressure equalisation underwater. For initial experimental work, the sphere was fitted between two plates as seen in the photograph (figure 7.9).

![Plan of (near) spherical, hollow compliant device](image)

**FIG. 7.9 - HOLLOW SPHERE COMPLIANT DEVICE SCHEMATIC AND PHOTOGRAPH**
The spherical device was also employed by Technical Software Consultants using a Stewart platform arrangement of linear transducers between two steel plates [Broome 94]. These measured the relative displacement of the two plates with the compliant rubber sphere between them. This was not a successful device as the available underwater transducers were too long for the size of compliance and the length of travel that was required. However, with a re-designed compliance and the constant improvement in availability of sub-sea components, this may still provide a solution in the future. A photograph and schematic of this arrangement is shown in figure 7.10. In the example of figure 7.10, the exterior of the neck of the spherical compliance is filled with more polymer in an attempt to stiffen the compliant device. This makes the device a cylinder in external appearance.

FIG. 7.10 - INSTRUMENTED COMPLIANCE AS A FORCE TRANSDUCER
7.5. Design of the Compliant Device Based on Stiffness Requirements

7.5.1. The Need For Directional Stiffness

To assess the stiffness requirements of the compliant device, it was necessary to examine the tasks that the probe would be expected to undertake under compliant force control. The basic design criteria used is based on the idea of position and force controlled orthogonal degrees of freedom [Mason 81]. This requires that, for axes limited by surface contact, a compliant response is required (low stiffness) whilst for axes able to be travelled without restraint, a stiff response is required (high stiffness). In other words, a stiff compliance is required along, or about, any axis which is unconstrained by the environment.

It should be noted that these constraints vary during the approach phase. As the probe makes contact, constraints come into existence. When the inspection position is reached, additional constraints come into existence. The compliant device is designed to achieve the appropriate response for the key section of the inspection task; moving sideways along the weld. It is assumed that the system will still be operable during the less critical parts of the inspection despite a sub-optimal compliant response.

Figure 7.11 illustrates the steps required for the probe to make contact with the surface, move into a tracking position and then track the weld for the purpose of inspection. These break down into the following motions in tool co-ordinates:

1. Touch down onto the surface to be inspected. The key motion in the +y direction is a constrained task requiring a low stiffness along the y axis.

2. Move across the surface until the weld is encountered. For all the surface contact, the motion is constrained in its rotation about the x and the z axis and so these are low stiffness rotations. When the weld is encountered, the motion in the +z direction is also constrained and therefore a low stiffness is required along the z axis.

3. Track along the weld. This motion in the ±x direction is unconstrained by the environment and so requires a high stiffness along the x axis. Equally, as the probe tracks sideways along the weld, rotation about the y axis is unconstrained (or nearly so) and so requires a high stiffness about y axis. Step 3 defines the constraints for which the compliant device is designed.
These constraints allow a simple, workable framework within which to design the compliant device but are a simplification of the truth. In reality, unconstrained directions will have irregularities in the metal giving occasional constraints i.e. weld spatter on the chord surface would block the theoretically smooth traverse of the probe in the -x direction. Additionally, friction on the feet may easily have a component acting to constrain the probe in a theoretically unconstrained direction. Occasionally, an unusual geometry may create an unconstrained direction which is theoretically constrained i.e. reaching the edge of a flat plate inspection.

In summary, the relative stiffness required for each dof is tabulated in table 7.1. The terms low and high stiffness are used to give a relative value for the different axes. The actual stiffness of the compliant device is dictated by the materials used in its construction and the configuration or geometry of its component parts.

<table>
<thead>
<tr>
<th>Translations</th>
<th>rotations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
</tr>
<tr>
<td>required stiffness</td>
<td>high</td>
</tr>
</tbody>
</table>

TABLE 7.1 - RELATIVE STIFFNESSES REQUIRED BY THE COMPLIANT DEVICE
7.5.2. Designing for Directional Stiffness

To achieve these different relative stiffness values, a simple idea was chosen. This was to use a construction in which springs or linear stiffness members were so arranged as to produce the desired stiffness for the compliant device. The reasons for this were threefold:

1. It was simple to create mathematical models of the compliant device because the basic elements were so simple. The geometry of the compliant device could therefore be rapidly varied until the desired stiffness was achieved.

2. It was straightforward to create a real compliant device, using tensioned springs, that could simply have its geometry adjusted and hence its stiffness.

3. It was thought that it would be relatively straightforward to create a polymer form of the final compliant device.

The first step was to create a variable geometry mathematical model of the compliant device within Matlab. The code for this is included in Appendix H. This was a vector model of the springs and plates in space. The local stiffness in each direction was found by translating or rotating the outer attachment plates in space by a small amount relative to the inner attachment plate (see figure 7.12). The resultant force could then be calculated and hence the stiffness as a function of force and displacement along and around the principle axes.
The basic model used the springs aligned with the x axis as shown in figure 7.12. This was chosen as it appeared likely to give the desired relative stiffness. For each attachment plate, the four patterns of spring placement shown were used.

The arrangement was always symmetrical either side of the inner plate. This resulted in sixteen possible combinations of inner and outer attachment plate connection, each with their associated directional stiffnesses.

The relative translational and rotational stiffness for each dof is detailed in tables 7.2 and 7.3. The values are normalised to a range of 0 to 1 to indicate whether the relative stiffness is high or low. It is normalised for the translational and rotational terms independently because the units are not equivalent. In addition, high values are written as red text and low values, blue text, for ease of visualisation.

From an examination of these results we can see two main things:

1. In table 7.2, the translations, every single case matches the broad requirement for stiffness. The high stiffness in the unconstrained (x) direction is most important and so, the cases lying on the diagonal of the table (the shaded boxes) have been chosen as the most suitable. The diagonal values equate to the cases where the springs are parallel to one another as the inner plate has the same pattern as the two outer plates. This makes sense intuitively.

2. In table 7.3, the rotations, the cases in the four shaded boxes match all three stiffness criteria. When combining these cases with those selected from table 7.2, it is only the darkly shaded box, outlined in yellow, that meets the criteria of both the tables. This is, therefore, the initial pattern chosen to use in the compliant device.

**FIG. 7.12 - ATTACHMENT PATTERNS FOR THE TRANSVERSE SPRING COMPLIANT DEVICE**
**Table 7.2 - Normalised Relative Stiffness of Translational Compliant Device Axes with Varying Geometry Compliant Elements**

<table>
<thead>
<tr>
<th>Translation Stiffness Axes</th>
<th>Desired Stiffness</th>
<th>Normalised Stiffness (range 0 - 1) for Three Translational Degrees of Freedom</th>
<th>Inner Plate Spring Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>high</td>
<td>1.00 0.94 0.94 0.89</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>y</td>
<td>low</td>
<td>0.00 0.00 0.06 0.06</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>z</td>
<td>low</td>
<td>0.00 0.06 0.00 0.00</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>x</td>
<td>high</td>
<td>0.94 1.00 0.89 0.94</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>y</td>
<td>low</td>
<td>0.00 0.00 0.06 0.06</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>z</td>
<td>low</td>
<td>0.06 0.00 0.06 0.00</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>x</td>
<td>high</td>
<td>0.94 0.89 1.00 0.94</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>y</td>
<td>low</td>
<td>0.06 0.06 0.00 0.00</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>z</td>
<td>low</td>
<td>0.00 0.06 0.00 0.06</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>x</td>
<td>high</td>
<td>0.89 0.94 0.94 1.00</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>y</td>
<td>low</td>
<td>0.06 0.06 0.00 0.00</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>z</td>
<td>low</td>
<td>0.06 0.00 0.06 0.00</td>
<td>![Pattern]</td>
</tr>
</tbody>
</table>

**Table 7.3 - Normalised Relative Stiffness of Rotational Compliant Device Axes with Varying Geometry Compliant Elements**

<table>
<thead>
<tr>
<th>Rotation Stiffness Axes</th>
<th>Desired Stiffness</th>
<th>Normalised Stiffness (range 0 - 1) for Three Rotational Degrees of Freedom</th>
<th>Inner Plate Spring Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>low</td>
<td>0.00 0.06 0.06 0.00</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>y</td>
<td>high</td>
<td>1.00 0.94 0.94 0.89</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>z</td>
<td>low</td>
<td>0.94 0.94 0.94 0.89</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>x</td>
<td>low</td>
<td>0.06 0.00 0.14 0.02</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>y</td>
<td>high</td>
<td>0.34 0.36 0.32 0.34</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>z</td>
<td>low</td>
<td>0.94 1.00 0.89 0.94</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>x</td>
<td>low</td>
<td>0.06 0.14 0.00 0.02</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>y</td>
<td>high</td>
<td>0.94 0.89 1.00 0.94</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>z</td>
<td>low</td>
<td>0.34 0.32 0.36 0.34</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>x</td>
<td>low</td>
<td>0.00 0.02 0.02 0.00</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>y</td>
<td>high</td>
<td>0.34 0.34 0.34 0.36</td>
<td>![Pattern]</td>
</tr>
<tr>
<td>z</td>
<td>low</td>
<td>0.34 0.34 0.34 0.36</td>
<td>![Pattern]</td>
</tr>
</tbody>
</table>

Outer Plate Spring Pattern: ![Pattern]
7.5.3. Compliant Device Number 3 - Transverse Spring

The adopted transverse spring compliant device is illustrated in figure 7.13. A photograph is shown in figure 7.15. Certain practical additions were made to the original idea to allow for later design ideas or more complex analyses leading to refinements. The engineering drawings for this design are included in Appendix I.

1. The compliant device constructed has additional positions for re-location of the springs on the inner and outer plate (an array of twenty points shown in yellow and blue in figure 7.13). This was simply to allow the chosen spring geometry to be placed in a wider variety of positions relative to each plate.

2. The outer plates may be positioned further apart to allow for longer springs. The pre-tension in the springs is provided by bolts through the outer plates which are drilled at the tips to connect to the springs as shown in figure 7.14. The bolts then protrude to a greater or lesser degree through the plate according to the position of a retaining nut and a locking nut.
7.5.4. Designing for Directional Stiffness - Phase II

The next stage of this development, was to model the local stiffness patterns under a range of pre-loads. These pre-loads would be the expected operating position for the probe relative to the manipulator during inspection.

It was envisaged that there should be some vertical compression into the workpiece along the $y$ axis to give a measurable contact force. In addition, it was thought that there would want to be some rotation about the $x$ axis to keep the front foot of the probe on the workpiece surface as shown in figure 7.16. These offset positions would become the "neutral" starting point for compliant surface inspection. It was expected that such
offsets would have a considerable impact on the stiffness properties of the compliant device.

To examine the effects of a pre-load, the computer model developed in section 7.5.2 was reconfigured to match the actual device operating position. That is, the model was adjusted to include the expected permanent position and rotation offset described above. The code for this is included in Appendix H. For a variety of different inner and outer attachment plate spring connections, the local stiffness was calculated whilst the outer attachment plates were translated and rotated relative to the inner attachment plate.

![Diagram of pre-loads](image)

**FIG. 7.16 - EXAMPLE PRE-LOADS APPLIED TO THE COMPLIANT DEVICE**

The incremental moves were 2mm translations along the $y$ axis and $5^\circ$ rotations about the $x$ axis as shown in figure 7.16. The key results are shown as tables of graphs in tables 7.4 and 7.5.

The tables show the following:

1. Each table shows a representative sample of the results for two different inner plate spring configurations and four outer plate spring configurations.

2. Within each table, each column corresponds to a different rotation about the $x$ axis.

3. Within each table, each row corresponds to a different outer plate spring configuration.

4. Each individual graph within a table shows how the relative stiffness of the compliant device varies as the outer plates move along the $y$ axis relative to the inner plate.

5. Each graph shows the variation of the translational stiffness with solid lines, and the rotational stiffness with dashed lines.
<table>
<thead>
<tr>
<th>Inner Plate Springs</th>
<th>Outer Plate Springs</th>
<th>Rotation About the x Axis (deg)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
<td>0</td>
</tr>
</tbody>
</table>

**Key:**
- **x** translation stiffness
- **y** translation stiffness
- **z** translation stiffness
- **x** rotation stiffness
- **y** rotation stiffness
- **z** rotation stiffness

**Graph axes title and units:**
- **x**-axis: offset translation in *y* direction (mm)
- **y**-axis: normalised relative stiffness calculated separately for the translation and rotation data

**Table 7.4 - Variation of Local Stiffness as Translation in *y* and Rotation in *x* Take Place for a Range of Inner and Outer Plate Geometries**

121
TABLE 7.5 - VARIATION OF LOCAL STIFFNESS AS TRANSLATION IN Y AND ROTATION IN X TAKE PLACE FOR A RANGE OF INNER AND OUTER PLATE GEOMETRIES
From an examination of the graphs in tables 7.4 and 7.5, the following conclusions may be drawn:

1. In all cases, as the translational offset in \( y \) increases, so the \( y \) direction local stiffness increases (the solid blue line). Critically, for a translation greater than about 1.5 mm, the local \( y \) stiffness exceeds the local \( x \) stiffness which is counter to our design goals. It is therefore noted that, ideally, a translational \( y \) offset of less than 1.5 mm will be used.

2. In some cases, and in particular when the inner and outer plate patterns are widely spaced, we see an increase in the \( x \) rotational stiffness (dotted red line) as the offset angle of the device is increased. Again, this is counter to our design goals which are to have a high rotational stiffness about \( y \) compared to those about \( x \) and \( z \).

3. It can be seen that, for the cases where the inner plate patterns are widely spaced, the \( y \) rotational stiffness is relatively high, backing up the findings of section 7.5.2. If the outer plate spacings are narrow, then the relative stiffness of the \( x \) rotation term as the pre-load is increased becomes less of a factor (compare rows 5 and 6 of table 7.5). This results in a variation from the conclusions of section 7.5.2 and would indicate that the fourth term of row three in table 7.3 should be used instead of the third term (the outer plate pattern changes from wide to narrow). This only fractionally alters the relative stiffness of the terms within that table and so appears an acceptable result.

7.5.5. Compliant Device Number 4 - Transverse Polymer Spring Element

The final prototype to be manufactured simply replaces the spring elements with polymer cylinders. Each cylinder has a thread set into each end to enable it to be attached to the inner and outer plates in a manner analogous to the transverse spring compliant device. The basic design is fully described by figure 7.13 and the engineering drawings of the mould to cast the polymeric elements and the slight variations to the attachment plates are included in Appendix J. The means by which these polymeric elements are joined to the inner and outer attachment plates is illustrated in figure 7.17.
This compliant device was built to take the concept to its next stage which is to have a robust underwater compliance proof against the marine environment. However, it was not used as a compliant device within the tests carried out in this thesis. It also became apparent that a severe difficulty in the manufacture of such devices is preventing the polymer splitting. Once a surface crack appears, it spreads very rapidly leading to catastrophic failure. On all polymeric devices made to date, failures have occurred at edges and sharp corners indicating that stress concentration is leading to material failure. It is therefore considered that a continuously curved and, therefore, relatively complex shape will be required eventually.

7.6. Conclusions

In this chapter we have seen some very real geometric constraints lead towards a particular design and it is worthwhile reflecting on this process.

The tight angle that the probe is required to reach, combined with the decision to use compliant force control, led directly to the force transducer being placed downstream of the compliance and thus dynamically non-co-located with the actuator. This results in a probe that will only be able to track at a low speed that is dependant on the properties of the compliance. It also resulted in a probe for which the recorded torque reflects the orientation of the probe and for which force and torque measurements do not need to be decoupled. These two outcomes are entirely in keeping with the design parameters.

The use of compliant shapes that had been selected for ease of manufacture proved wasteful, although they did offer clues as to what response was not wanted. It was only when the practical constraints of the required behaviour were considered that a
reasonable and workable design was achieved. In addition, the use of polymer devices, with the idea of transferring to a sub-sea environment, showed some of the problems of that as a material. It is prone to fatigue failure and needs well considered design to achieve robust results. In addition, it is speculated that the inherent hysteresis in polymer products will lead to a reduction in accuracy of position prediction using artificial neural networks as outlined in chapter eight.

The process of modelling the system to optimise the arrangement of springs on the transverse compliant device showed how important it is to consider the model within its normal operating range. The optimal arrangement for the compliant device in its neutral horizontal position (section 7.5.2) gave considerably different results from in its operating position (section 7.5.3).

In summary, the design process outlined in this chapter resulted in a useful laboratory compliant device through a process that followed this structure:

1. Examine previous compliant devices used in this field.
2. Examine the physical task constraints.
3. Build the first prototype six degree-of-freedom compliant device to get a sense of what is required and to be able to develop other aspects of the project.
4. Use task constraints and mathematical modelling to find an optimal compliant device based on the perceived stiffness requirement.
5. Build a prototype laboratory compliant device (the transverse spring model).
6. Build a polymeric compliant device as a first step towards achieving a usable sub-sea compliant device.

As will be seen in the remainder of this thesis, the transverse spring compliant device fulfils the design requirements. It has six dof, possesses the desired directional stiffness and enables the manipulator arm to place the probe when attached behind it instead of above it hence allowing access to tighter welds.
8.1. Introduction

This chapter describes the implementation and testing of an artificial neural network to model the compliant device used for inspection. This comprises:

- An introduction to neural networks and a description of the operation of the chosen type of neural network.
- A description of the experimental method used to gather data for training the artificial neural networks.
- A description of the neural network training procedure.
- A description of the neural network validation procedure including a comparison with a more conventional spring element model.
- Conclusions.

8.2. Choosing an Artificial Neural Network Structure

8.2.1. Artificial Neural Networks

The purpose of the artificial neural network in this thesis is to mathematically model the relationship between the forces acting on the probe, and the position of the probe relative to the manipulator arm. Such a model will need to be solved every 28 ms because the control loop on a PUMA 560 has a clock rate of about 37 Hz. In many force control applications, even faster clock rates are used. With more traditional methods such as Finite Element Analysis (FEA), such a model would typically take several minutes to solve on a PC. In addition, the highly coupled relationship between the forces acting on the probe, and the position of the probe relative to the manipulator arm is a non-linear mapping problem and as such is well suited to a neural network solution [Irwin 95].
It is worth looking at some of the key artificial neural network structures and their primary uses. A useful definition of neural computing is that it is... “a 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 89]. The reality is far short of artificial intelligence, but these tools enable many previously intractable tasks such as non-linear modelling or pattern recognition [Irwin 95] [Lippman].

There are many types of artificial neural networks that have been developed for a variety of different purposes. Some of the key types are shown in table 8.1.

<table>
<thead>
<tr>
<th>Type of Network</th>
<th>Primary purpose/application</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceptron (single layer)</td>
<td>simple pattern recognition problems</td>
<td>[Rosenblatt 61]</td>
</tr>
<tr>
<td>Linear</td>
<td>signal processing, control and prediction</td>
<td>[Widrow 85]</td>
</tr>
<tr>
<td>Back Propagation</td>
<td>function approximation</td>
<td>[Rumelhart 86]</td>
</tr>
<tr>
<td>Radial Basis Function</td>
<td>function approximation</td>
<td>[Chen 91]</td>
</tr>
<tr>
<td>Associative Learning</td>
<td>local function approximation</td>
<td>[Hebb 49]</td>
</tr>
<tr>
<td>Self Organising</td>
<td>categorisation</td>
<td>[Kohonen 87]</td>
</tr>
</tbody>
</table>

**TABLE 8.1 - PRINCIPAL ARTIFICIAL NEURAL NETWORK TYPES**

For our purposes of non-linear function approximation, back propagation or radial basis functions are most suited. The radial basis function is a faster training alternative, but has the problem that, due to the manner in which each node represents only a local area, can leave gaps within its learnt space. It does not necessarily outperform the back propagation network [Tan]. In addition, at the time of starting this work, it was not available in commercial toolboxes and so back propagation was chosen.

### 8.2.2. Back Propagating Artificial Neural Networks

The chosen artificial neural network is a back propagating network with one hidden layer. The back propagating network has been one of the most widely used architectures and, with only a single hidden layer, is able to approximate any continuous function [Cybenko 89].

The structure of the network is illustrated in figure 8.1.
The vectors may be defined as:

\[
W1 = \begin{pmatrix} W_{1,1} & W_{1,2} & \cdots & W_{1,n} \\ W_{2,1} & \cdots & \cdots & \cdots \\ \vdots & \ddots & \ddots & \vdots \\ W_{m,1} & \cdots & \cdots & W_{m,n} \end{pmatrix}, \quad B1 = \begin{pmatrix} B_{1,1} \\ B_{1,2} \\ \vdots \\ B_{1,n} \end{pmatrix} \quad (8.1)
\]

\[
W2 = \begin{pmatrix} W_{2,1} & W_{2,2} & \cdots & W_{2,n} \\ W_{3,1} & \cdots & \cdots & \cdots \\ \vdots & \ddots & \ddots & \vdots \\ W_{m,1} & \cdots & \cdots & W_{m,n} \end{pmatrix}, \quad B2 = \begin{pmatrix} B_{2,1} \\ B_{2,2} \\ \vdots \\ B_{2,n} \end{pmatrix} \quad (8.2)
\]

In this thesis, the input nodes are always the six forces and torques recorded by the force sensor. The output nodes are always the six vector terms describing the position and orientation of the PUMA relative to a neutral (centred) probe position. Section 8.3.2 describes how this data was gathered.

During training, the neural network functions as follows:

1. The input vector \( P \) is applied to the nodes of the input layer of the network.
2. Each input node is connected to each of the hidden layer nodes and weighted by a weighting matrix \( W_1 \). At each node of the hidden layer, the weighted inputs are summed and added to a bias vector \( B_1 \), before being applied to the tan-sigmoid function giving an intermediate output vector \( R \) such that:

\[
R = \text{tanh}[(W_1 \cdot P) + B_1]
\] (8.3)

where \( \text{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \) (8.4)

3. Each output from the hidden layer is connected to each of the output layer nodes where, as before, they are weighted by the matrix \( W_2 \), summed, and added to a bias vector \( B_2 \) before the sum is applied to a linear function giving the output vector \( O \) such that:

\[
O = \text{purelin}[(W_2 \cdot R) + B_2]
\] (8.5)

where \( \text{purelin}(x) = x \) (8.6)

4. The output vector, \( O \), is then compared with its target output vector, \( T \), and an appropriate sum square error function is given by:

\[
E = \sum_{i=1}^{n} (T_i - O_i)^2 (T_i - O_i)
\] (8.7)

5. This error is minimised by adjusting the output layer weights and biases. This error is then linearly backpropagated to the previous layer (the hidden layer) where the weights and biases are adjusted to again minimise the error. This can be continued for as many layers as exist; in our case only one.

6. The process is then repeated with the data re-input at the input layer until either the maximum number of training cycles (epochs) is reached or the sum square error is achieved.

After training, the neural network is used in the feed forward direction only. The input vector \( P \) is applied to the input nodes and an output vector \( O \) is the result. Practical problems arising in the training of artificial neural networks, such as how to choose the number of input nodes, are discussed in section 8.4.
8.3. Experimental Procedure For Data Gathering

8.3.1. Experimental Method

A method was required to gather data with which to train the artificial neural network. It was necessary to measure the forces incurred as the probe moved through a range of translations and rotations relative to the manipulator end flange.

To this end, it was decided to use the PUMA manipulator as the input device for this data gathering. As stated in chapter four, the repeatability of this particular manipulator had been found to be 0.13mm and its accuracy to be ±0.32% [Greig 92]. The training method was to measure motion relative to a neutral start position. This meant errors of the order of 0.13mm as absolute accuracy was not an issue and only the repeatability would bring error to the procedure.

It was considered that to manufacture any device that could approach these levels of accuracy would be a very difficult undertaking. In addition, given the nature of compliant force control where position errors are accommodated by the compliance, a greater accuracy would be unnecessary.

The manipulator control program, mak.moves, its sub-programs and the PC based data logging programs are listed in Appendix K.

8.3.2. Experimental Equipment

The method adopted was to bolt the probe to a rigid workpiece and then drive the manipulator end flange through a range of translations and orientations whilst recording the manipulator position and the corresponding measured forces. The equipment set up is shown in figure 8.2. The enlarged section shows the neutral (starting) position and the tool co-ordinate axes.

A VAL II program was written for the PUMA controller to move the end flange through the desired range. In each position, the current manipulator position was sent via a serial link to the data logging PC. This triggered the PC to write the position to file and to read the current analogue value of force and torque from the force transducer which was also written to file. To avoid any dynamic effects, the manipulator would pause after each motion for 0.5 s to allow it to settle prior to transmitting its location.

The training range of the end flange positions relative to the initial position are shown (in tool co-ordinates) in table 8.2. These values were selected after open loop tracking of surfaces had taken place.
This was done to find typical translation and rotation ranges in which the forces were within the transducer range and had a sufficient signal to noise ratio that they could be adequately filtered. This range was then extended in all directions such that the artificial neural network would always be interpolating its output within the range of true data rather than extrapolating it. Backpropagating neural networks are not reliable when extrapolated [Stonham 94].

It should be noted that, although the motions are planned in tool co-ordinates which rotate about fixed axes, these had to be transformed into PUMA xyzOAT motions to command the manipulator in global co-ordinates. As a result of this, some of the angles in the results are outside the range shown in table 8.2 but are in fact consistent with the global rotations commanded.

<table>
<thead>
<tr>
<th></th>
<th>Translational range (mm)</th>
<th>Rotational range (deg)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>Minimum value</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Maximum value</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Step size</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Number of steps</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Total number of data points = 8100

TABLE 8.2 - TRANSLATIONAL AND ROTATIONAL MOTION RANGE USED FOR DATA GATHERING FOR NEURAL NETWORK
8.4. Training the Artificial Neural Network

8.4.1. Minimising the Training Time for Artificial Neural Networks

The back propagating network uses a gradient descent training method. For a complex non-linear function, the data surface may be of a form with many local minima and maxima. As a result, the training may stop at a local minima far from the global solution. To reduce this possibility, a momentum method of back propagation is used which deliberately overshoots small minima to see if there are further reductions in error to be found 'on the other side'. This is implemented by adding a fraction of the previous weight change to the current weight change arising from the error backpropagation.

This may be imagined by a marble rolling on a plane with valleys and dips. If the marble rolls into a small dip high up, and it is very light, it may stop as shown in figure 8.3a. Figure 8.3b shows how a heavier marble with greater momentum might roll farther out of the dip and discover a further gradient descending beyond it. This is effectively a low pass filter where minor perturbations on the data surface are ignored.

Another method of improving the training speed is by using an adaptive learning rate. The learning rate is the gain by which weights and biases are adjusted after an error is calculated. If the learning rate is too low, training times will be large. If the learning rate is too high, the training may become unstable with the solution jumping back and forth across a minima without settling in it.

The solution is to look at the change in error. If the new error exceeds the old error by more than a certain ratio then the learning rate is reduced. Conversely, if the new error falls short of the old error by more than another ratio, the learning rate is increased. Figure 8.4 illustrates the variation in learning rate during a 1000 epoch training of an artificial neural network.

![FIG. 8.3 - EFFECT OF MOMENTUM ON GRADIENT DESCENT TRAINING](image)
Notice that the error term within the Matlab toolbox is the sum-squared error, not the root mean square (rms) error which is used later in the thesis to examine the results. The error is taken to be the difference between the target output and the neural network output.

Both the momentum and adaptive learning rate techniques are part of the Matlab neural networks toolbox [Matlab NN 93].

8.4.2. Pre Processing of Data

To train an artificial neural network using raw data can result in very poor network performances. One reason for this occurring is differences of range of the data. For example, if one value is ranging from -1 to 1 whilst another is ranging from -10 to 10 (the input translations and rotations) then the network will assign greater significance to the larger magnitude numbers as errors for the smaller magnitude range will appear less.

Another common problem is that of over generalisation. This is the situation when data is presented to the artificial neural network in a very ordered form. For example, the first n data values have x=1, the next n values have x=0 and so forth. As training begins, the data set is applied to the network in the order that it is stored. In this case, this causes the weights to be first minimised to fit the x=1 terms and then minimised to fit
the \( x=0 \) terms. However, the minimisation to fit the \( x=0 \) terms may well contradict the earlier fit resulting in a network biased towards the \( x=0 \) solution. A common and effective solution is to randomise the data order [Venkataraman 93] [Stonham 94].

The data gathered was therefore prepared for training by the following procedures:

1. The data was divided into a 'training' set and a 'verification' set. As the names imply, the 'training' set was used to train the neural networks whilst the 'verification' set was used to verify the trained neural network on data it had not previously seen. The training set contained 5400 vectors of position and force data, the verification set contained 2700 vectors.

2. Both sets were normalised to the range \( \pm 1 \) before training began. The normalising coefficients were saved and were used to process real data prior to applying it to the artificial neural network in real time. This ensured equal weighting would be given to the different input ranges.

3. Any noise on the data set was assumed to be gaussian so that the 'sum of squares' minimisation technique would be valid [Morellan 1989].

4. The data had its order randomised to prevent the artificial neural network from over generalising.

8.4.3. Selection of the Number of Hidden Layer Neurons

The number of layers of hidden neurons and the number of neurons in each layer has a significant effect on the solutions provided by the artificial neural network. The main problems arising are those of underfitting or overfitting the data.

Underfitting occurs if insufficient neurons are used and the problem may not be soluble. It is advised that, in the event of errors remaining high after large training times, too few neurons have been used [Matlab NN 93].

Overfitting may result if too many neurons are used. Every point in the training data set may have a solution, but if new points are interpolated between training points the error may be large. This is illustrated graphically in figure 8.5. Thus, to use too many neurons in the hidden layer will have the penalty of increasing the noise on output data derived from previously unseen input data.
An empirical approach was taken to assess the optimum number of hidden layer neurons. A procedure was then used whereby the back propagation minimisation was run on the training vectors for only 1000 training epochs (an epoch being a single forward and backward pass of the data through the network). This was a relatively rapid procedure (8 hours on a Pentium 95MHz PC for twenty different network sizes) and was used to give an indication of the number of neurons that would be most cost-effective. The underlying assumption was that the better solution would be indicated fairly early on in the training procedure. The resulting rms and maximum errors for both the training and verification data set are shown in figure 8.6.

An anomaly may be seen in that the maximum training error for the translation data flattens out as the number of hidden layer neurons exceeds two. Examination of the original data set indicates that two of the data vectors have corrupted force data and give this unusually high error that distorts the graph. What is particularly significant is that the verification data still performs well. This implies that the artificial neural network was able to train correctly despite two corrupted values. This is to be expected as these comprise only 0.02% of the data set as a whole and so will contribute little to final values of the weights and biases. The rms error from the training data shows a slightly better
performance than from the verification data. This is expected as this was the data that it was designed to fit.

It is apparent that above six hidden layer neurons, the errors reduce considerably. It is therefore supposed that six is the minimum number of hidden layer neurons and this makes sense since it is a six dof system that is being modelled.

Above six hidden layer neurons, there is a significant trend for further reduction in both types of error as the number of hidden layer neurons increase up to about nine neurons. Above that value, the improvements are rather harder to distinguish, and there is little noticeable improvement in the performance.

It was decided to use eleven hidden neurons in the artificial neural network. This put it slightly above the nine value where rapid improvement seemed to slow down. However, since even with twenty neurons the verification data error was still at a similar level, there appeared to be no danger of overfitting the data at this level.

![Graph showing optimal number of hidden layer neurons in the ANN](image)

**FIG. 8.6 - OPTIMAL NUMBER OF HIDDEN LAYER NEURONS IN THE ANN**
8.4.4. Training

The training was carried out in Matlab as a batch process where the entire data set was passed to function \texttt{trainbpx.m} which then applied the backpropagation to minimise the network errors. This function returned the network weights, $W_1$ and $W_2$, and biases, $B_1$ and $B_2$, as well as the number of epochs trained and the errors and learning rates during training.

The initial weights and biases were chosen randomly for the linear output layer using function \texttt{rands.m}. For the tan sigmoid layer, the weights and biases are initialised using function \texttt{nwtan.m}. This applies the Nguyen-Widrow random generator such that their linear response region occurs in the region of the input space where inputs are likely to occur [Nguyen 90].

The chosen network was trained for 4000 epochs. Several training passes were carried out. In all cases, after 3000 epochs there was found to be little variation in the sum square error regardless of the starting values of $W_1$, $W_2$, $B_1$ and $B_2$. In addition, there was little variation in the final solution. This would seem to indicate that the training for this system is fairly robust to variations in starting value.

The Matlab program used to elect initial weights and biases and then train the neural network is shown in Appendix L.

8.5. Verifying the Artificial Neural Network

8.5.1. Comparison Between Training and Verification Data Sets

The means by which the artificial neural network is verified is by comparing the output of the neural network to the original data. This is most useful when it is compared to both the original training data and new data which the artificial neural network has not yet seen. As a graphical representation, figure 8.7 shows the true value compared to the artificial neural network value when the true forces were applied to the artificial neural network for data that it had not previously seen. It is apparent that there is good agreement between the true values in red and the calculated values in blue.

For a more quantitative comparison, table 8.3 shows both the rms error and the maximum error occurring when using the training and verification data sets. The distribution of all the errors for both data sets is shown in figure 8.8 and can be clearly seen to be an approximately normal distribution. Appendix M shows the Matlab program used to calculate the error.
FIG. 8.7 - COMPARISON BETWEEN REAL DISPLACEMENTS AND DISPLACEMENTS CALCULATED BY THE NEURAL NETWORK FOR PREVIOUSLY UNSEEN DATA

In table 8.3, the maximum absolute error is noticeably larger for the y translation than for the other values and is in fact outside the horizontal axis scale of figure 8.8. As was mentioned in section 8.4.3, this results from two corrupt data points within the training data set.

<table>
<thead>
<tr>
<th></th>
<th>Translation error (mm)</th>
<th>rotation error (rad)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>training data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max. absolute error</td>
<td>0.324</td>
<td>0.983</td>
</tr>
<tr>
<td>rms error</td>
<td>0.083</td>
<td>0.051</td>
</tr>
<tr>
<td>verification data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max. absolute error</td>
<td>0.368</td>
<td>0.253</td>
</tr>
<tr>
<td>rms error</td>
<td>0.093</td>
<td>0.052</td>
</tr>
</tbody>
</table>

TABLE 8.3 - ABSOLUTE MAXIMUM ERROR AND RMS ERROR FROM APPLICATION OF THE ARTIFICIAL NEURAL NETWORK TO THE TRAINING AND THE VERIFICATION DATA SET
8.5.2. Interpretation of the Error Distributions

The distribution of error from the artificial neural network is a useful measure of its performance. However, its final use is to give a position of the probe in space to enable monitoring of the surface under inspection. It is therefore necessary to know what type of error may be expected in this final calculation.

To assess this, a short Matlab program was written. The verification force data was applied to the artificial neural network and then scaled to give an appropriate position vector, \( \mathbf{T}_1 \). The true verification position vector was given the title, \( \mathbf{T} \). These two sets of position vectors were then passed through an identical procedure which was a copy of the process carried out by the supervisory computer to ascertain the position of the probe toes and heels in space. This process is explained in detail in chapter nine.

For this comparison, given a fixed manipulator position in space, the probes' toe, left heel and right heel co-ordinates were calculated for each position/force vector in the data set. To compare them, a vector error was calculated between the true position, \( \mathbf{T} \), and the calculated position, \( \mathbf{T}_1 \). The magnitude of this error is illustrated as a histogram in figure 8.9 with the mean and standard distribution of these error magnitudes listed in table 8.4.
As can be seen, these errors are more significant than the neural network output errors because the errors have been magnified by the dimensions of the probe. Unsurprisingly, the mean heel errors are about half of the mean toe errors because the toe is about twice the distance from the centre of force as the heels.

### 8.5.3. Comparison With A Simple Spring Model

The nature of the compliant device meant that, in principal, it was relatively straightforward to model mathematically as a series of linear springs interconnected in space. A model was developed in Matlab that, from a given input translation and orientation, could calculate the resultant forces acting on the force sensor. The neural network model does this in reverse, and so it was necessary to reverse the model such
that it would give a position corresponding to an applied force. The linear spring model and the means by which it was inverted is explained in Appendix N.

Table 8.5 shows the maximum and rms errors of the spring model compared to the neural network model when applied to the verification data set. It can be seen that the neural network model outperforms the linear spring model with the rms error values being between two and five greater for the linear spring model.

Once again, it is useful to see what the effect of this is on the final toe and heel positions calculated. Table 8.6 shows the mean and standard deviation of the magnitude of error experienced at these points. When compared to table 8.4, it can be seen that the errors are again twice as bad for the toe as for the heels. In addition, all the errors are about three to four times what they were for the neural network model. Figure 8.10 shows a histogram of the errors for the spring model so that a direct comparison may be made with figure 8.9 showing the same result for the neural network model.

It is believed that the discrepancy from the linear spring model arises through small variations from the idealised spring stiffnesses and geometry in the real device.

<table>
<thead>
<tr>
<th>Neural Network Data:</th>
<th>Translation error (mm)</th>
<th>Rotation error (rad)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>Max. absolute error</td>
<td>0.368</td>
<td>0.253</td>
</tr>
<tr>
<td>RMS error</td>
<td>0.093</td>
<td>0.052</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Spring Vector Model Data:</th>
<th>Translation error (mm)</th>
<th>Rotation error (rad)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>Max. absolute error</td>
<td>1.324</td>
<td>0.867</td>
</tr>
<tr>
<td>RMS error</td>
<td>0.403</td>
<td>0.258</td>
</tr>
</tbody>
</table>

TABLE 8.5 - RELATIVE ERROR OF THE SPRING AND NEURAL NETWORK MODELS OF THE COMPLIANT DEVICE

<table>
<thead>
<tr>
<th>Toe</th>
<th>Left Heel</th>
<th>Right Heel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean of magnitude of error (mm)</td>
<td>11.180</td>
<td>5.482</td>
</tr>
<tr>
<td>Standard deviation of magnitude of error (mm)</td>
<td>7.677</td>
<td>3.587</td>
</tr>
</tbody>
</table>

TABLE 8.6 - MEAN AND STANDARD DEVIATION OF THE MAGNITUDE OF PROBE TOE AND HEEL POSITION ERRORS
8.6. Conclusions

It is apparent that the backpropagating artificial neural network offers a method of modelling the input/output characteristics of the compliant device. Some of the key ramifications of this are:

1. The neural network is robust, even with some of the training data corrupted.

2. The output error of the neural network is a near normal distribution. The verification data shows rms translation errors of 0.08mm and rms rotation errors of 0.02 rad (1.1°). This results in mean position errors of the toe of 3.4mm and the left heel of 1.7mm. Given the near normal distribution of the errors, it is not unreasonable to assume that, by averaging the toe and heel positions during inspection, the effects of these errors may be further reduced.

3. The solution requires few computer resources. A single pass through the neural network, including scaling and re-scaling terms to the range ±1, requires only 371 floating point operations. By comparison, a single pass through the spring vector model requires 845 floating point operations. Since the spring model must also be inverted, an iterative process requiring from 6 to 30 solving passes, this results in the
vector model requiring from 5070 to 25350 floating point operations. This requires from 13 to 68 times as much processing time.

4. The spring vector model was found to work reasonably well for a very constrained and precisely known geometry of compliant device. However, it was still not as accurate as the neural network model giving a mean left heel position error of 5.5mm compared to 1.7mm for the neural network model. It is thought that if the compliant device geometry grew more complex or used non-linear materials, then the difference in performance would be even greater.

As a result of this, it was concluded that this would be a suitable tool for its intended purpose. The experimental application of the neural network is described in chapter ten.
9.1. Introduction

This chapter describes the task kinematics used for this thesis. These are a description of the task kinematics dealing with Euler angles and homogeneous transformations. This describes the mathematics necessary to be able to define the position and orientation of a point in space and to transform between different co-ordinate systems.

9.2. Task Kinematics

9.2.1. Manipulator Kinematics

Fu defined manipulator kinematics by:

"Robot arm (Manipulator) kinematics deals with the analytical study of the geometry of motion of a robot arm with respect to a fixed reference co-ordinate system as a function of time without regard to the forces/moments that cause the motion" [Fu 87].

With regard to a manipulator comprising a number of rigid links, it is possible to answer the two fundamental questions of kinematics:

1. Given the joint angles and lengths of all the links, what is the position and orientation of the end-effector with respect to a fixed co-ordinate frame? This is known as the forward kinematics problem and always has a unique solution.

2. Given the lengths of all the links and the current position and orientation of the end-effector, what are the current joint angles? This is known as inverse kinematics problem and may have multiple solutions or even be indeterminate.

In general, most robot control tasks are stated in terms of a cartesian reference frame although a polar co-ordinate frame may be used to describe the tool path about a cylindrical workpiece for example. The manipulator control variables are the joint angles, and so the inverse kinematics problem is solved more frequently.
For the purposes of this thesis, it is not necessary to cover all the complexities of manipulator kinematics. The PUMA manipulator and controller used, solves the forward and inverse kinematics for the tasks being carried out. Instead, the problem as it relates specifically to the compliant inspection task is illustrated and the key forward and inverse kinematic tasks for this section are broken down.

9.2.2. Euler Angles

Euler angles enable an orientation to be described within a cartesian co-ordinate system (also called a cartesian frame). When combined with three co-ordinates describing an object’s translation in the cartesian frame, its position and orientation may be precisely determined by a 6x1 vector containing three translations and three rotations.

The three angles describe rotations about particular axes of the co-ordinate system and many different combinations and orders of the axes may be used. Only three angles are required to completely describe any orientation.

FIG. 9.1 - EULER ANGLE ROTATIONS
For example, the PUMA controller uses Orientation, Attitude and Twist (OAT) angles. In a standard right handed co-ordinate system these relate to a rotation of ‘O’ degrees about the z axis. This is followed by a rotation of ‘A’ degrees about the rotated y axis followed by a rotation of ‘T’ degrees about the rotated z axis. This is illustrated in figure 9.1.

This works by starting at the PUMA co-ordinate system XYZ shown by the black axes in figure 9.1.a. A rotation of the co-ordinate frame is made about the Z axis of angle ‘O’. This rotates the frame to a new position X’Y’Z’ shown by the blue axes.

Figure 9.1.b shows the new frame being rotated about the new Y axis Y’ by angle ‘A’. This rotates the frame to a new position X”Y”Z” shown by the red axes.

Finally, in figure 9.1.c, the frame is rotated about the new Z” axis by angle ‘T’ resulting in the final axis system frame X’”Y’”Z’” shown by the green axes.

All three rotations are shown superimposed in figure 9.1.d.

This vector representation may then be translated to a mathematically more useful matrix form comprising a 3x3 rotation matrix.

9.2.3. Rotation Matrices

Each rotation about an axis may be described by a rotation matrix. This is an important feature. Subsequently these rotation matrices may be multiplied together to create a composite rotation matrix which will rotate an object from the original cartesian frame into the orientation described by the three Euler angles. As we shall see, these may be incorporated into a homogeneous transformation matrix which includes terms describing translations within the co-ordinate frame and scaling. The scaling terms are never used in robotics applications but are often used in graphics applications [Fu 87].

The rotation matrix is a 3x3 matrix which operates on a position vector in a three dimensional cartesian frame by mapping its co-ordinates in the rotated frame to the reference co-ordinate frame. The position vector rotates with the axes being rotated. The rotated reference frame is an attached body frame and may be imagined as a co-ordinate frame attached to a rigid body that can rotate freely in space. Its origin always coincides with that of the reference frame.

The development of such rotation matrices is widely described in references and so is not developed here [Fu 87] [Asada 86]. However, the key outcome is to develop the
following rotation matrices for rotations about the $x$, $y$ or $z$ axis of a co-ordinate frame of angles $\alpha$, $\beta$ and $\varphi$ respectively:

\[
\begin{align*}
R_{x,\alpha} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\alpha & -S\alpha \\ 0 & S\alpha & C\alpha \end{bmatrix}, & R_{y,\beta} &= \begin{bmatrix} C\beta & 0 & S\beta \\ 0 & 1 & 0 \\ -S\beta & 0 & C\beta \end{bmatrix}, & R_{z,\varphi} &= \begin{bmatrix} C\varphi & -S\varphi & 0 \\ S\varphi & C\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}.
\end{align*}
\] (9.1)

where $C\alpha = \cos(\alpha)$, $S\alpha = \sin(\alpha)$ and similarly for $\beta$ and $\varphi$.

Each matrix will have the effect of rotating the axes of the attached body frame about the specified axis by the specified angle.

The composite rotation matrix may then be created by multiplying combinations of these single axis rotation matrices together. Matrix multiplication does not commute and the order of multiplication affects the outcome. The following rules indicate the procedure to select the correct order of matrix multiplication [Fu 87]:

1. Initially both co-ordinate frames are coincident and so the rotation matrix is a 3x3 identity matrix $I$.

2. If the rotating co-ordinate frame is rotating about one of the principal axes of the reference (unmoving) frame, then pre-multiply the previous rotation matrix by the appropriate single axis rotation matrix.

3. If the rotating co-ordinate frame is rotating about one of its own principal axes, then post-multiply the previous rotation matrix by the appropriate single axis rotation matrix.

Thus, to achieve the PUMA OAT rotations which rotate about the principal axes of the rotating frame (case 3 above) in the order $z$, $y$, $z$ by angles $\alpha$, $\beta$ and $\varphi$, we do the following:

\[
R_{\text{OAT}} = I.R_{z}.R_{y}.R_{z}.
\] (9.2)

\[
R_{\text{OAT}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\alpha & -S\alpha \\ 0 & S\alpha & C\alpha \end{bmatrix}\begin{bmatrix} C\beta & 0 & S\beta \\ 0 & 1 & 0 \\ -S\beta & 0 & C\beta \end{bmatrix}\begin{bmatrix} C\varphi & -S\varphi & 0 \\ S\varphi & C\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}.
\] (9.3)

\[
R_{\text{OAT}} = \begin{bmatrix} C\alpha.C\beta.C\varphi - S\alpha.S\varphi - C\alpha.C\beta.S\varphi - S\alpha.C\varphi & C\alpha.S\beta \\ S\alpha.C\beta.C\varphi + C\alpha.S\varphi - S\alpha.C\beta.S\varphi - C\alpha.C\varphi & S\alpha.S\beta \\ -S\beta.C\varphi & S\beta.C\varphi & C\beta \end{bmatrix}.
\] (9.4)
This set of rotation matrices is orthogonal and therefore has the useful property that their inverses are their transpose such that:

\[ R^T = R^{-1} \]  \hspace{1cm} (9.5)
\[ R.R^{-1} = R.R^T = I \]  \hspace{1cm} (9.6)

This leads to a rapid solution of the inverse of rotation matrices and, in the next section, transformation matrices.

9.2.4. Homogeneous Transformation Matrices

To enable the inclusion of translation and scaling terms in our rotation matrix, it is expanded to a 4x4 matrix by the inclusion of a 4x1 homogeneous position vector. The first three terms are the position vector, \( p \), whilst the fourth term is a scaling factor which is always 1 in robotics applications. This means that the homogeneous position vector equates to a true physical position. The remaining 1x3 vector space in the 4x4 vector are taken by perspective terms that are useful for computer vision.

Thus the homogeneous transformation matrix appears as \([Fu 87]\):

\[
T = \begin{bmatrix}
    R_{3x3} & P_{3x1} \\
    F_{1x3} & 1
\end{bmatrix} = \begin{bmatrix}
    \text{rotation} & \text{position} \\
    \text{matrix} & \text{vector} \\
    \text{perspective} & \text{scaling} \\
    \text{vector} & \text{value}
\end{bmatrix} \hspace{1cm} (9.7)
\]

Composite homogeneous transformation matrices may be created in exactly the same way as for rotation matrices and following precisely the same rules (section 9.2.3).

Figure 9.2 illustrates the effect of this matrix. In figure 9.2a, it is to translate the zero co-ordinate of the body attached frame to position \( p \) in the reference frame and rotate it according to the 3x3 rotation matrix. By varying the order of the component transformation matrices, the transformation could be equally well made to rotate the frame and then translate to a position, \( p \), in the body attached frame as shown in figure 9.2b.

N.B. using the right hand screw rule, \( \varphi \) is negative in this example.
To find the inverse of a transformation matrix is slightly more complex. The rotation part of the matrix may still be transposed. The position terms must then be calculated using the following formula [Fu 87]:

\[
R = \begin{bmatrix}
  nx & nx & nx & -n^T p \\
nx & nx & nx & -n^T p \\
nx & nx & nx & -n^T p \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
T^{-1} = \begin{bmatrix}
  C\varphi - S\varphi & 0 & 0 & 0 \\
  S\varphi & C\varphi & 0 & 0 \\
  0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

This inverse now represents the position and orientation of the origin of the reference frame with respect to the attached body frame.
9.3. Application of Transformation Matrices in this Thesis

9.3.1. Use of Transformation Matrices to Map the Probe Position

As was discussed in chapter eight, the key task in this work is to be able to find the transformation between the manipulator end flange and the probe contacting the surface. Given the non-linear nature of the compliance (see chapter seven), this is not trivial to calculate.

When this transformation is known, the manipulator may be commanded to move in the probe co-ordinate frame which, when the probe is in full contact with the surface, is aligned with the workpiece co-ordinate frame. It is not necessarily aligned with the tool co-ordinate frame due to distortion of the compliance. This is illustrated in figure 9.3.

The PUMA controller is able to create an offset of the manipulator end flange position and so it is possible to obtain a transform describing the offset flange co-ordinate frame. This means that any commanded motions or rotations will be about this point which is at the centre of the spring array. Using the artificial neural networks, it is possible to generate a transformation describing the position and orientation of the probe relative to the end flange offset. Thus, by multiplying them together, a composite transformation matrix may be obtained that defines the probe co-ordinate system.

The neural network methodology was described in chapter eight. The way in which the result is used is described in the next section.

![Diagram of local coordinate frames with a probe in contact with a surface](image.png)
9.3.2. Extraction of the Data from the Artificial Neural Network

When a force reading is received from the probe, there are several steps required before the required outputs are achieved. The first two steps of the explanation refer to figure 9.4, which shows schematically:

- the reference co-ordinate axes of the manipulator base (b).
- The local (body attached) co-ordinate axes of the manipulator end flange at the start of the training phase (fst) which is a fixed, known position and, the same axes at any time during the training phase (ft).
- The local (body attached) co-ordinate axes of the probe centre (pt) during training which is also a known fixed position.
- The arrows show the transformations or vectors that describe the downstream axes with respect to the upstream axes (where downstream indicates the pointed end of the arrow).

**FIG. 9.4 - TRANSFORMATIONS TO CALCULATE PROBE RELATIVE TO MANIPULATOR**
The steps required to calculate the required result are several. In the following explanation, the term position (applied to the probe, for example) refers to the location of the body attached reference axis frame:

1. Having trained the ANN as described in chapter eight, any force $F$ input to the ANN will output a vector (in the PUMA form using Euler angles) which is equivalent to:
   
   $\text{ANN output} = bV_{ft} - bV_{C_{fs}}$  \hspace{1cm} (9.10)

2. The required first solution is to calculate the transformation of the probe position relative to the manipulator end effector position. This is necessary to allow subsequent mathematical operations to be performed as the vector form may not be multiplied. It can be seen that this may be calculated from:

   $bV_{ft} = bV_{C_{fs}} + \text{ANN output}$ \hspace{1cm} (9.11)

   $bT_{C_{pt}} = bT_{ft} \cdot T_{pt}$ \hspace{1cm} (9.12)

   $bT_{ft} = \text{trans}(bV_{ft}, 'post', 'oat')$ \hspace{1cm} (9.13)

   $fT_{pt} = \text{hom_inv}(bT_{ft}) \cdot bT_{C_{pt}}$ \hspace{1cm} (9.14)

Function ‘trans’ converts a vector to a transformation using ‘post’ multiplication of matrices and the PUMA ‘oat’ Euler angle form. Function ‘hom_inv’ calculates the inverse of a homogeneous transformation matrix.

The remaining steps of the explanation refer to figure 9.5, which shows schematically:

- The reference co-ordinate axes of the manipulator base (b).
- The local (body attached) co-ordinate axes of the manipulator end flange at the start of the operational phase (fs) which is a fixed, known position.
- The local (body attached) co-ordinate axes of the manipulator end flange at its actual position during operation (fa) and at its desired (future) position during operation (fd)
- The local (body attached) co-ordinate axes of the probe at its actual position during operation (pa) and at its desired (future) position during operation (pd).
- The arrows again show the transformations or vectors that describe the downstream axes with respect to the upstream axes.
It is now necessary to calculate a relative motion vector such that the motion command for the manipulator end flange may be given in the probe co-ordinate axes. This enables the motion to be, for example, in the direction of the probe $x$ axis which will not necessarily be aligned with the manipulator end flange $x$ axis due to deformation of the compliance. This final required term is $f_sV_{fd}$. This is achieved by the following steps:

3. At the very start of operation (when the supervisory PC takes command of the PUMA) we record the flange start position as communicated by the PUMA. This is communicated as a transformation and gives us $bTC_{fs}$.

From equation 9.14, we have calculated $aT_{pA}$. This term is renamed $aT_{pa}$. This is equivalent because in both cases it describes the actual probe position relative to the manipulator end flange position.

$$f_sT_{pa} = aT_{pa} \quad (9.15)$$

In addition, during the incremental commanded motions we assume that the compliant device will not deform. This is assumed as the motions are slow and the
commanded motion is updated every 28 ms hence any geometrical change in the compliant device will be very small. Therefore:

\[ \mathbf{b}_T \mathbf{p}_a = \mathbf{f}_d \mathbf{T}_p d \]            \hspace{1cm} (9.16)

4. We know \( \mathbf{b}_T \mathbf{C}_f \). The previously commanded position of the manipulator is \( \mathbf{g}_V \mathbf{f}_a \) and hence we can calculate \( \mathbf{g}_T \mathbf{f}_a \) using the ‘trans’ function. Hence we can solve:

\[ \mathbf{b}_T \mathbf{f}_a = \mathbf{b}_T \mathbf{C}_f \mathbf{f}_a \mathbf{F}_a \mathbf{T}_f a \]            \hspace{1cm} (9.17)

Where \( \mathbf{b}_T \mathbf{f}_a \) is the transformation of the current manipulator flange position relative to the base.

We can also now solve for the actual probe position relative to the base:

\[ \mathbf{b}_T \mathbf{p}_a = \mathbf{b}_T \mathbf{f}_a \mathbf{F}_a \mathbf{T}_p a \]            \hspace{1cm} (9.18)

5. The next step is to define the required probe motion in probe co-ordinates \( \mathbf{p}_a \mathbf{v}_p d \). This is dependant on the current probe task and would be, for example, a \(+x\) motion if the probe were starting to track along a weld. From this we can calculate the transformation \( \mathbf{p}_a \mathbf{T}_p d \) using function ‘trans’ and hence we can solve for the desired final probe position relative to the base:

\[ \mathbf{b}_T \mathbf{p}_d = \mathbf{b}_T \mathbf{p}_a \mathbf{p}_a \mathbf{T}_p d \]            \hspace{1cm} (9.19)

6. From this we can solve for the transformation of the final desired manipulator end flange position relative to the base:

\[ \mathbf{b}_T \mathbf{f}_d = \mathbf{b}_T \mathbf{p}_d \mathbf{h}_o m\_i n v (\mathbf{f}_a \mathbf{T}_p a) \]            \hspace{1cm} (9.20)

Hence we can calculate the final transformation:

\[ \mathbf{g}_T \mathbf{f}_d = \mathbf{h}_o m\_i n v (\mathbf{b}_T \mathbf{C}_f \mathbf{f}_d) \mathbf{b}_T \mathbf{f}_d \]            \hspace{1cm} (9.21)

Hence, by a decomposition we can convert \( \mathbf{g}_T \mathbf{f}_d \) to calculate the required relative motion vector \( \mathbf{g}_V \mathbf{f}_d \). This is the commanded motion to the PUMA to move the flange in probe coordinates.

7. The final required calculation is the co-ordinates of the toe and heels of the probe in order to model the workpiece surface. This is a simpler calculation and simply needs the local co-ordinates of the toe and heels relative to the probe axis frame. The global position of each is then calculated from:
\[ t_{oe}^{xyz} = b_{pa} T_{oe} C_{\text{toe}} \]  \hspace{1cm} (9.22)
\[ \text{leftheel}^{xyz} = b_{pa} T_{oheel} C_{\text{leftheel}} \]  \hspace{1cm} (9.23)
\[ \text{rightheel}^{xyz} = b_{pa} T_{oheel} C_{\text{rightheel}} \]  \hspace{1cm} (9.24)

Where \( C_{\text{toe}} \) is the co-ordinate of the toe relative to the probe axis frame and so forth.

### 9.4. Conclusions

It is apparent that the solution of the required manipulator motion in probe co-ordinates may be calculated by the procedure described in this chapter. The form of the solving process may be summarised by:

1. Record the forces.
2. Using the ANN, convert this to a relative position vector.
3. Use the previously commanded manipulator position vector as the current manipulator position vector.
4. Define the required probe motion as a position vector.
5. Convert all position vectors to their equivalent transformations.
6. Using simple matrix multiplication solve for the desired manipulator motion transformation.
7. Decompose this transformation into a desired motion vector in a form that may be readily understood by the PUMA controller.

These calculations are simple enough to be done within the 21 ms time window (see table 4.4) on a standard 66 MHz 486 PC.
CHAPTER 10
EXPERIMENTAL USE OF AN ARTIFICIAL NEURAL NETWORK MODEL TO ENABLE COMPLIANT FORCE CONTROL

10.1. Introduction

This chapter will describe the practical implementation of the artificial neural networks to control the motion of the probe over a surface. The purpose of the experiment is twofold. Given that compliant surface tracking is taking place using a simple PID controller, it is required to:

1. Use the artificial neural network to calculate the direction in which the manipulator should move in order to track the probe in the surface tangent direction.
2. Use the artificial neural network to calculate the position of the probe toe, left heel and right heel such that the workpiece surface may be mapped in space.

This chapter is laid out in the following steps:

- A description of the experimental apparatus and operation.
- A description of the experimental method.
- Results.
- Conclusions.

10.2. Experiment

10.2.1. Apparatus

The experimental layout is similar to that used in the data gathering phase described in chapter eight. However, in addition to the flat plate, a section of curved plate was used to illustrate the mapping procedure. The equipment is shown in figure 10.1.
10.2.1. Operation of Hardware

The software controlling the experiment is described in detail in chapter four. To restate the key details, the core functionality of the data cycle repeats every 28ms and is as follows:

1. The supervisory PC sends motion commands to the PUMA controller via a serial link operating at 38.4 Kbd (38400 bits per second). Each message contains modifications to the manipulator end flange position relative to its start position. These commands are sent as a six element Euler vector of tool co-ordinate changes consisting of the $x$, $y$ and $z$ translations and rotations. The rotations are about the $x$, $y$ and $z$ axis in turn. Each rotation is about a rotating axis frame (see chapter nine for an explanation).

2. The PUMA controller transforms these into the global frame and calculates appropriate joint angles for the manipulator. The appropriate drive signals are sent to the manipulator servos. The returned joint angles close the loop between the PUMA controller and manipulator and are not passed to the supervisory PC.

3. The force sensor attached to the probe sends current force data to the supervisory PC. This data is then used to calculate the next $xyzOAT$ vector to send to the PUMA controller as explained in chapter four.
10.3. Method

To determine the effectiveness of the artificial neural network, it was decided to perform the following steps:

1. Track over both a curved (convex) and a flat plate.

2. Accurately record the start and end positions of the toe and two heels on each tracked surface.

3. Record the calculated toe and heel positions for subsequent illustration of the outcome.

To fulfil step 1, two cases were recorded on a curved surface and a single case was recorded on a flat surface.

To fulfil step 3, a simple change was made to the C++ program such that the data was written to file.

To fulfil step 2, the PUMA manipulator was used as the measuring device. The means of doing this was to program the PUMA controller such that the end flange centre of motion was coincident with one of the vertices of the compliant device attachment plates as shown in figure 10.2. The command ‘where’ typed at the PUMA monitor returns the current Euler vector of the tool co-ordinate centre point. With the centre transformed as shown in figure 10.2, the command ‘where’ now returns the current Euler vector of the corner with the attached co-ordinate frame.

**FIG. 10.2 - TRANSFORMED TOOL CO-ORDINATES FOR MEASURING WORKPIECE POSITION**
Thus, by moving the corner to physically touch a point on the surface under inspection, the ‘where’ command will return the Euler vector of that point. In fact, the orientation part of the Euler vector will be meaningless in describing the point and so only the first three terms are needed to give the co-ordinate.

During a tracking experiment, the C++ code was adapted such that the manipulator paused at the start and end of a surface track. This allowed the surface to be marked at the exact position of the toe and heels at both the start and end of the track. After the experiment, these points were measured by the method described in step 2 above.

This was thought to be an appropriate and sufficiently accurate measuring device for this reason:

The positions given for the toe and heels in space by the artificial neural network are a product of the neural network output and the position of the manipulator in space as calculated by the PUMA controller. There is no absolute external measure of position. This means that the accuracy of the manipulator is not going to cause much of a problem. The surface measurement takes place in the same volume of space as the tracking experiment and any accuracy errors are likely to be the same for both as the joints will all be in a similar position. These errors will thus cancel out when comparing the results. The repeatability errors will still have an effect but with these errors being only 0.13 mm, as we saw in chapter four, this is an order of magnitude less than the likely heel errors found in chapter eight. As a result, it is unlikely to prejudice the experimental results.

10.4. Results

10.4.1. Recorded Toe and Heel Co-ordinates

Table 10.1 shows the recorded toe and heel co-ordinates for the three tests. These values are superimposed on some of the graphs in section 10.3.2 showing the probe path as calculated by the artificial neural networks. The estimated accuracy of each of these measurements is ±1mm.

Table 10.2 shows the true scalar lengths between the heels and the toe and compares them to the lengths derived from the measurements detailed in table 10.1. The errors are highlighted in blue and show values ranging from 0.02mm to 2.08mm with an average error of 0.74mm. This would indicate that the maximum measurement error is of the order of 1mm, as the worst case error will arise when two measurements are each wrong.
by 1mm in opposite directions, thus aggravating the total error. This supports the idea of a measurement accuracy of ±1mm.

<table>
<thead>
<tr>
<th>All units are mm</th>
<th>Start of tracking</th>
<th>End of tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>toe</td>
<td>left heel</td>
</tr>
<tr>
<td>test 1 flat x</td>
<td>-48</td>
<td>-76</td>
</tr>
<tr>
<td>y</td>
<td>652</td>
<td>574</td>
</tr>
<tr>
<td>z</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>test 2 curve x</td>
<td>-220</td>
<td>-248</td>
</tr>
<tr>
<td>y</td>
<td>789</td>
<td>720</td>
</tr>
<tr>
<td>z</td>
<td>123</td>
<td>88</td>
</tr>
<tr>
<td>test 3 curve x</td>
<td>-220</td>
<td>-248</td>
</tr>
<tr>
<td>y</td>
<td>789</td>
<td>720</td>
</tr>
<tr>
<td>z</td>
<td>123</td>
<td>88</td>
</tr>
</tbody>
</table>

**TABLE 10.1 - TOE AND HEEL CO-ORDINATES AT THE START AND END OF TRACKING**

<table>
<thead>
<tr>
<th>Measurement</th>
<th>toe to left heel (mm)</th>
<th>toe to right heel (mm)</th>
<th>left heel to right heel (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true distances</td>
<td>(1) 81.82</td>
<td>81.82</td>
<td>55.00</td>
</tr>
<tr>
<td>start of test 1</td>
<td>(2) 82.88</td>
<td>81.02</td>
<td>56.04</td>
</tr>
<tr>
<td>error (</td>
<td></td>
<td>1)-(2)</td>
<td></td>
</tr>
<tr>
<td>end of test 1</td>
<td>(4) 81.94</td>
<td>81.96</td>
<td>56.01</td>
</tr>
<tr>
<td>error (</td>
<td></td>
<td>1)-(4)</td>
<td></td>
</tr>
<tr>
<td>start of test 2</td>
<td>(6) 82.28</td>
<td>81.44</td>
<td>56.01</td>
</tr>
<tr>
<td>error (</td>
<td></td>
<td>1)-(6)</td>
<td></td>
</tr>
<tr>
<td>end of test 2</td>
<td>(8) 81.05</td>
<td>79.74</td>
<td>55.02</td>
</tr>
<tr>
<td>error (</td>
<td></td>
<td>1)-(8)</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 10.2 - TRUE AND MEASURED LENGTHS BETWEEN HEELS AND TOE**

### 10.4.2. Surfaces Mapped by the Artificial Neural Network

The graphs in this section show the toe and heel paths of the probe as calculated by the artificial neural network. In all cases, the toe path is shown by a green line, the left heel by a red line and the right heel by a blue line. In the case of the YZ plots, the true workpiece surface is drawn in black. For test 1 it is a line representing the edge of a flat plane with its surface aligned with the XY plane. For tests 2 and 3, the surface is drawn as an arc of a circle representing the side view of a cylinder surface. The cylinder was aligned with its axis along the global X axis.

The paths for test 2 are shown as three two dimensional views in figure 10.3. The loci all make sense but only the top right hand image is useful in visualising the error of
the system. This YZ view is thus shown for all three tests in figure 10.4 and is shown enlarged for test 2 in figure 10.5.

FIG. 10.3 - 2-D PROJECTIONS OF PROBE LOCI IN GLOBAL CO-ORDINATE FRAME

FIG. 10.4 - PROBE LOCI IN THE YZ PLANE FOR THREE WORKPIECE PASSES
Figures 10.3 to 10.5 give a flavour of the type of error occurring in all the test cases. To look more precisely at the error, the moment at which stable contact has been achieved and just before tracking begins is examined in more detail. This is the moment at which the heels and toe are in the positions marked by the blue circles in figure 10.5. At this moment, the probe should be in the start position measured by the manipulator and so a direct comparison between real and calculated probe position may be made. The overall error (magnitude of error) of the toe and heel positions as well as the components of error in the $x$, $y$ and $z$ directions are given in table 10.3.

<table>
<thead>
<tr>
<th>test</th>
<th>toe (mm)</th>
<th>left heel (mm)</th>
<th>right heel (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>1</td>
<td>0.54</td>
<td>2.68</td>
<td>9.58</td>
</tr>
<tr>
<td>2</td>
<td>1.03</td>
<td>-0.85</td>
<td>12.07</td>
</tr>
<tr>
<td>3</td>
<td>0.66</td>
<td>-1.26</td>
<td>13.26</td>
</tr>
</tbody>
</table>

TABLE 10.3 - COMPONENT AND COMBINED ERROR OF CALCULATED PROBE POSITION
10.5. Conclusions

10.5.1. Sources of Error

It can be seen from table 10.3 that the combined magnitudes of error are large. These are far beyond the range of errors expected from either the neural network (see chapter eight) or from errors of measurement of the surface position in space using the manipulator (see this chapter section 10.3).

By examining the components of position error, it is apparent that the error in all cases arises mostly from the \( z \) direction with a fair contribution from the \( y \) direction and a minimal contribution from the \( x \) direction. In fact, the \( x \) errors lie within the error bounds of measurement and neural network error. The \( y \) errors lie at the edge of this error margin and exceed it for the left heel. The \( z \) errors exceed the error margins for all cases.

By examining the form of the three workpiece surface models in figure 10.4., it is apparent that all the toe and heel paths parallel the workpiece surface giving an extremely stable result. However, it is as though the probe experienced larger surface normal forces than actually existed and so was tilted further back about its \( x \) axis. This is illustrated in figure 10.6 by superimposing the true and the modelled probe position onto the modelled surfaces from figure 10.5. In other words, the forces recorded lead the neural network to believe that the probe is under a greater deformation than it is, and that this deformation is taking place in the \( yz \) plane. This is in keeping with the distribution of the components of error.

The combination of the distribution of the components of error and the stability of the errors point to a calibration error. The error will have arisen because of the different way in which the force sensor is zeroed in the data gathering phase and in the surface tracking phase as explained in the next section.

![Illustration of true and modelled position of tracking probe](image)
10.5.2. Error Due to Calibration

Prior to data gathering, the probe start position is set to zero by eye. The probe is set on a worktable and the manipulator adjusted until the compliant device is in a neutral geometrical position. This is done by taking a sight across the top edges of the three connection plates as is illustrated in figure 10.7. This enables alignment along the z axis and about the x axis. The other alignments take place by finding their natural equilibrium of the probe on the flat plate. The probe is then bolted to the work table and the force sensor is zeroed.

A different method is used to zero the force sensor prior to surface tracking. First, the initial droop angle of the probe relative to horizontal is measured. As the probe’s mass and centre of mass are known, the resulting true forces acting on the force sensor may be calculated. Thus, a bias may be calculated that will offset the initial force reading to the true value. Thereafter, the bias may be subtracted from the force reading to give the true force.

A difference between these two methods would result in an error in the toe and heel position calculations. The second method will give a true zero, but a flaw in the first method arises because, although the spring plates may be aligned to be geometrically at zero, they may not be delivering zero force in reality. This is due to small imperfections in the geometry and variations in the pre-tension on the springs. This means that the zero position (force sensor biased to zero) may in reality have net forces and torques acting on the force sensor.

FIG. 10.7 - ALIGNMENT OF THE COMPLIANT DEVICE TO FIND THE ZERO FORCE POINT
If these net forces exist they would comprise force in the \( z \) direction and torque about the \( x \) and \( y \) axes. These are the force constrained degrees of freedom. The other degrees of freedom are unconstrained and so will have automatically found their zero force equilibrium point before the probe is bolted to the work table. Of the constrained degrees of freedom, the \( y \) rotation is least likely to happen. This is because, even with the probe drooping beneath its own mass, there would be symmetry about the \( y \) axis and so it is relatively simple to adjust the springs until it is centred. The \( z \) translation and \( x \) rotation, however, cannot be fully zeroed by eye as they are always affected by the probe mass.

It should be noted therefore, that the un-calibrated degrees of freedom for this system are the \( z \) displacement and the \( x \) rotation. Errors in these degrees of freedom would produce precisely the type of steady state error seen in the results.

The \( x \) translation is unconstrained in the training phase and will therefore be truly zeroed prior to the probe being bolted to the work table. This would indicate that any error due to poor calibration would not affect the \( x \) axis position which is fully in keeping with the results of table 10.3 which then gives errors varying from 0.06mm to 1.07mm. This would be a very fine result for a surface mapping system such as this.

10.5.3. Summary

This chapter has related the final experiments that took place to validate the artificial neural network as a suitable tool for modelling the compliant device for real time surface tracking.

1. It has shown a consistent output from the neural network but with an error larger than expected. The error has been consistent and at no time has a large outlier appeared giving radically incorrect results.

2. It has demonstrated that a calibration problem is what has left the errors so large. Future data gathering for the training of the artificial neural network must therefore begin with the same biasing and zeroing procedure that the tracking software used to eliminate such errors. It is apparent that the errors in the \( x \) direction (that is unaffected by the calibration error) are small.

3. It has demonstrated maximum errors of only 1.07mm for a properly calibrated dof (in this case the \( x \)-axis) thus indicating a precise system.
The implications of these results are that:

1. An artificial neural network will give reasonable and stable output for this problem.

2. A surface may be mapped with a high precision using this technique.

3. The mapping may be done in real time as the artificial neural network allows the force/position relationship of the compliant device to be solved rapidly (in less than 28 ms).

4. As the errors are of the order of ±1 mm in the worst case once calibration is carried out, the axis of the cylinder may be determined to a high accuracy fairly rapidly. After 20 mm of travel, an error of 1 mm will result in an axis misalignment of only 2.8°. A mean error of considerably less than 1 mm will result in similarly reduced misalignments.

So, we can see that a useful tool to assist compliant force control of sub-sea inspection has been developed.
11.1. Introduction

This thesis has presented a range of work aimed at implementing compliant force control for automated sub-sea inspection. This chapter will examine the results achieved and assess the extent to which this goal has been reached.

11.2. Thesis Summary

Chapter one introduced the need for sub-sea intervention to maintain existing and future offshore jackets. The many hazards faced by divers and the substantial costs of such operations were outlined along with the potential advantages of unmanned missions in terms of cost, safety, speed and accuracy.

In chapter two, a background to robotics was detailed, covering key force control methodologies and introducing the force control scheme used in the experimental work.

Chapter three gave a more detailed description of the problem under investigation. A physical description of the components of the sub-sea intervention system was given. In addition, some of the task constraints were outlined and the requirement for compliant force control was explained.

Chapter four described the geometry, accuracy and limits of the PUMA 560 manipulator. In addition, it detailed the C++ software developed to control the manipulator from a supervisory PC.

Chapter five described experiments carried out to measure the magnitude of motion experienced by an ROV docked onto a sub-sea node by three vacuum feet.

Chapter six described the testing of a stereo vision system designed to compensate for the problem of an ROV moving relative to the structure it is inspecting.

Chapter seven examined the mechanical design of the compliant device based on the physical constraints of the inspection task. Once the initial design had been chosen,
mathematical modelling was used to optimise the arrangement of springs to give a useful performance when in its designed operating position. This result was a useful laboratory compliant device that was used throughout the investigation carried out in this thesis.

Chapter eight examined the use of a backpropagating artificial neural network to model the input/output characteristics of the compliant device. This gave some background on artificial neural networks, applied the data to a network and examined the accuracy of the output. In addition, the artificial neural network performance was compared to that of a more conventional stiffness model.

Chapter nine detailed the mathematics of the transformation matrices used to describe the position of the probe in various co-ordinate frames. It also explained how this could be used to transform the output of the artificial neural network into a form that the manipulator controller could use.

Chapter ten detailed the final experiments to validate the artificial neural network as a suitable tool for modelling the compliant device for real time surface tracking.

11.3. The PUMA 560 and Associated Control System

It was seen that the PUMA 560 manipulator is a versatile tool enabling many motion controlled tasks to be carried out to a high accuracy. The measured accuracy of this manipulator was ±0.32% and the repeatability was 0.13mm [Greig 96].

The PUMA 560 does have some limitations of which the most serious is the low clock rate at which it will receive commands from an external computer; only once every 28ms. However, for compliant force control with low speed tracking, it is apparent that this does not prevent effective control. Another drawback is the relatively slow serial link with which it talks to the supervisory PC thus reducing the available processing time per clock cycle for the PC from 28ms to 21ms - a decrease of 25%. The final drawback is that the force sensor is non-colocated with the manipulator and so is able to vibrate out of phase with the manipulator and so create a control problem at higher tracking speeds. Experimentally, it was found that the simple controller used was unable to maintain stable control at tracking speeds of over 25 mm/s and would become oscillatory above 15mm/s.

To apply force control to the PUMA and compliant device, the control system outlined in figure 11.1 was used. The inner PUMA controller uses a PID control methodology with fixed gains that is implemented at a hardware and software level.
The outer force control loop is controlled by a PC. This implements a PID force control algorithm in which contact forces are measured and new manipulator positions commanded. An ANN is used to derive the required motions from the measured forces.

### 11.4. Applicability of Existing Sub-Sea Intervention Systems

It was seen that the ARM system comprises an advanced six dof hydraulic manipulator and an ROV deployment system that enables the ROV to 'dock' onto the oil-rig structure. This is a high quality system proven in trials. The estimated manipulator accuracy of 5-10mm absolute and 1-3mm between points is of an order that may be accommodated by a compliant inspection device. The compliant device designed in this thesis has a range of travel designed for a higher accuracy manipulator. However, this could be scaled up for the real application.

The identified real application is that of running a probe along a weld toe at the junction of two cylinders when the angle between the cylinders is 30° (in the worst case). Compliant force control was chosen as an appropriate solution to the problem because inspection speed is not critical, model errors are expected and it is known that the docked ROV is unlikely to be a rigid platform.
11.5. ROV Motions and Disturbances

It was seen that under real loads, with the ROV docked onto the wall of a water tank, the displacements of the geometric centre of the ROV were of the order of:

1. Dynamic loads. For forces of the order of 1300N, position excursions varied from 8mm to 15mm. It was established that likely dynamic forces would arise from manipulator contact with the structure under inspection or from wave forces near the surface [Broome 96]. These would be unlikely to exceed this load.

2. Static loads. Approximately 5mm excursions resulting from forces of the order of 170N. It is expected that steady state loads induced by water currents might be of the order of 400N to 1000N in a 0.375 ms\(^{-1}\) current [Broome 96]. Thus greater excursions would be likely but, as they are steady state, would be unlikely to cause larger control problems.

It was thought that a more detailed analysis, and perhaps some other measurements would result in a better model to use in the simulation. For example, an examination of the stiffness of the cable used to apply impulse loads to the ROV would enable a better understanding of the load profile.

It was also thought that to use two or less attachment legs would result in unreasonably large motions of the ROV. Three docking legs are therefore essential for automating tasks such as manipulator based inspection or grind repair.

11.6. Alternative Means of Minimising Manipulator Disturbances

A stereo vision motion compensation system was investigated. The purpose of this was to compensate for motion of the ROV relative to the oil-rig. It was found that:

1. The vision system worked and enabled tip errors to be reduced to 6-7mm when the relative motion was of the order of 50mm. The vision system resolution was of the order of 1-2mm in x and y and 3-4mm in z.

2. The scale of error was directly related to the slow processing time of the vision system and the communications between the various processing modules. It should be noted that the vision system was built from an existing 3D underwater viewing system and was never intended to operate at high speeds.
These tests showed that a stereo vision based system has the potential to significantly reduce motion-induced manipulator-tip errors, and therefore form the basis of a practical motion compensation system. It also showed that improvements would be required to the vision system resolution and update rate to be able to track more distant targets at an acceptable rate.

It is noted that a real system is unlikely to improve on errors of several mm due to resolution limitations and so a compliant force solution is still required.

11.7. Design of a Novel Compliant Device

To deal with the geometric constraints of a real inspection scenario, a new compliant device was designed. The design process led to an arrangement that would allow a probe to reach into a tight angle. To enable real time mapping of the surface, it was decided that the force transducer would be placed downstream of the compliance and thus be dynamically non-co-located with the actuator. The choice of the position of the contact feet relative to the force sensor enabled torque measurements to directly reflect the probe orientation. The resultant compliant device was able to track at a low speed that was dependant on the physical properties of the compliant device and the bandwidth limitations of the controlling force loop.

A steel spring and aluminium body prototype compliant device was built and successfully tested. Prototype polymer compliant devices were also built although not, as yet, tested. It is speculated that the inherent hysteresis in polymer products will lead to a reduction in accuracy of position prediction using artificial neural networks.

The mathematical modelling of the compliant device to choose the arrangement of springs allowed optimal designs to be generated. It also highlighted how important it was to consider the model within its normal operating range.

The finished transverse spring compliant device fulfilled the design requirements. It has six dof, possesses the desired directional stiffness and enables the manipulator arm to be attached behind the probe instead of above it hence allowing access to tighter welds.

11.8. Artificial Neural Network Modelling of the Compliant Device

The backpropagating artificial neural network offered an effective, low computational overhead means of modelling the input/output characteristics of the compliant device. This was a novel application of artificial neural networks.
It was found that the neural network is robust, even with some of the training data corrupted. In addition, the output error of the neural network was a near normal distribution. The verification data showed rms translation errors of 0.08 mm and rms rotation errors of 0.02 rad (1.1°). This resulted in mean position errors of the toe of 3.4 mm and the left heel of 1.7 mm. Given the near normal distribution of the errors, it is not unreasonable to assume that, by averaging the toe and heel positions during inspection, the effects of these errors may be further reduced to produce accurate surface mapping.

As mentioned, the solution requires few computer resources. A single pass through the neural network, including scaling and re-scaling terms to the range ±1, requires only 371 floating point operations. By comparison, a single pass through the spring vector model requires 845 floating point operations. Since the spring model must also be inverted, an iterative process requiring from 6 to 30 solving passes, this results in the vector model requiring from 5070 to 25350 floating point operations. This requires from 13 to 68 times as much processing time. Since the application had only a small processing window due to the serial communication delays between the PC and the PUMA, this was an extremely important factor.

The alternative spring vector model that was considered was found to work reasonably well for the precisely known geometry of the compliant device. However, it was still not as accurate as the neural network model giving a mean left heel position error of 5.5 mm compared to 1.7 mm for the neural network model. It is thought that if the compliant device geometry grew more complex or used non-linear materials, then the difference in performance would be even greater.

It is concluded that the ANN is a suitable tool for its intended purpose. It provides a fast and accurate method of establishing the probe position from the measured forces. It is apparent that the solution of the required manipulator motion in probe co-ordinates may then be calculated from the probe position by a relatively simple set of matrix operations.

11.9. Experimental Conclusions

It was demonstrated that the whole system was able to give:
1. Real time manipulator motion commands in the probe axis co-ordinate system. As the probe was in full contact with a curved surface, this gave motion commands along the tangent to the curved surface.

2. Real time probe toe and heel co-ordinates in the global co-ordinate system. This enabled the surface being tracked to be mapped.

3. Stable output from the artificial neural network giving consistent direction and position data.

A problem arose with the surface mapping data that meant the toe co-ordinates were parallel to the surface but offset in the global y and z directions. This is consistent with a calibration error resulting from the different means by which the force sensor was zeroed prior to the training data collection and prior to a surface tracking task. After the calibration error was taken into account (by looking at only the x dof that was unaffected by this error), it was seen that the errors vary from 0.06mm to 1.07mm. This is of greater accuracy than the manipulator and so more than adequate for its task.

As was stated in chapter ten, the implications of these results are that:

1. An artificial neural network will give reasonable and stable output for this problem.
2. A surface may be mapped with a high precision using this technique.
3. The mapping may be done in real time as the artificial neural network allows the force/position relationship of the compliant device to be solved rapidly (in less than 21 ms).
4. As the errors are small once calibration is carried out, the axis of the cylinder may be determined to a high accuracy fairly rapidly. i.e. after 20 mm of travel, a mean error of 1 mm will result in an axis misalignment of less than 2.8°.

As regards the fulfillment of the key tasks for success, outlined in chapter 1, it is apparent that the ROV motion has been measured successfully and an effective compliant device design has been made.

It was also required that certain quantitative results be seen to be achievable:

1. A desired accuracy for local modelling of the probe relative to the surface was to have the commanded direction to lie within 2° of the true surface tangent. The
compliant device is under force control and so, for this to be achieved, the global accuracy is immaterial. From chapter 8, we have seen mean toe errors of 3.4 mm and mean heel errors of 1.7 mm. As the toe and heels are rigidly connected and tend to rotate about a common centre, it is apparent that the errors are related and if one is positive, the other will be positive as illustrated in figure 11.2. Thus the mean worst case difference in errors will be 1.7 mm which, with a length of probe of 75 mm, will give rise to an angular error of 1.3°. This is within the desired accuracy range.

![FIG. 11.2 - COUPLED NATURE OF TOE AND HEEL ERRORS](image)

An error of 1.3° results in a step force disturbance of only 0.15 N when tracking at 10 mm/s. This is because each 28 ms clock cycle, a motion of 0.28 mm is commanded. With an error of 1.3°, this gives a lateral motion error of 0.006 mm. This is less than the minimum commanded motion of the PUMA and so, instead, we take the minimum motion of 0.031 mm to be the lateral motion error. Since the actual stiffness in the workpiece direction is 4.93 N/mm, this results in a mean force disturbance each clock cycle of only 0.15 N.

As the probe is under force control, errors in surface position will not affect the tracking as it will follow what is actually there as oppose to what was thought to be there.

A desired accuracy of global modelling was to be able to map the crack on the weld to within 10 mm of its real position. The cumulative errors arising from the different areas of inaccuracy may be given by table 11.1. This indicates that the global mapping of the the crack is not achievable to the desired standard of ±10 mm and in fact would be from ±12.5 to ±18 mm.
The local mapping is thus within the target accuracy. The global mapping, although outside the stated target, is certainly usable now and could be brought within the target by improvements to a stereo vision system in the case when dynamic motion of the ROV is taking place, and by proper calibration of the sub-sea manipulator to improve the absolute accuracy in both cases.

### 11.10. Summary

Work investigating the motion of an ROV attached to a rigid sub-sea structure has been conducted allowing an insight into the type of motions that a manipulator control system might have to cope with. This information has been used to develop a useful tool to assist compliant force control of sub-sea inspection. This tool is a novel compliant device, of simple design, using steel springs, that was built to achieve required relative stiffness levels in orthogonal directions. By using a novel application of an artificial neural network, the force/position relationship for the compliant device was modelled with sufficient accuracy that the toe and heel positions could be mapped with mean position errors for the toe of 3.4mm and for the left heel of 1.7mm.

### 11.11. Future Development

The work to date has developed a laboratory proving ground for the system. There are a number of possible future developments that would continue or augment this work.

#### 11.11.1. Generating Stiffness Data from the Artificial Neural Network

A possible future development of this neural network compliant model is to use it to supply stiffness information about the compliant device. The existing ANN is being
successfully used in a static model of the probe contacting the environment. It is
foreseen that the ANN may be used to supply dynamic information by considering that at
any time, $t$, we have:

\[
\begin{align*}
\mathbf{F} & \quad \text{- measured force} \\
\mathbf{P} & \quad \text{- ANN output} \\
\Delta \mathbf{F} & \quad \text{- contrived small force increment} \\
\Delta \mathbf{P} & \quad \text{- contrived small ANN output increment}
\end{align*}
\]

We can therefore input a contrived force vector to the ANN and obtain a slightly
changed output:

\[
(\mathbf{P} + \Delta \mathbf{P}) = ANN(\mathbf{F} + \Delta \mathbf{F})
\]

Thus we can evaluate the local stiffness vector, in workpiece co-ordinates, as:

\[
\mathbf{K} = \frac{\Delta \mathbf{F}}{\Delta \mathbf{P}}
\]  

If the local stiffness may be simply predicted in any dof of the probe axis system, this
may prove a useful tool in modelling dynamic effects of the probes performance under a
stiffness control scheme.

11.11.2. Calibration Procedures

An additional calibration procedure may be needed in operation. This might be to move
the probe to a known surface prior to the inspection of an unknown surface (ie a known
point on the ROV structure). At rest on the surface, the force bias could be adjusted to
ensure that the correct heel and toe co-ordinates are calculated. This bias may then be
used for the forthcoming inspection.

This method has the added advantage that it would eliminate the effect of any slow
drift of the force sensor as it would be re-calibrated before such a slow affect could
influence results appreciably.

It is envisaged that any compliant device in actual use would need to be re-calibrated
in terms of gathering data and retraining its ANN at regular intervals. If it was not re-
trained, there would still be a requirement to test the compliant device for accuracy. This
need would arise because:

1. The compliant device material properties will change with time due to work
   hardening of the materials.
2. A hard impact might permanently deform part of the compliant device thus changing its properties.

3. Without such re-training, the quality of data gathered would be suspect. Such tests could not be carried out by the hydraulic manipulator as it is not sufficiently accurate. The compliant device would need to return to a base with a device of an accuracy equivalent to the PUMA 560 for testing and/or re-training.

It would also be necessary to add gravity compensation to enable surfaces to be tracked in any orientation.

11.11.3. Development of a Compliant Device for Sub-Sea Application

Preliminary work was carried out in developing a polymer compliant device that would be more resistant to the corrosive effects of sea-water than the steel and aluminium probe. This could also be achieved by coating the springs in a polymer. This area would require the following work:

1. Design and build of a suitable polymer compliant device.

2. Test to see the accuracy of model that may be built using an ANN. This may have problems due to hysteresis.

3. Design and build a coated spring compliant device.

4. Test the accuracy of the ANN model with the coated spring compliant device.

11.11.4. Development of a Force Sensor for Sub-Sea Application

The force sensor is currently unsuitable for sub-sea work. A project is required to develop a small force sensor that could function in the sub-sea environment. This may require encapsulation of the sensor in a polymer or similar coating.
APPENDIX A

TECHNICAL SPECIFICATION FOR THE SLINGSBY ENGINEERING MRV

Technical specification for the Slingsby Engineering MRV (multi-role vehicle):

Core Vehicle:

<table>
<thead>
<tr>
<th>Vehicle configuration</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length (mm)</td>
<td>1920</td>
<td>2300</td>
<td>2700</td>
</tr>
<tr>
<td>Width (mm)</td>
<td>1500</td>
<td>1500</td>
<td>1500</td>
</tr>
<tr>
<td>Height (mm)</td>
<td>1560</td>
<td>1560</td>
<td>1560</td>
</tr>
<tr>
<td>Depth rating (m)</td>
<td>600</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Weight in Air (^{\dagger}) (kg)</td>
<td>1485</td>
<td>1860</td>
<td>2100</td>
</tr>
<tr>
<td>Payload Capacity (^{\dagger\dagger}) (kg)</td>
<td>260</td>
<td>220</td>
<td>350</td>
</tr>
<tr>
<td>Power pack (shaft kW/HP)</td>
<td>32/43</td>
<td>50/67</td>
<td>75/100</td>
</tr>
<tr>
<td>Thruster diameter (mm) - 6 off</td>
<td>300</td>
<td>380</td>
<td>380</td>
</tr>
<tr>
<td>Bollard Pull (^{\dagger\dagger}) - Intervention mode:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FWD/AFT (kgf)</td>
<td>300</td>
<td>550</td>
<td>630</td>
</tr>
<tr>
<td>Lateral (kgf)</td>
<td>300</td>
<td>550</td>
<td>630</td>
</tr>
<tr>
<td>Vertical (up) (kgf)</td>
<td>300</td>
<td>520</td>
<td>520</td>
</tr>
<tr>
<td>Bollard Pull (^{\dagger\dagger\dagger}) - Survey mode:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FWD (kgf)</td>
<td>350</td>
<td>640</td>
<td>720</td>
</tr>
<tr>
<td>Lateral (kgf)</td>
<td>270</td>
<td>500</td>
<td>570</td>
</tr>
<tr>
<td>Lift capacity (kg)</td>
<td>7000</td>
<td>7000</td>
<td>7000</td>
</tr>
</tbody>
</table>

\(^{\dagger}\) - excludes payload

\(^{\dagger\dagger}\) - based on average payload specific gravity of 6.0

\(^{\dagger\dagger\dagger}\) - based on fitted power pack only

Standard fit:

- Gyro/fluxgate compass
- Pitch/roll sensor
- Digiquartz depth gauge
- Auto functions (heading/depth/altitude/pitch/roll/trim)
- Temperature measurement interface
- General function valve pack (II function)

Interfaces for:

- Manipulators (left/right)
- Sonar
- Light control
- General function valve pack (2\(^{nd}\))
- Camera control (5 cameras includes still camera trigger)

Telemetry:

- Transmission via optical fibres
- Real time diagnostics

Power distribution:

- Individual vehicle system isolator
- Spare power available for customer

181
Technical specification for the Slingsby Engineering A206 manipulator:

<table>
<thead>
<tr>
<th>Weight (in air)</th>
<th>155 Kg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight (in water)</td>
<td>103 Kg</td>
</tr>
<tr>
<td>Reach (maximum)</td>
<td>2500 mm</td>
</tr>
<tr>
<td>Reach (minimum)</td>
<td>250 mm</td>
</tr>
<tr>
<td>Lift capacity at full reach (140 bar - arm vertical)</td>
<td>20 Kg</td>
</tr>
<tr>
<td>Lift capacity at full reach (140 bar - arm horizontal)</td>
<td>15 Kg</td>
</tr>
<tr>
<td>Hydraulic maximum pressure</td>
<td>210 bar</td>
</tr>
<tr>
<td>Hydraulic flowrate</td>
<td>15 lpm</td>
</tr>
<tr>
<td>Compensation system working pressure</td>
<td>0.14 to 0.35 bar</td>
</tr>
<tr>
<td>Compensation system capacity</td>
<td>1.8 litres</td>
</tr>
</tbody>
</table>

**TABLE B.1 - SLINGSBY A206 MANIPULATOR GENERAL SPECIFICATIONS**

<table>
<thead>
<tr>
<th>movement (degrees)</th>
<th>max torque (Nm)</th>
<th>limb length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>shoulder yaw</td>
<td>±135°</td>
<td>1500</td>
</tr>
<tr>
<td>shoulder pitch</td>
<td>+180° -90°</td>
<td>1500</td>
</tr>
<tr>
<td>elbow pitch</td>
<td>+90° -180°</td>
<td>886</td>
</tr>
<tr>
<td>wrist pitch</td>
<td>+90° -150°</td>
<td>409</td>
</tr>
<tr>
<td>wrist yaw</td>
<td>±90°</td>
<td>409</td>
</tr>
<tr>
<td>wrist rotation</td>
<td>0 to 340° position feedback or continuous rotation</td>
<td>203</td>
</tr>
</tbody>
</table>

**TABLE B.2 - SLINGSBY A206 MANIPULATOR GEOMETRY SPECIFICATIONS**

183
Features:

- High accuracy resolvers for joint angle measurements.
- Separate compensation system for electrical enclosures.
- Rotary actuators at all joints
- Separate compensation system for all rotary actuator bearing enclosures.
- Down arm services supplied through slip-rings at joints
- Water flushing system and secondary dirt seals between all moving surfaces.
- Internal tool supply routed through to the wrist rotate joint.

The accuracy and repeatability of the manipulator are not defined by the manufacturers. It has not been formally calibrated but the estimates of its accuracy and repeatability (made by General Robotics Ltd) in operation are:

<table>
<thead>
<tr>
<th>Absolute accuracy</th>
<th>Accuracy tracking between defined points</th>
<th>Repeatability</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-10mm</td>
<td>1-3mm</td>
<td>0.5-1mm</td>
</tr>
</tbody>
</table>

TABLE B.3 - ACCURACY AND REPEATABILITY OF THE SLINGSBY A206 MANIPULATOR
C.1. Supervisory Control Software

All the supervisory control software is written in C++ using the Microsoft Visual C++ Compiler V 1.0. The code shown here is laid out in the order of these programs and header files:

1) all_def.h Address and constant definitions for program Track4.cpp
2) track4.cpp Main program
3) JCDaq100.h Address and constant definitions for program JCDaq100.cpp
4) JCDaq100.cpp Class to handle the ADC board
5) JCControl.h Address and constant definitions for program JCControl.cpp
6) JCControl.cpp Class to handle force control decisions
7) JCAnn.h Address and constant definitions for program JCAnn.cpp
8) JCAnn.cpp Class to handle the ANN
9) JCMatrix.h Address and constant definitions for program JCMatrix.cpp
10) JCMatrix.cpp Class to handle matrix and vector calculations
11) JCPuma.h Address and constant definitions for program JCPuma.cpp
12) JCPuma.cpp Class to handle serial communication with the PUMA

C.1.1. all_def.h

```c
// Define Constants for JCDaq100.cpp
// Daq100a ioboard definitions
#define CHANNEL_GAIN DgainX1
#define PORT_ADDRESS PORT_0320
#define DMA_ADDRESS DMA5 + 10
#define ENABLE_TAG -1
#define DISABLE_TAG 0
#define CHANNEL_0 0
#define CHANNEL_1 1
#define CHANNEL_2 2
```
#define CHANNEL_3 3
#define CHANNEL_4 4
#define CHANNEL_5 5
#define CHANNEL_6 6
#define CHANNEL_7 7
#define START_CHANNEL 0
#define END_CHANNEL 5

// define constants for JCAnn.cpp

// values to calibrate voltage read into true forces and torques
#define FORCE_SCALE_1 2047
#define FORCE_SCALE_2 409.6
#define A_0 8.2089
#define B_0 0.0353
#define A_1 8.0684
#define B_1 0.0372
#define A_2 8.6804
#define B_2 0.0676
#define A_3 0.3813
#define B_3 0.0000
#define A_4 0.3857
#define B_4 0.0006
#define A_5 0.4093
#define B_5 0.0011
#define DROOP_ANGLE 8 // probe droop relative to flange (deg)

// Define Constants for JCAnn.cpp

// Co-ordinates of the probe (centre of force)
// relative to the flange in tool co-ords
#define INIT_PROBE_FLANGE_0
#define INIT_PROBE_FLANGE_5
#define INIT_PROBE_FLANGE_7
#define INIT_PROBE_FLANGE_10
#define INIT_PROBE_FLANGE_11
#define INIT_PROBE_FLANGE_15

// PUMA vector of initial flange position and orientation
// relative to the base
#define INIT_FLANGE_OAT_0 1
#define INIT_FLANGE_OAT_1 14
#define INIT_FLANGE_OAT_2 1
#define INIT_FLANGE_OAT_3 51.75
#define INIT_FLANGE_OAT_4 1

// Vector of left heel relative to centre of force in probe co-ords
#define LEFT_HEEL_X -27.5
#define LEFT_HEEL_Y 29
#define LEFT_HEEL_Z -20.5

// Vector of right heel relative to centre of force in probe co-ords
#define RIGHT_HEEL_X 27.5
#define RIGHT_HEEL_Y 29
#define RIGHT_HEEL_Z -20.5
// Vector of toe relative to centre of force in probe co-ords
// [10.0 26.5 56.5]
define TOE_X 0
#define TOE_Y 26.5
#define TOE_Z 56.5
#define PI 3.14159265359

// Define Constants for JCPuma.cpp
// Define function codes for COMBIOS
#define FN_EXISTS 0 // Function 0 = COMBIOS exists
#define FN_START 1 // Function 1 = Start Hardware Interrupts
#define FN_STOP 2 // Function 2 = Stop Hardware Interrupts
#define FN_RXCHAR 3 // Function 3 = Receive character
#define FN_TXCHAR 4 // Function 4 = Transmit character
#define FN_RXREADY 5 // Function 5 = Receive queue length
#define FN_RXCLEAR 6 // Function 6 = Clear receive queue
#define FN_TXREADY 7 // Function 7 = Transmitter Ready
#define FN_PORTNO 8 // Function 8 = Number of ports
#define FN_MODEMST 9 // Function 9 = Modem Status Register
#define FN_MODEMCTRL 10 // Function 10 = Modem control register
#define FN_BREAK 11 // Function 11 = Send a Break
#define Esc 27 // Exit with Escape character = ASCII 27
#define INTERRUPT_NUMBER 99 // Software interrupt number
#define INTERRUPT_CHANNEL 0 // Software interrupt channel

// Define byte stuffing constants
#define DEL 0xFF
#define DLE 0x90
#define STX 0x82
#define ETX 0x83
#define START_CHECK 0x00
#define MASK_VALUE 0x60
#define ERR_MASK_VALUE 0x7
#define ALT_MASK_VALUE 0x4
#define ZERO 0
#define DOUBLE_ZERO 0.0
#define HEX_ZERO 0x0

// Define Constants for JCContrl.cpp
// Define PUMA speeds for free space motion (TOOL CO-ORDS?)
#define FREE_SPEED_X 0.028 // PUMA X speed in mm/s
#define FREE_SPEED_Y 10 // PUMA Y speed in mm/s
#define FREE_SPEED_Z 0 // PUMA Z speed in mm/s
#define FREE_ROTN_X 0 // PUMA Xrot speed in deg/s
#define FREE_ROTN_Y 0 // PUMA Yrot speed in deg/s
#define FREE_ROTN_Z 0 // PUMA Zrot speed in deg/s
#define DISTANCE_SCALE 0.028
#define LINEAR_SCALE 32
#define ROTN_SCALE 182.0444
#define DEGREES_TO_RADIANS 0.017453293 // PI/180 converts rads to degrees
#define RADIANS_TO_DEGREES 57.29577951 // 180/PI converts degrees to rads
#define CONTACT_THRESHOLD 1

#define MAXIMUM_TRANSLATION_DISTANCE 0.56 // equiv to 20mm/s
#define MAXIMUM_ROTATION_DISTANCE 0.004886921 // equiv to 10 deg/s
#define TIME_STEP 0.028
// PUMA tracking force thresholds
#define TRACK_FORCE_X 0
#define TRACK_FORCE_Y 4
#define TRACK_FORCE_Z 0
#define TRACK_TORQUE_X -0.04
#define TRACK_TORQUE_Y 0
#define TRACK_TORQUE_Z 0

// PUMA force thresholds for deadband
#define DEAD_BAND_FORCE_X 0.5
#define DEAD_BAND_FORCE_Y 0.5
#define DEAD_BAND_FORCE_Z 0.5
#define DEAD_BAND_TORQUE_X 0.015
#define DEAD_BAND_TORQUE_Y 0.015
#define DEAD_BAND_TORQUE_Z 0.015

// PUMA tracking force gains
#define PROPORTIONAL_GAIN_SCALE 0.001 // multiplied by gain gives true gain
#define PROPORTIONAL_GAIN_TX -20 // must be negative
#define PROPORTIONAL_GAIN_TY -50 // must be negative
#define PROPORTIONAL_GAIN_TZ 0 // must be positive
#define PROPORTIONAL_GAIN_RX -20 // must be negative
#define PROPORTIONAL_GAIN_RY -20 // must be negative
#define PROPORTIONAL_GAIN_RZ 10 // must be positive

#define DIFFERENTIAL_GAIN_SCALE 0.00000001 // multiplied by gain gives true gain
#define DIFFERENTIAL_GAIN_TX 0
#define DIFFERENTIAL_GAIN_TY 100 // must be positive
#define DIFFERENTIAL_GAIN_TZ 0
#define DIFFERENTIAL_GAIN_RX 50 // must be positive
#define DIFFERENTIAL_GAIN_RY 50 // must be positive
#define DIFFERENTIAL_GAIN_RZ -50 // must be negative

#define INTEGRAL_GAIN_SCALE 0 // multiplied by gain gives true gain
#define INTEGRAL_GAIN_TX 0
#define INTEGRAL_GAIN_TY 0
#define INTEGRAL_GAIN_TZ 0
#define INTEGRAL_GAIN_RX 0
#define INTEGRAL_GAIN_RY 0
#define INTEGRAL_GAIN_RZ 0

// Filter coefficients
#define FILTER_ZERO 1 0.13768813589753
#define FILTER_POLE 2 0.13768813589753
#define FILTER_POLE -0.72462372820495

C.1.2. track4.cpp

// FILE track4.cpp

// File Name : track4.cpp
// Function Name : track4
// Purpose : Main program to instruct PUMA to move and modify
// movement according to Force/torque information
// read
// through the A/D board.
//
// Author : J. Tisdall
// Written on : 21/5/96

188
// Copyright (c) Jason Tisdall 1996.

// INCLUDE FILES
#include <math.h>
#include <conio.h>
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include "JCDaqlOO.h"
#include "JCAnn.h"
#include "JCMatrix.h"
#include "JCPuma.h"
#include "JCContrl.h"
#include "all_def.h" // contains constant definitions

// Global output data files
FILE *test;
FILE *force1;
FILE *force2;
FILE *pos1;
FILE *toeheel;

// MAIN PROGRAM BODY
void main(void)
{
    JCDaqlOO ReadCard;
    JCAnn Ann;
    JCMatrix MatMult;
    JCPuma SerialComms;
    JCContrl Move;

    // LOCAL VARIABLES
    int i = ZERO;
    int j = ZERO;
    int hit = ZERO;
    int StableFlag = ZERO;
    int PumaExit = ZERO;
    int ExceptionCode = ZERO;
    int marker = ZERO; // 1 = alter stopping
    int counter = ZERO;

    unsigned char temp;
    unsigned char NullMessage[12];
    unsigned char Message[12];
    unsigned char CumulatedMessageBytes[12];
    short CumulatedMessage[6];
    short ContactMessage[6];
    short OffsetMessage[6];
    short TrueMessage[6];
    short TempMessage = ZERO;
    double TrueForceTorque[6], NullForceTorque[6], FilterForceTorque[6];
    double DroopForceTorque[6], BiasForceTorque[6];
double TempForceTorque[6];
double faTpa[16], faTfd[16], fcTfd[16], fsTfd[16], fsTfc[16], inv_fsTfc[16];
double fsVfd[6], fcVfd[6], faVfd[6];
double StartTransform[16], TrueStartTransform[16], TrueStartVector[6];
double CosTheta, SinTheta;

// Initialise data output files
test = fopen("test.dat","w");
force1 = fopen("allforce.dat","w");
forcez = fopen("filtforc.dat","w");
posl = fopen("allpos.dat","w");
toeheel = fopen("toeheel.dat","w");

// INITIALISE VARIABLES
TempMessage = ZERO;
for(i=0;i<16;i++)
{
    faTpa[i] = DOUBLE_ZERO;
    faTfd[i] = DOUBLE_ZERO;
    fcTfd[i] = DOUBLE_ZERO;
    fsTfd[i] = DOUBLE_ZERO;
    fsTfc[i] = DOUBLE_ZERO;
    inv_fsTfc[i] = DOUBLE_ZERO;
    StartTransform[i] = DOUBLE_ZERO;
    TrueStartTransform[i] = DOUBLE_ZERO;
}
for(i=0;i<12;i++)
{
    NullMessage[i] = HEX_ZERO;
    Message[i] = HEX_ZERO;
    CumulatedMessageBytes[i] = HEX_ZERO;
}
for(i=0;i<6;i++)
{
    CumulatedMessage[i] = ZERO;
    OffsetMessage[i] = ZERO;
    TrueMessage[i] = ZERO;
    TrueForceTorque[i] = DOUBLE_ZERO;
    NullForceTorque[i] = DOUBLE_ZERO;
    FilterForceTorque[i] = DOUBLE_ZERO;
    DroopForceTorque[i] = DOUBLE_ZERO;
    TempForceTorque[i] = DOUBLE_ZERO;
    fsVfd[i] = DOUBLE_ZERO;
    fcVfd[i] = DOUBLE_ZERO;
    faVfd[i] = DOUBLE_ZERO;
    TrueStartVector[i] = DOUBLE_ZERO;
}
for(i=0;i<4;i++) // set fsTfd to identity matrix
{
    fsTfd[i*5] = 1.0;
}
// calculate DroopForceTorque for the droop angle in all_def.h
CosTheta = cos(DROOP_ANGLE*PI/180);
SinTheta = sin(DROOP_ANGLE*PI/180);
DroopForceTorque[1] = -1.4*CosTheta;
DroopForceTorque[3] = 1.4*(0.0177*CosTheta - 0.0108*SinTheta);
// write initial messages to user
cout << "About to start External Alter PUMA." << endl;
cout << "IMPORTANT - SWITCH ON POWER SUPPLY FOR SENSOR BEFORE BEGINNING." << endl;
cout << "Waiting for External Alter to start in PUMA." << endl;
cout << "Hit any key to abort." << endl;

// read initial PUMA position and print it to file
SerialComms.PumaGetStartPosition(StartTransform);
fprintf(test, "PUMA StartTransform = 
");
for (i = 0; i < 16; i++)
    fprintf(test, "%f %f %f %f
", StartTransform[i];
fprintf(test, "PUMA TrueStart Transform = 
");
for (i = 0; i < 16; i++)
    fprintf(test, "%f %f %f %f
", TrueStartTransform[i]);

// create true start transform matrix
// set correct PUMA rotations i.e. PUMA - (90,-90,0)
MatMult.DecomposePUMA(StartTransform, TrueStartVector);
MatMult.Trans(TrueStartVector, 2, 1, TrueStartTransform);
fprintf(test, "PUMA TrueStart Vector = 
");
for (i = 0; i < 6; i++)
    fprintf(test, "%f %f %f %f %f %f
", TrueStartVector[i]);
fprintf(test, "PUMA TrueStart Transform = 
");
for (i = 0; i < 6; i++)
    fprintf(test, "%f %f %f %f
", TrueStartTransform[i]);

// waits for ALTER STARTING from PUMA
while (!(SerialComms.PumaGetInitialMessage())) & (&_kbhit() == 0));

// send initial message to PUMA
SerialComms.PumaSendInitialMessage();

// find non contact (free space) readings and print to file
ReadCard.DaqReadAllSix(NULLForceTorque);
for (i = 0; i < 6; i++)
    BiasForceTorque[i] = NULLForceTorque[i] - DropForceTorque[i];
for (i = 0; i < 6; i++)
    TrueForceTorque[i] = NULLForceTorque[i] - BiasForceTorque[i];
for (i = 0; i < 6; i++)
    TrueForceTorque[i] = NULLForceTorque[i] - BiasForceTorque[i];
for (i = 0; i < 6; i++)
    FilterForceTorque[i] = TrueForceTorque[i];
Move.InitFilterForce(FilterForceTorque);
for (i = 0; i < 6; i++)
    fprintf(force1, "%8.4f %8.4f %8.4f %8.4f
", TrueForceTorque[i];
for (i = 0; i < 6; i++)
    fprintf(force2, "%8.4f %8.4f %8.4f %8.4f
", FilterForceTorque[i];
hit = 0; // set free space marker to zero. This ensures free space section is covered
counter = 0; // extra counter to force transition into contact mode during debug
do // loops until keyboard is hit or PUMA aborts
{
    // READ spring probe: CH0=Fx; CH1=Fy; CH2=Fz; CH3=Tx; CH4=Ty; CH5=Tz;
    ReadCard.DaqReadAllSix(TempForceTorque);
    for(i=0;i<6;i++) TrueForceTorque[i] = TempForceTorque[i] - BiasForceTorque[i];
    for(i=0;i<6;i++) FilterForceTorque[i] = TrueForceTorque[i];
    fprintf(force1, "%8.4f %8.4f %8.4f %8.4f %8.4f %8.4f\n", FilterForceTorque[0],
            FilterForceTorque[1], FilterForceTorque[2], FilterForceTorque[3],
            FilterForceTorque[4], FilterForceTorque[5]);
    Move.FilterForce(FilterForceTorque);
    fprintf(force2, "%8.4f %8.4f %8.4f %8.4f %8.4f %8.4f\n", FilterForceTorque[0],
            FilterForceTorque[1], FilterForceTorque[2], FilterForceTorque[3],
            FilterForceTorque[4], FilterForceTorque[5]);
}

    // calculate toe and heel positions and print to file
    // do ANN calculation and return transformation of probe relative to flange
    Ann.CDTransform(FilterForceTorque, faTpa);

    // check for contact. Sets hit = 1 if Fy has changed by more than THRESHOLD
    // currently this is a once only switch. It may be necessary at some
    // later stage to change it to a sliding mode switch that can flip back
    // and forth between the two states.
    if(fabs(FilterForceTorque[1])<CONTACT_THRESHOLD) hit = 1;

    // calculate required move
    if(hit==0) // do free space moves
    {
        // calculate new posn according to free space algorithm
        Move.ControlFreeSpace(faTfd);
    }

    if((hit==1)&&(StableFlag==0)) // move into initial contact force equilibrium
    {
        // calculate new posn according to control algorithm
        Move.ControlForce(FilterForceTorque, faTfd, &StableFlag);
    }

    if((hit==1)&&(StableFlag==1)) // move along surface in probe z direction
    {
        // calculate new posn according to control algorithm
        Move.ControlAnn(FilterForceTorque, faTfd, faTpa);
    }

    // new fsTfd = old fsTfd*faTfd
    MatMult.HMat4(fsTfd, faTfd, fsTfd);
    MatMult.Decompose(fsTfd, fsVfd);

    // set current value of cumulated message to the contact value
    if(hit==0) for(j=0;j<16;j++) fsTfc[j] = fsTfd[j];

    // Convert radians to degrees
    for(j=3;j<6;j++) fsVfd[j] = fsVfd[j]*RADIANS_TO_DEGREES;

    // convert to the correct PUMA scale
    for(j=0;j<3;j++) fsVfd[j] = fsVfd[j]*LINEAR_SCALE;
    for(j=3;j<6;j++) fsVfd[j] = fsVfd[j]*ROTN_SCALE;
// calculate the cumulated message to send to PUMA in non-cumulative mode
for(j=0;j<6;j++)
{
    CumulatedMessageBytes[j*2] = (unsigned char)((short)fsVfd[j] & 0xFF);
    CumulatedMessageBytes[j*2+1] = (unsigned char)((short)fsVfd[j] >> 8);
}
SerialComms.PumaGetMessage(&marker); // receive PUMA message
if(marker==1) PumaExit = 1; // flag loop breaker if PUMA sends ALTER STOPPING

// send path modification data to PUMA
SerialComms.PumaSendMessage(CumulatedMessageBytes, &ExceptionCode);
}

while( (_kbhit()==0) && (PumaExit==0) );

//------------- IF KEYBOARD HIT, ABORT PUMA PROGRAM -------------------
if(!(_kbhit()==0)) // send exception code to trip subroutine and stop PUMA
{
    i = 0;
    marker = 0;
    if(!(_kbhit()==0)) temp = _getch(); // clear keyboard buffer
    ExceptionCode = -1;
    SerialComms.PumaGetMessage(&marker);
    SerialComms.PumaSendMessage(NullMessage, &ExceptionCode); // trip subroutine
    do // read from PUMA until alter stopping received
    {
        SerialComms.PumaGetMessage(&marker);
        if(marker==1) i = 1; // flag loop breaker if PUMA sends ALTER STOPPING
        fprintf(test," marker = %d \n",marker);
        SerialComms.PumaSendMessage(NullMessage, &ExceptionCode);
    } while((_kbhit()==0) && (i==0));
}
fclose(test);
close(force1);
close(force2);
close(pos1);
close(toeheel);

/
/
/
/
/

C.1.3. JCDaq100.h

// FILE JCDaq100.h

 ifndef JCDAQ100
#define JCDAQ100

class JCDaq100
{
    public:
    JCDaq100();
    ~JCDaq100();

    // Attributes
    public:

protected:
  double m_dCh0;
  double m_dCh1;
  double m_dCh2;
  double m_dCh3;
  double m_dCh4;
  double m_dCh5;
  double m_dCh6;
  double m_dCh7;

// Operations
public:
  double DaqReadCh0(); // single read of channel 0
  double DaqReadCh1(); // single read of channel 1
  double DaqReadCh2(); // single read of channel 2
  double DaqReadCh3(); // single read of channel 3
  double DaqReadCh4(); // single read of channel 4
  double DaqReadCh5(); // single read of channel 5
  double DaqReadCh6(); // single read of channel 6
  double DaqReadCh7(); // single read of channel 7
  void DaqReadAllSix(double dData[6]); // single read of six channels (first
  // and last defined in all_def.h)

protected:
};
#endif

C.1.4. JCDaq100.cpp

// FILE JCDaq100.cpp

// INCLUDE FILES
#include "JCDaq100.h"
#include "daqbook.h"
#include "all_def.h"

// CONSTRUCTOR
JCDaq100::JCDaq100()
{
  m_dCh0 = 0.0;
  m_dCh1 = 0.0;
  m_dCh2 = 0.0;
  m_dCh3 = 0.0;
  m_dCh4 = 0.0;
  m_dCh5 = 0.0;
  m_dCh6 = 0.0;
  m_dCh7 = 0.0;

  daqInit(PORT_ADDRESS,DMA_ADDRESS); // initialise daqboard
daqAdcSetTag(DISABLE_TAG); // disable channel tagging

// DESTRUCTOR---------------------------------------------------------------
JCDaq100::~JCDaq100()
{
    daqClose(); // close and exit
}

// MEMBER FUNCTIONS--------------------------------------------------------
double JCDaq100::DaqReadCh0()
{
    unsigned int nSingleData;
    daqAdcRd(CHANNEL_0,&nSingleData,CHANNEL_GAIN);
    // Voltage scaling calculation
    m_dCh0 = (nSingleData-FORCE_SCALE_1)/FORCE_SCALE_2;
    return m_dCh0;
}

double JCDaq100::DaqReadCh1()
{
    unsigned int nSingleData;
    daqAdcRd(CHANNEL_1,&nSingleData,CHANNEL_GAIN);
    // Voltage scaling calculation
    m_dCh1 = (nSingleData-FORCE_SCALE_1)/FORCE_SCALE_2;
    return m_dCh1;
}

double JCDaq100::DaqReadCh2()
{
    unsigned int nSingleData;
    daqAdcRd(CHANNEL_2,&nSingleData,CHANNEL_GAIN);
    // Voltage scaling calculation
    m_dCh2 = (nSingleData-FORCE_SCALE_1)/FORCE_SCALE_2;
    return m_dCh2;
}

double JCDaq100::DaqReadCh3()
{
    unsigned int nSingleData;
    daqAdcRd(CHANNEL_3,&nSingleData,CHANNEL_GAIN);
    // Voltage scaling calculation
    m_dCh3 = (nSingleData-FORCE_SCALE_1)/FORCE_SCALE_2;
    return m_dCh3;
}

double JCDaq100::DaqReadCh4()
{
    unsigned int nSingleData;
    daqAdcRd(CHANNEL_4,&nSingleData,CHANNEL_GAIN);
    // Voltage scaling calculation
m_dCh4 = (nSingleData-FORCE_SCALE_1)/FORCE_SCALE_2;
return m_dCh4;
}

double JCDaq100::DaqReadCh5()
{
    unsigned int nSingleData;
    daqAdcRd(CHANNEL_5,&nSingleData,CHANNEL_GAIN);
    // Voltage scaling calculation
    m_dCh5 = (nSingleData-FORCE_SCALE_1)/FORCE_SCALE_2;
    return m_dCh5;
}

double JCDaq100::DaqReadCh6()
{
    unsigned int nSingleData;
    daqAdcRd(CHANNEL_6,&nSingleData,CHANNEL_GAIN);
    // Voltage scaling calculation
    m_dCh6 = (nSingleData-FORCE_SCALE_1)/FORCE_SCALE_2;
    return m_dCh6;
}

double JCDaq100::DaqReadCh7()
{
    unsigned int nSingleData;
    daqAdcRd(CHANNEL_7,&nSingleData,CHANNEL_GAIN);
    // Voltage scaling calculation
    m_dCh7 = (nSingleData-FORCE_SCALE_1)/FORCE_SCALE_2;
    return m_dCh7;
}

void JCDaq100::DaqReadAllSix(double dData[6])
{
    unsigned int nData[6];
    daqAdcRdScan(START_CHANNEL,END_CHANNEL,nData,CHANNEL_GAIN);
    // Voltage scaling calculation
    for(int i=0;i<6;i++) dData[i] = ((double)nData[i]-FORCE_SCALE_1)/FORCE_SCALE_2;

    dData[0] = dData[0]*A_0 + B_0;
    dData[1] = dData[1]*A_1 + B_1;
    dData[2] = dData[2]*A_2 + B_2;
    dData[3] = dData[3]*A_3 + B_3;
    dData[4] = dData[4]*A_4 + B_4;
    dData[5] = dData[5]*A_5 + B_5;
}
#ifndef JCCONTRL
#define JCCONTRL

#include "JCMatrix.h"
#include "JCAnn.h"

class JCContrl
{
public:
    JCContrl();
    ~JCContrl();

    // Attributes
    public:
    //private:
    double m_dTimeStep;
    double m_dfaVfdNew[6];
    double m_dfaVfdOld[6];
    double m_dForcePreset[6];
    double m_dProportionalGain[6];
    double m_dDifferentialGain[6];
    double m_dIntegralGain[6];
    double m_dDeadBand[6];
    double m_dErrorNew[6];
    double m_dErrorOld1[6];
    double m_dErrorOld2[6];
    double m_dOldForceTorque[6];
    double m_dOldFilteredForceTorque[6];
    double m_dOldAnn[6];
    double m_dOldFilteredAnn[6];
    double m_dFilterZero1;
    double m_dFilterZero2;
    double m_dFilterPole2;

    JCMatrix MatFunc;
    JCAnn AnnFunc;

    // Operations
    public:
    void JCContrl::ContrlFreeSpace(double faTfd[16]);
    void JCContrl::ContrlForce(double TrueForceTorque[6], double faTfd[16], int *StableFlag);
    void JCContrl::ContrlAnn(double TrueForceTorque[6], double faTfd[16], double faTpa[16]);
    void JCContrl::InitFilterForce(double TrueForceTorque[6]);
    void JCContrl::FilterForce(double TrueForceTorque[6]);
    void JCContrl::InitFilterAnn(double AnnOutput[6]);
    void JCContrl::FilterAnn(double AnnOutput[6]);

protected:
    
};
#endif
C.1.6. JCContrl.cpp

// FILE      JCContrl.cpp

// File Name : JCContrl.cpp
// Function Name : JCContrl
// Purpose : Class to read A/D board
//
// Author : J. Tisdall
// Written on : 22/5/96

// INCLUDE FILES
#include "JCContrl.h"
#include <stdio.h>
#include <math.h>
#include "all def.h"
extern FILE *test;
extern FILE *allfatpa;

// CONSTRUCTOR
----------------------------------------------------------
JCContrl::JCContrl()
{
    // Required force presets
    m_dForcePreset[0] = TRACK_FORCE_X;
    m_dForcePreset[1] = TRACK_FORCE_Y;
    m_dForcePreset[2] = TRACK_FORCE_Z;
    m_dForcePreset[3] = TRACK_TORQUE_X;
    m_dForcePreset[4] = TRACK_TORQUE_Y;
    m_dForcePreset[5] = TRACK_TORQUE_Z;

    m_dDeadBand[0] = DEAD_BAND_FORCE_X;
    m_dDeadBand[1] = DEAD_BAND_FORCE_Y;
    m_dDeadBand[2] = DEAD_BAND_FORCE_Z;
    m_dDeadBand[3] = DEAD_BAND_TORQUE_X;
    m_dDeadBand[4] = DEAD_BAND_TORQUE_Y;
    m_dDeadBand[5] = DEAD_BAND_TORQUE_Z;

    m_dProportionalGain[0] = PROPORTIONAL_GAIN_TX*PROPORTIONAL_GAIN_SCALE;
    m_dProportionalGain[1] = PROPORTIONAL_GAIN_TY*PROPORTIONAL_GAIN_SCALE;
    m_dProportionalGain[2] = PROPORTIONAL_GAIN_TZ*PROPORTIONAL_GAIN_SCALE;
    m_dProportionalGain[3] = PROPORTIONAL_GAIN_RX*PROPORTIONAL_GAIN_SCALE;
    m_dProportionalGain[4] = PROPORTIONAL_GAIN_RY*PROPORTIONAL_GAIN_SCALE;
    m_dProportionalGain[5] = PROPORTIONAL_GAIN_RZ*PROPORTIONAL_GAIN_SCALE;

    m_dDifferentialGain[0] = DIFFERENTIAL_GAIN_TX*DIFFERENTIAL_GAIN_SCALE;
    m_dDifferentialGain[1] = DIFFERENTIAL_GAIN_TY*DIFFERENTIAL_GAIN_SCALE;

    m_dIntegralGain[0] = INTEGRAL_GAIN_TX*INTEGRAL_GAIN_SCALE;
    m_dIntegralGain[1] = INTEGRAL_GAIN_TY*INTEGRAL_GAIN_SCALE;
    m_dIntegralGain[2] = INTEGRAL_GAIN_TZ*INTEGRAL_GAIN_SCALE;
    m_dIntegralGain[3] = INTEGRAL_GAIN_RX*INTEGRAL_GAIN_SCALE;
    m_dIntegralGain[4] = INTEGRAL_GAIN_RY*INTEGRAL_GAIN_SCALE;
    m_dIntegralGain[5] = INTEGRAL_GAIN_RZ*INTEGRAL_GAIN_SCALE;
}
m_dTimeStep = TIME_STEP;
for (int i=0;i<6;i++)
{
    m_dfaVfdNew[i] = DOUBLE_ZERO;
    m_dfaVfdOld[i] = DOUBLE_ZERO;
    m_dErrorNew[i] = DOUBLE_ZERO;
    m_dErrorOld1[i] = DOUBLE_ZERO;
    m_dErrorOld2[i] = DOUBLE_ZERO;
}

// filter coefficients
m_dFilterZero1 = FILTER_ZERO_1;
m_dFilterZero2 = FILTER_ZERO_2;
m_dFilterPole2 = FILTER_POLE_2;

// DESTRUCTOR ---------------------------------------------------------------
JCCContrl::~JCCContrl()
{
    ;
}

// MEMBER FUNCTIONS --------------------------------------------------------

void JCCContrl::ContrlFreeSpace(double faTfd[16])
{
    double new_pos[6];
    // distance in mm/cycle and rad/cycle
    new_pos[0] = FREE_SPEED_X*DISTANCE_SCALE;
    new_pos[1] = FREE_SPEED_Y*DISTANCE_SCALE;
    new_pos[3] = FREE.RotateX*DEGREES_TO_RADIANS*DISTANCE_SCALE;
    new_pos[4] = FREE.RotateY*DEGREES_TO_RADIANS*DISTANCE_SCALE;
    new_pos[5] = FREE.RotateZ*DEGREES_TO_RADIANS*DISTANCE_SCALE;
    MatFunc.Trans(new_pos, 2, 2, faTfd); // rotating axes xyz
    // from D:\JAS_WORK\CWU\SPR\SURF_MOD\CYLINDER\chktool.m it is apparent
    // that the tool rotates about rotating XYZ axes
}

void JCCContrl::ContrlForce(double TrueForceTorque[6], double faTfd[16], int *StableFlag)
{
    double temp1, temp2, temp3, temp4, temp5;
    unsigned int InDeadBand = 0x0;
    int i;

    // required motion in flange coords from PID controller
    for (i=0;i<6;i++)
    {
        // set old values to current values
        m_dErrorOld2[i] = m_dErrorOld1[i];
        m_dErrorOld1[i] = m_dErrorNew[i];
        m_dfaVfdOld[i] = m_dfaVfdNew[i];

        // calculate the current error
        m_dErrorNew[i] = (TrueForceTorque[i] - m_dForcePreset[i]);
    }
// perform difference equation calculation
    temp1 = m_dDifferentialGain[i]/m_dTimeStep;
    temp2 = m_dIntegralGain[i]*m_dTimeStep;
    temp3 = (m_dProportionalGain[i] + temp1 + temp2)*m_dErrorNew[i];
    temp4 = (m_dProportionalGain[i] + 2*temp1)*m_dErrorOld1[i];
    temp5 = temp1*m_dErrorOld2[i];
    m_dfaVfdNew[i] = m_dfaVfdOld[i] + temp3 - temp4 - temp5;

// include a deadband
if (fabs(m_dErrorNew[i])<m_dDeadBand[i])
{
    m_dfaVfdNew[i] = 0;
    InDeadBand = (InDeadBand|(1«(i)));// set bit when it reaches deadband
}

// calculate the new transformation of desired flange position relative to actual flange position
int Flag1 = 1; // pre-multiplying about fixed axes
int Flag2 = 2; // rotating about xyz axes
MatFunc.Trans(m_dfaVfdNew, Flag1, Flag2, faTfd);

// check if all forces are inside deadband
if((InDeadBand&0x3a)==0x3a) *StableFlag= 1;
}

void JCControll::ContrlAnn(double TrueForceTorque[6], double faTfd[16], double faTpa[16])
{
    double inv_faTpa[16], faVfd[6],paVpd[6], paTpdl[16], Temp6[16];
    double temp1, temp2, temp3, temp4, temp5;
    unsigned char InDeadBand = 0x0;
    int i;

    // required motion in flange coords from PID controller
    for (i=0;i<6;i++)
    {
        // set old values to current values
        m_dErrorOld2[i] = m_dErrorOld1[i];
        m_dErrorOld1[i] = m_dErrorNew[i];
        m_dfaVfdOld[i] = m_dfaVfdNew[i];

        // calculate the current error
        m_dErrorNew[i] = (TrueForceTorque[i] - m_dForcePreset[i]);

        // perform difference equation calculation
        temp1 = m_dDifferentialGain[i]/m_dTimeStep;
        temp2 = m_dIntegralGain[i]*m_dTimeStep;
        temp3 = (m_dProportionalGain[i] + temp1 + temp2)*m_dErrorNew[i];
        temp4 = (m_dProportionalGain[i] + 2*temp1)*m_dErrorOld1[i];
        temp5 = temp1*m_dErrorOld2[i];
        m_dfaVfdNew[i] = m_dfaVfdOld[i] + temp3 - temp4 - temp5;

        // include a deadband
        if (fabs(m_dErrorNew[i])<m_dDeadBand[i])
        {
            m_dfaVfdNew[i] = 0;
            InDeadBand = (InDeadBand&1<«(i-1));
        }
    }

    // required motion in tool coords. This will actually be
// calculated from the force differentials
paVpd[0] = 0.0;
paVpd[1] = 0.0;
paVpd[2] = 4.0*DISTANCE_SCALE;
paVpd[3] = 0.0;
paVpd[4] = 0.0;//2*PI/180;
paVpd[5] = 0.0;

int Flag1 = 1; // pre-multiplying about fixed axes
int Flag2 = 2; // rotating about xyz axes
MatFunc.Trans(paVpd, Flag1, Flag2, paTpd);

// faTfd = faTpa*paTpd*hom_inv(faTpa)
MatFunc.HomInv(faTpa, inv_faTpa);
MatFunc.MatXMat4(faTpa, paTpd, Temp6);
MatFunc.MatXMat4(Temp6, inv_faTpa, faTfd);

// calculate the vector solution
MatFunc.Decompose(faTfd, faVfd);

// print out diagnostic files
fprintf(test."motion from ANN = % 7.2f % 7.2f % 7.2f % 7.2f % 7.2f % 7.2f \n", faVfd[0],
faVfd[1], faVfd[2], faVfd[3], faVfd[4], faVfd[5]);
fprintf(test."motion from force = % 7.2f % 7.2f % 7.2f % 7.2f % 7.2f % 7.2f \n", m_dfaVfdNew[0],
m_dfaVfdNew[1], m_dfaVfdNew[2], m_dfaVfdNew[3], m_dfaVfdNew[4], m_dfaVfdNew[5]);

// recalculate the transformation
MatFunc.Trans(faVfd, Flag1, Flag2, faTfd);

void JCContrl::InitFilterForce(double TrueForceTorque[6])
{
    for (int i=0;i<6;i++)
    {
        m_dOldForceTorque[i] = TrueForceTorque[i];
        m_dOldFilteredForceTorque[i] = TrueForceTorque[i];
    }
}

void JCContrl::FilterForce(double TrueForceTorque[6])
{
    double FilteredForce[6];

    for (int i=0;i<6;i++)
    {
        FilteredForce[i] = m_dFilterZero1*TrueForceTorque[i] +
                           m_dFilterZero2*m_dOldForceTorque[i] -
                           m_dFilterPole2*m_dOldFilteredForceTorque[i];
        m_dOldForceTorque[i] = TrueForceTorque[i];
        TrueForceTorque[i] = FilteredForce[i];
        m_dOldFilteredForceTorque[i] = FilteredForce[i];
    }
void JCContrl::InitFilterAnn(double AnnOutput[6])
{
    for (int i=0; i<6; i++)
    {
        m_dOldAnn[i] = AnnOutput[i];
        m_dOldFilteredAnn[i] = AnnOutput[i];
    }
}

void JCContrl::FilterAnn(double AnnOutput[6])
{
    double FilteredAnn[6];
    for (int i=0; i<6; i++)
    {
        FilteredAnn[i] = m_dFilterZero1*AnnOutput[i] + 
                         m_dFilterZero2*m_dOldAnn[i] - 
                         m_dFilterPole2*m_dOldFilteredAnn[i];
        m_dOldAnn[i] = AnnOutput[i];
        AnnOutput[i] = FilteredAnn[i];
        m_dOldFilteredAnn[i] = FilteredAnn[i];
    }
}

C.1.7. JCAnn.h

#include "JCMatrix.h"

class JCAnn
{
    public:
        JCAnn();
        ~JCAnn();

    // Attributes
    public:
        // protected:
    private:
        double m_dB1[11]; // 11x1
        double m_dB2[6]; // 6x1
        double m_dW1[66]; // 11x6
        double m_dW2[66]; // 6x11
        double m_dMinP[6]; // 6x1
        double m_dMinT[6]; // 6x1
        double m_dInitFlange[6];
        double m_dInitProbeFlange[16];
        double m_dInitProbePuma[16];
        double m_dInitFlangeOat[6];
double m_dLocalToe[4];
double m_dLocalLeftHeel[4];
double m_dLocalRightHeel[4];

JCMatrix MatMult;

// Operations
public:
void JCAnn::ReturnB1(double dB1[11]);
void JCAnn::ReturnB2(double dB2[6]);
void JCAnn::CDTransform(double TrueForceTorque[6], double faTpa[16]);
void JCAnn::ToePosition(double TrueForceTorque[6], double fsTfd[16], double bTCfs[16],
double faTpa[16]);

protected:
};

#endif

C.1.8. JCAnn.cpp

---

// FILE : JCAnn.cpp
---

// File Name : JCAnn.cpp
// Function Name : JCAnn
// Purpose : Defines the ANN class to:
// 1) reads in data files for the ANN from data files
// MIN_P.dat, MIN_T.dat, RAT_P.dat, RAT_T.dat
// 2) Also defines some of the other transformations
// Author : J. Tisdall
// Written on : 16/4/96
---

// Copyright (c) Jason Tisdall 1996.
---

// INCLUDE FILES ---------------------------------------------
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <math.h>
#include "JCAnn.h"
#include "all_def.h"

extern FILE *test;
extern FILE *toeheel;

// CONSTRUCTOR ---------------------------------------------
JCAnn::JCAnn()
{
// READ ANN DATA FILES -----------------------------------------
int i,j,k;
int FILE_B1, FILE_B2, FILE_W1, FILE_W2;
int FILE_RATP, FILE_RATT, FILE_MINP, FILE_MINT;
char buffer[2000];
char tempbuffer[60];
int bytesread;
unsigned int numbytes = 2000;

// read B1
FILE_B1 = _open("B1col_11.dat",_O_RDONLY);  // open data file to read
bytesread = _read(FILE_B1,buffer,numbytes); // read data file
_close(FILE_B1);  // close data file
buffer[bytesread] = '0';  // put null at end of data file

j=0; k=0;
for(i=0;i<11;i++)
{
    while (!(buffer[j]==0xa))
    {
        tempbuffer[k] = buffer[j]; j++; k++;
    }
    tempbuffer[k] = '0'; j++; k=0;
    m_dB1[i] = atof(tempbuffer);  // convert to number using atof
}

// read B2
FILE_B2 = _open("B2col_11.dat",_O_RDONLY);  // open data file to read
bytesread = _read(FILE_B2,buffer,numbytes); // read data file
_close(FILE_B2);  // close data file
buffer[bytesread] = '0';  // put null at end of data file

j=0; k=0;
for(i=0;i<6;i++)
{
    while (!(buffer[j]==0xa))
    {
        tempbuffer[k] = buffer[j]; j++; k++;
    }
    tempbuffer[k] = '0'; j++; k=0;
    m_dB2[i] = atof(tempbuffer);  // convert to number using atof
}

// read W1
FILE_W1 = _open("W1col_11.dat",_O_RDONLY);  // open data file to read
bytesread = _read(FILE_W1,buffer,numbytes); // read data file
_close(FILE_W1);  // close data file
buffer[bytesread] = '0';  // put null at end of data file

j=0; k=0;
for(i=0;i<66;i++)
{
    while (!(buffer[j]==0xa))
    {
        tempbuffer[k] = buffer[j]; j++; k++;
    }
    tempbuffer[k] = '0'; j++; k=0;
    m_dW1[i] = atof(tempbuffer);  // convert to number using atof
}

// read W2
FILE_W2 = _open("W2col_11.dat",_O_RDONLY);  // open data file to read
bytesread = _read(FILE_W2,buffer,numbytes); // read data file
_close(FILE_W2);  // close data file
buffer[bytesread] = '0';  // put null at end of data file
j=0; k=0; for(i=0; i<66; i++)
{
    while (! (buffer[j]==0xa))
    {
        tempbuffer[k] = buffer[j]; j++; k++;
    }
    tempbuffer[k] = '0'; j++; k=0;
m_dW2[i] = atof(tempbuffer);
}

// read RAT_P
FILE_RATP = _open("rat_p.dat",_O_RDONLY); // open data file to read
bytesread = _read(FILE_RATP,buffer,numbytes); // read data file
_close(FILE_RATP); // close data file
buffer[bytesread] = '0'; // put null at end of data file
j=0; k=0; for(i=0; i<6; i++)
{
    while (! (buffer[j]==0xa))
    {
        tempbuffer[k] = buffer[j]; j++; k++;
    }
    tempbuffer[k] = '0'; j++; k=0;
m_dRatP[i] = atof(tempbuffer);
}

// read RAT_T
FILE_RATT = _open("rat_t.dat",_O_RDONLY); // open data file to read
bytesread = _read(FILE_RATT,buffer,numbytes); // read data file
_close(FILE_RATT); // close data file
buffer[bytesread] = '0'; // put null at end of data file
j=0; k=0; for(i=0; i<6; i++)
{
    while (! (buffer[j]==0xa))
    {
        tempbuffer[k] = buffer[j]; j++; k++;
    }
    tempbuffer[k] = '0'; j++; k=0;
m_dRatT[i] = atof(tempbuffer);
}

// read MIN_P
FILE_MINP = _open("min_p.dat",_O_RDONLY); // open data file to read
bytesread = _read(FILE_MINP,buffer,numbytes); // read data file
_close(FILE_MINP); // close data file
buffer[bytesread] = '0'; // put null at end of data file
j=0; k=0; for(i=0; i<6; i++)
{
    while (! (buffer[j]==0xa))
    {
        tempbuffer[k] = buffer[j]; j++; k++;
    }
    tempbuffer[k] = '0'; j++; k=0;
m_dMinP[i] = atof(tempbuffer);
FILE_MINT = _open("min_t.dat",_O_RDONLY); // open file to read
bytesread = _read(FILE_MINT,bufiFer,numbyles); // read data file
_close(FILE_MINT); // close data file
buffer[bytesread] = '0'; // put null at end of data file

j=0; k=0;
for(i=0;i<6;i++)
{
    while (!((buffer[j]===0xa))
    {
        tempbuffer[k] = buffer[j]; j++; k++;
    }
    tempbuffer[k] = '0'; j++; k=0;
    m_dMinT[i] = atof(tempbuffer);
}

// initialise PUMA transformations
for(i=0;i<16;i++)
{
    m_dInitProbePuma[i] = DOUBLE_ZERO;
    m_dInitProbeFlange[i] = DOUBLE_ZERO;
}

m_dInitProbeFlange[0] = INIT_PROBE_FLANGE_0;
m_dInitProbeFlange[5] = INIT_PROBE_FLANGE_5;
m_dInitProbeFlange[7] = INIT_PROBE_FLANGE_7;
m_dInitProbeFlange[10] = INIT_PROBE_FLANGE_10;
m_dInitProbeFlange[11] = INIT_PROBE_FLANGE_11;
m_dInitProbeFlange[15] = INIT_PROBE_FLANGE_15;
m_dInitFlangeOat[0] = INIT_FLANGE_OAT_0;
m_dInitFlangeOat[1] = INIT_FLANGE_OAT_1;
m_dInitFlangeOat[2] = INIT_FLANGE_OAT_2;
m_dInitFlangeOat[3] = INIT_FLANGE_OAT_3;
m_dInitFlangeOat[4] = INIT_FLANGE_OAT_4;
m_dInitFlangeOat[5] = INIT_FLANGE_OAT_5;
MatMult.Trans(m_dInitFlangeOat, 2, 1, m_dInitProbePuma);
MatMult.MatXMat4(m_dInitProbePuma,m_dInitProbeFlange,m_dInitProbePuma);

m_dLocalToe[0] = TOE X;
m_dLocalToe[1] = TOE Y;
m_dLocalToe[2] = TOE Z;
m_dLocalToe[3] = 1;
m_dLocalLeftHeel[0] = LEFT_HEEL X;
m_dLocalLeftHeel[1] = LEFT_HEEL Y;
m_dLocalLeftHeel[2] = LEFT_HEEL Z;
m_dLocalLeftHeel[3] = 1;
m_dLocalRightHeel[0] = RIGHT_HEEL X;
m_dLocalRightHeel[1] = RIGHT_HEEL Y;
m_dLocalRightHeel[2] = RIGHT_HEEL Z;
m_dLocalRightHeel[3] = 1;

}
MEMBER FUNCTIONS

```cpp
void JCAnn::ReturnB1(double dB1[11])
{
    for(int i=0;i<11;i++)
        dB1[i] = m_dB1[i];
}

void JCAnn::ReturnB2(double dB2[6])
{
    for(int i=0;i<6;i++)
        dB2[i] = m_dB2[i];
}

void JCAnn::CDTransform(double TnetForceTorque[6], double faTpa[16])
{
    int i,j;
    double ScaleForceTorque[6];
    double TruePosition[6];
    double n1[11], nB1[11], out1[11];
    double n2[6], nB2[6], T_net[6];
    double bTft[16];
    double inv_bTft[16];

    // Transform F/T vector into range ±1
    // As per page 4.73, it is necessary to scale the forces using
    // the same scale as in the ANN training phase
    for(i=0;i<6;i++) ScaleForceTorque[i] = (TrueForceTorque[i]-m_dMinP[i])*m_dRatP[i] - 1;

    // calculate ANN output from force data
    for(i=0;i<11;i++)
    {
        n1[i] = 0;
        for(j=0;j<6;j++)
        {
            n1[i] = n1[i] + m_dW1[i*6+j]*ScaleForceTorque[j];  // n1=W1*all_force;
        }
        nB1[i] = n1[i] + m_dB1[i];
        out1[i] = tanh(nB1[i]); // out1=tansig(n1,B1);
    }

    for(i=0;i<6;i++)
    {
        n2[i] = 0;
        for(j=0;j<11;j++)
        {
            n2[i] = n2[i] + m_dW2[i*11+j]*out1[j]; // n2=W2*out1;
        }
        nB2[i] = n2[i] + m_dB2[i];
        T_net[i] = nB2[i]; // T_net=purelin(n2,B2);
    }

    // transform output back to original range & convert to global co-ordinate system
    for(i=0;i<6;i++)
    {
        // ...
    }
}
```
TruePosition[i] = \(((T_{net}[i]+1)/m_{dRatT}[i])+m_{dMinT}[i]\) + m_{dInitFlangeOat}[i];

// convert to a transformation using post (2) and OAT (1) format (rotating axes)
MatMult.Trans(TruePosition, 2, 1, bTft);

// invert the homogeneous transformation
MatMult.HomInv(bTft, inv_bTft);

// solve for ftPp to transform from workpiece to tool co-ords (mat x mat)
MatMult.MatXMat4(inv_bTft.m_dInitProbePumaTaTpa);

void JCAnn::ToePosition(double TrueForceTorque[6], double fsTfd[16], double bTCfs[16], double faTpa[16])
{
    int j;
    double Temp[16], bTpa[16];
    double GlobalToe[4], GlobalLeftHeel[4], GlobalRightHeel[4];

    // Temp = bTCfs.fsTfd
    MatMult.MatXMat4(bTCfs, fsTfd, Temp);

    // bTpa = Temp.faTpa
    MatMult.MatXMat4(Temp, faTpa, bTpa);

    // calculate toe and heel positions
    MatMult.Mat4XMat1(bTpa, m_dLocalToe, GlobalToe);
    MatMult.Mat4XMat1(bTpa, m_dLocalLeftHeel, GlobalLeftHeel);
    MatMult.Mat4XMat1(bTpa, m_dLocalRightHeel, GlobalRightHeel);

    // print toe, leftheel and rightheel to file
    for(j=0; j<3; j++) fprintf(toeheel, "%8.3f ", GlobalToe[j]);
    for(j=0; j<3; j++) fprintf(toeheel, "%8.3f ", GlobalLeftHeel[j]);
    for(j=0; j<3; j++) fprintf(toeheel, "%8.3f ", GlobalRightHeel[j]);
    fprintf(toeheel, "n");
}

C.1.9. JCMatrix.h

==================================
// FILE JCMatrix.h
==================================

#ifndef JCMATRIX
#define JCMATRIX

class JCMatrix
{
public:
    JCMatrix();
    ~JCMatrix();

    // Attributes
    public:
    protected:
private:

208
public:

void JCMatrix::MatXMat4(double Mat1[16], double Mat2[16], double Solution[16]);
void JCMatrix::MatXMat3(double Mat1[9], double Mat2[9], double Solution[9]);
void JCMatrix::Mat4XMat1(double Mat1[16], double Mat2[4], double Solution[4]);
void JCMatrix::HomInv(double Mat[16], double InvMat[16]);
void JCMatrix::Trans(double Vector[6], int Flag1, int Flag2, double Solution[16]);
void JCMatrix::Decompose(double Transformation[16], double Solution[6]);
void JCMatrix::DecomposePUMA(double Transformation[16], double Solution[6]);
protected:

};
#endif

C.1.10. JCMatrix.cpp

// INCLUDE FILES
#include "JCMatrix.h"
#include <math.h>
#include "all_def.h"

// CONSTRUCTOR
---------------------------------------------
JCMatrix::JCMatrix()
{
    ;
}

// DESTRUCTOR
---------------------------------------------
JCMatrix::~JCMatrix()
{
    ;
}

// MEMBER FUNCTIONS
---------------------------------------------
void JCMatrix::MatXMat4(double Mat1[16], double Mat2[16], double Solution[16])
{
    // multiply two 4x4 matrices
    double dTemp[16],
    for(int i=0;i<4;i++)
    {
        for(int j=0;j<4;j++)
        {
            dTemp[i*4+j] = 0;
            for(int k=0;k<4;k++)
            {
                dTemp[i*4+j] = dTemp[i*4+j] + Mat1[i*4+k]*Mat2[k*4+j];
            }
        }
    }
}
for(i=0;i<16;i++) Solution[i] = dTemp[i];

void JCMatrix::Mat4XMatl(double Mat1[16], double Mat2[4], double Solution[4])
{
    // multiply a 4x4 matrix by a 4x1 vector
    double dTemp[4];
    Solution[0] = 0;
    for(int i=0;i<4;i++)
    {
        dTemp[i] = 0;
        for(int j=0;j<4;j++)
        {
            dTemp[i] = dTemp[i] + Mat1[i*4+j]*Mat2[j];
        }
    }
    for(i=0;i<4;i++) Solution[i] = dTemp[i];
}

void JCMatrix::MatXMat3(double Mat1[9], double Mat2[9], double Solution[9])
{
    // multiply two 3x3 matrices
    double dTemp[9];
    Solution[0] = 0;
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            for(int k=0;k<3;k++)
            {
                dTemp[i*3+j] = dTemp[i*3+j] + Mat1[i*3+k]*Mat2[k*3+j];
            }
            for(int j=0;j<3;j++)
            {
                dTemp[i*3+j] = dTemp[i*3+j] + Mat1[i*3+j]*Mat2[j*3+j];
            }
        }
    }
    for(i=0;i<9;i++) Solution[i] = dTemp[i];
}

void JCMatrix::Homlnv(double Mat[16], double lnvMat[16])
{
    // invert the homogeneous transformation matrix
    lnvMat[0] = Mat[0];
    lnvMat[1] = Mat[1];
    lnvMat[3] = 0;
    lnvMat[7] = 0;
    lnvMat[8] = Mat[8];
    lnvMat[9] = Mat[9];
    lnvMat[10] = Mat[10];
    lnvMat[11] = 0;
    lnvMat[12] = 0;
    lnvMat[13] = 0;
    lnvMat[14] = 0;
    lnvMat[15] = 1;
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            lnvMat[i*3+j] = lnvMat[i*3+j] + ((-Mat[j*4+i])*Mat[j*4+3]);
        }
    }
}


```c
void JCMatrix::Trans(double Vector[6], int Flag1, int Flag2, double Solution[16])
{
    // create a transformation matrix from the vector
    // the angles must be in radians

    // Flag1 = 1 pre-multiply the matrices to rotate about fixed axes
    // Flag1 = 2 post-multiply the matrices to rotate about rotating axes

    // Flag2 = 1 configure axes in OAT style. Rotate about ZYZ axes
    // Flag2 = 2 configure axes in XYZ style. Rotate about XYZ axes

    double R1[9], R2[9], R3[9], Temp[9], Temp2[9];

double cx = cos(Vector[3]);
double sx = sin(Vector[3]);
double cy = cos(Vector[4]);
double sy = sin(Vector[4]);
double cz = cos(Vector[5]);
double sz = sin(Vector[5]);

switch(Flag1)
{
    case 1:
        break;
    case 2:
        break;
}

switch(Flag2)
{
    case 1:
        R1[0] = cx; R1[1] = -sx; R1[2] = 0.0;
        R1[6] = 0.0; R1[7] = 0.0; R1[8] = 1.0;

        R2[0] = cy; R2[1] = 0.0; R2[2] = sy;
        R2[3] = 0.0; R2[4] = 1.0; R2[5] = 0.0;
        R2[6] = -sy; R2[7] = 0.0; R2[8] = cy;

        R3[0] = cz; R3[1] = -sz; R3[2] = 0.0;
        R3[6] = 0.0; R3[7] = 0.0; R3[8] = 1.0;
        break;

    case 2:
        R1[0] = 1.0; R1[1] = 0.0; R1[2] = 0.0;
        R1[6] = 0.0; R1[7] = sx; R1[8] = cx;

        R2[0] = cy; R2[1] = 0.0; R2[2] = sy;
        R2[3] = 0.0; R2[4] = 1.0; R2[5] = 0.0;
        R2[6] = -sy; R2[7] = 0.0; R2[8] = cy;

        R3[0] = cz; R3[1] = -sz; R3[2] = 0.0;
        R3[6] = 0.0; R3[7] = 0.0; R3[8] = 1.0;
        break;
}

switch(Flag2)
{
    case 1:
        // pre-multiplication for fixed axis rotations
        MatXMat3(R3,R2,Temp);
        MatXMat3(Temp,R1,Temp2);
        break;

    case 2:
        // post-multiplication for fixed axis rotations
        break;
}
```

MatXM3(R1, R2, Temp);
MatXM3(Temp, R3, Temp2);

// place the 3x3 matrix within the 4x4 transformation
for (int i = 0; i < 3; i++)
{
    Solution[i*4 + 3] = Vector[i];
    Solution[i + 12] = 0.0;
    for (int j = 0; j < 3; j++)
    {
        Solution[i*4 + j] = Temp2[i*3 + j];
    }
}
Solution[15] = 1.0;

void JCMatrix::Decompose(double Transformation[16], double Solution[6])
{
    // create a vector from the transformation matrix
    // this solves for rotations about fixed axes in order XYZ
    Solution[0] = Transformation[3];
    Solution[1] = Transformation[7];
    Solution[3] = atan2(Transformation[9], Transformation[10]);
    Solution[5] = atan2(Transformation[4], Transformation[0]);
    double Temp = Transformation[9]*sin(Solution[3]) + Transformation[10]*cos(Solution[3]);
    Solution[4] = atan2(-Transformation[8], Temp);
}

void JCMatrix::DecomposePUMA(double Transformation[16], double Solution[6])
{
    // create a vector from the transformation matrix
    // this solves for rotations about fixed axes in order ZYZ
    // using code originally supplied by Marcus Pie of Unimation
    for (int i = 0; i < 6; i++) Solution[i] = 0;
    double Temp = sqrt((Transformation[2]*Transformation[2]) + (Transformation[6]*Transformation[6]));
    Solution[4] = atan2(-Transformation[10], Temp);
    if (Temp < 0.001)
    {
        Solution[3] = atan2(Transformation[5], Transformation[1]);
        Solution[5] = 0;
    }
    else
    {
        Solution[3] = atan2(Transformation[2], -Transformation[6]);
        Solution[5] = atan2(Transformation[9], -Transformation[8]);
    }
    Solution[0] = Transformation[3];
    Solution[1] = Transformation[7];
    // convert to true Euler angles
}
C.1.11. JCPuma.h

// FILE JCPuma.h

#ifndef JCPUMA
#define JCPUMA

class JCPuma
{
public:
    JCPuma();
    ~JCPuma();

    // Attributes
    public:

    private:

    // private member variables
    unsigned char m_cDEL;  // bytestuffing control bytes
    unsigned char m_cDLE;  // bytestuffing control bytes
    unsigned char m_cSTX;  // bytestuffing control bytes
    unsigned char m_cETX;  // bytestuffing control bytes
    unsigned char m_cNullMessage[12];

    unsigned int m_nInterruptNumber;    // Combios interrupt number
    unsigned int m_nInterruptChannel;   // Combios interrupt number

    unsigned int m_nControlByte;        // PUMA status information bytes
    unsigned int m_nAlterMode;          // PUMA status information bytes
    unsigned int m_nSegmentState;       // PUMA status information bytes
    unsigned int m_nSegmentNumber;      // PUMA status information bytes
    unsigned int m_nMotionPercent;      // PUMA status information bytes
    unsigned int m_nAlterStarted;       // PUMA status information bytes
    unsigned int m_nErrorCode;

    double m_dStartTransform[16];

    // private member functions
    void PumaGetByte(unsigned char *Byt);
    void PumaSendByte(unsigned char *Byt);
    void PumaGetTransformation(double StartTransform[16], int *marker);
    unsigned int PerformInterrupt(unsigned int i, unsigned int a, unsigned int b, unsigned int c);

    // Operations
    public:

    // public member functions
    void PumaSendInitialMessage(void);
    void PumaSendMessage(unsigned char Message[12], int *ExceptionCode);

    int PumaGetInitialMessage();
    void PumaGetMessage(int *AlterStopping);
    void PumaFakeMessage(int *AlterStopping);

    void PumaGetStartPosition(double StartTransform[16]);

    void PumaSetBytes(unsigned char Message[12], unsigned char NonZeroMessage[12],
                      unsigned char *SelectBits, unsigned char *NumSelectBits);

    213
private:
};

#endif

C.1.12. JCPuma.cpp

// File: JCPuma.cpp
// File Name: JCPuma.cpp
// Function Name: JCPuma
// Purpose: Defines the PUMA serial comms class. This enables the PUMA controller to be contacted and controlled using COMBIOS serial comms interrupts
// Author: J. Tisdall
// Written on: 19/4/96

// INCLUDE FILES
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <dos.h> // For Turbo C include DOS.H for int86() and REGS.
#include "JCPuma.h"
#include "all def.h"
extern FILE *test;

// CONSTRUCTOR
JCPuma::JCPuma()
{
    m_nInterruptNumber = INTERRUPT_NUMBER; // Combios interrupt number
    m_nInterruptChannel = INTERRUPT_CHANNEL; // Combios interrupt channel

    m_cDEL = DEL; // byte stuffing control bytes
    m_cDLE = DLE; // byte stuffing control bytes
    m_cSTX = STX; // byte stuffing control bytes
    m_cETX = ETX; // byte stuffing control bytes

    // PUMA status information bytes
    m_nControlByte = ZERO;
    m_nAlterMode = ZERO;
    m_nSegmentState = ZERO;
    m_nSegmentNumber = ZERO;
    m_nMotionPercent = ZERO;
    m_nAlterStarted = ZERO;
    m_nErrorCode = ZERO;

    for(int i=0;i<12;i++) m_cNullMessage[i] = ZERO;

    // set up combios to read COM2
    // Check Bios exists BX=22, CX=33; Result in AX = BX+CX = 55
    if(PerformInterrupt(m_nInterruptNumber,FN_EXISTS,22,33) != 55)
        214
cout << "COMBIOS NOT FOUND AT INTERRUPT" << m_nInterruptNumber << "ABORTING....." << endl;
exit(0);

// Check number of active channels
int maxchnIs=PerformInterrupt(m_nInterruptNumber,FN_PORTNO,0,0);

// Start Hardware interrupt, if error detected say so
if (PerformInterrupt(m_nInterruptNumber,FN_START,0,0) != 0)
printf("nSTART INTERRUPT ERROR\n");

// DESTRUCTOR -----------------------------------------------------------
JCPuma::~JCPuma()
{
// Stop serial interrupt
if(PerformInterrupt(m_nInterruptNumber,FN_STOP,ZERO,ZERO) !=ZERO)
cout << "STOP INTERRUPT ERROR!" << endl:
}

// MEMBER FUNCTIONS ------------------------------------------------------
void JCPuma::PumaGetStartPosition(double StartTransform[16])
{
  // Purpose: Reads initial PUMA position from PUMA
  int j = 0;
  int k = 0;
  int ExceptionCode = -1;
  int marker = 0;  // 1 = alter stopping
double Transform[16];
  unsigned char NonZeroMessage[12];
  unsigned char SelectBits;
  unsigned char NumSelectBits;

  while (!PumaGetInitialMessage())&&(_kbhit()==ZERO));// Waits for ALTER STARTING
if (!(_kbhit()==0)) fprintf(test,"PumaGetStartPosition - I keyboard was hit\n");
PumaSendlnitialMessage(); // send initial message to PUMA

// receive PUMA message
PumaGetTransformation(m_dStartTransform, &marker);
for(int i=0;i<16;i++) StartTransform[i] = m_dStartTransform[i];
fprintf(test,"nStartTransform = \n");
for(j=0;j<4;j++)
{
  for(i=0;i<4;i++) fprintf(test,"%7.2f \n",StartTransform[j*4+i]);
  fprintf(test,"\n");
}

//cout << "Got transformation in PumaGetStartPosition" << endl;

// send PUMA exception message to end ext alter
ExceptionCode = -1;
PumaSendMessage(m_cNullMessage, &ExceptionCode);
//cout << "Sent exception message in PumaGetStartPosition" << endl;

marker = 0;
i = 0,
do  // read from PUMA until alter stopping received
{

    // Calculate SelectBits and NumSelectBits
    PumaSetBytes(m_cNullMessage, NonZeroMessage, &SelectBits, &NumSelectBits);
    // cout << "SetB>les in PumaGetStartPosition" « endl;
    PumaGetTransformation(Transform, &marker);
    fprintf(test,"\n");
    // cout << "Got transformation in PumaGetStartPosition" << endl;
    PumaSendMessage(NonZeroMessage, &ExceptionCode);
    if(marker==1) i = 1;
    // cout << "marker = " « marker « endl;
} while((i==0)&&(\_kbhit()==ZERO));
if (!(_kbhit()==0)) fprintf(test,"\nPumaGetStartPosition - 2 keyboard was hit\n");
// cout << "Read until alter stopping received in PumaGetStartPosition" « endl;

void JCPuma::PumaSetBytes(unsigned char Message[12], unsigned char NonZeroMessage[12],
    unsigned char * SelectBits, unsigned char * NumSelectBits)
{
// ensures that only non-zero bytes are transmitted to PUMA

    int CountLocal;
    unsigned char SetPoints = 0x0;

    int CountNumBits = 0x0;
    for(int i=0;i<6;i++)
    {
        int Count_A = 2*i;
        int Count_B = 2*i+1;

        // if either message part is non-zero
        if(!(!(Message[Count_A]==0))||(!(Message[Count_B]==0)))
        {
            CountLocal = 2*CountNumBits;
            CountNumBits = CountNumBits + 1;
            NonZeroMessage[CountLocal] = Message[Count_A];
            NonZeroMessage[CountLocal+1] = Message[Count_B];
            SetPoints = SetPoints|(0xl<<(5-i));
        }
    }

    * SelectBits = SetPoints;
    * NumSelectBits = CountNumBits;

}

int JCPuma::PumaGetInitialMessage()
{
    // Purpose: Reads initial byte stuffing message from PUMA
    // Returns true if received all starting without error else false
    unsigned char CtrlByte;
    unsigned char AltMode;
    unsigned char Temp;
    unsigned char MyChecksum;
    unsigned char MyTwosComp;
    unsigned char check = START_CHECK;

    // get bytes up to m_cSTX
    while(!(!(Temp == m_cSTX))&&(_kbhit()==ZERO)) PumaGetByte(&Temp);
    if(!(_kbhit()==0)) fprintf(test,"\nPumaGetInitialMessage - 1 keyboard was hit\n");

    PumaGetByte(&CtrlByte);
    if(CtrlByte==m_cDLE) PumaGetByte(&CtrlByte); // repeat get if it is DLE
    check = check + CtrlByte; // add to checksum

m_nControlByte = (unsigned)CtrlByte; // change type
PumaGetByte(&AltMode); // get alter mode
if(AltMode==m_cDLE) PumaGetByte(&AltMode); // repeat get if it is DLE
check = check + AltMode; // add to checksum
m_nAlterMode = (unsigned)AltMode; // change type
MyChecksum = check; // calculate two's complement of checksum
check=check+1;
while(check>=0x100) check=check-0x100;
while(check<0) check=check+0x100;
MyTwosComp = check;

// get bytes up to m_cETX
while(!(Temp==m_cETX))&&(_kbhit()==ZERO) PumaGetByte(&Temp);
if (!(_kbhit()==0)) fprintf(test, "PumaGetInitialMessage - 2 keyboard was hit\n");
PumaGetByte(&Temp); // get PUMA checksum
if (((unsigned)Temp-check) != ZERO) // check for error in message
{
    fprintf(test,"Checksum error in initial message (from PumaGetInitMessage)\n");
}
m_nAlterStarted = m_nControlByte&MASK_VALUE;
m_nErrorCode = m_nControlByte&ERR_MASK_VALUE;
if ((m_nAlterStarted==0x20)&&(m_nErrorCode==0x0))
{
    return 1;
}
else
{
    return 0;
}

void JCPuma::PumaGetMessage(int *AlterStopping)
{
    // Purpose: Reads regular byte stuffing message from PUMA
    unsigned char CtrlByte;
    unsigned char Temp;
    unsigned char SegmentNumberLow, SegmentNumberHigh;
    unsigned char MotionPercentLow, MotionPercentHigh;
    unsigned char check = START_CHECK;
    unsigned char AltStatus;

    // get bytes up to STX
    while(!(Temp==m_cSTX))&&(_kbhit()==ZERO) PumaGetByte(&Temp);
    if (!(_kbhit()==0)) fprintf(test, "PumaGetMessage - 1 keyboard was hit\n");
PumaGetByte(&CtrlByte); // get control byte
if(CtrlByte==m_cDLE) PumaGetByte(&CtrlByte); // repeat get if it is DLE
check = check + CtrlByte; // add to checksum
m_nControlByte = (unsigned)CtrlByte; // change type
AltStatus = CtrlByte & MASK_VALUE;
if((AltStatus==0x00)||AltStatus==0x20)||AltStatus==0x40)) // alter starting or alter running
{
    PumaGetByte(&Temp);
    if(Temp==m_cDLE) PumaGetByte(&Temp);
    m_nSegmentState=(unsigned)Temp,
}

217
check = check + Temp;

PumaGetByte(&SegmentNumberLow);  // get the segment number low byte
if(SegmentNumberLow==m_cDLE) PumaGetByte(&Temp);// repeat get if it is DLE
check = check + SegmentNumberLow;
PumaGetByte(&SegmentNumberHigh); // get the segment number high byte
if(SegmentNumberHigh==m_cDLE) PumaGetByte(&Temp); // repeat get if it is DLE
check = check + SegmentNumberHigh;
m_nSegmentNumber = (int)SegmentNumberHigh*0x100 +
(int)SegmentNumberLow;  // combine

PumaGetByte(&MotionPercentLow); // get the motion percent low byte
if(MotionPercentLow==m_cDLE) PumaGetByte(&Temp); // repeat get if it is DLE
check = check + MotionPercentLow;
PumaGetByte(&MotionPercentHigh); // get the motion percent high byte
if(MotionPercentHigh==m_cDLE) PumaGetByte(&Temp); // repeat get if it is DLE
check = check + MotionPercentHigh;
m_nMotionPercent = (int)MotionPercentHigh*0x100 + (int)MotionPercentLow;

*AlterStopping = 0;
}
else
// alter pausing or stopping
{
    fprintf(test, "alter stopping\n");  // alter stopping
    *AlterStopping = 1;
}
check = check + 1;

// get bytes up to m_cETX
while(!((Temp==m_cETX)&&(kbit()==ZERO))) PumaGetByte(&Temp);
if (!(_kbhit()==0)) fprintf(test, "in PumaGetMessage - 2 keyboard was hit\n");
PumaGetByte(&Temp);  // PumaGet checksum

while(check>=0x100)
{
    check=check-0x100;
}
while(check<0)
{
    check=check+0x100;
}
if (((unsigned)Temp-check) != ZERO)
{
    fprintf(test, "Checksum error in message during PumaGetMessage\n");
}

}

void JCPuma::PumaFakeMessage(int *AlterStopping)
{
    // Purpose: tests message transfer
    *AlterStopping = 1;
}

void JCPuma::PumaSendInitialMessage(void)
{
    // Purpose: Sends initial byte stuffing message to PUMA
    unsigned char check = START_CHECK;
    unsigned char data = ZERO;  // data, only one control byte
void JCPuma::PumaSendMessage(unsigned char Message[12], int *ExceptionCode)
{
    // Purpose: Sends regular byte stuffing message to PUMA
    unsigned char check = START_CHECK;  // data, only one control byte
    unsigned char data = ZERO;  // data, only one control byte
    unsigned char tempi, SelectBits, NumSelectBits;
    unsigned char NonZeroMessage[12];
    int i;

    PumaSetBytes(Message, NonZeroMessage, &SelectBits, &NumSelectBits);
    tempi = (unsigned char)*ExceptionCode;
    NumSelectBits = NumSelectBits*0x2;

    PumaSendByte(&m_cDEL);  
PumaSendByte(&m_cDLE);   
PumaSendByte(&m_cSTX);  
PumaSendByte(&tempi);   
check = check + tempi;   
PumaSendByte(&SelectBits); 
check = check + SelectBits;

    // This sends the required number of bytes
    // it is assumed that the bytes have been put in the
    // right order in subroutine control.c
    for (i=0; i<NumSelectBits; i++) // *NumSelectBits bytes to be sent
    {
        tempi = NonZeroMessage[i];
        PumaSendByte(&tempi); 
        if (tempi==m_cDLE) PumaSendByte(&tempi);  
        check = check + tempi;
    }

    PumaSendByte(&m_cDEL);  
PumaSendByte(&m_cETX);   
check=~check+1;         
while (check >= 0x100)  
{
    check=check-0x100; 
}

    while(check<0)
    {
        check=check+0x100; 
    }

    PumaSendByte(&check);
}

void JCPuma::PumaGetTransformation(double StartTransform[16], int *marker)
{
    // Purpose: Reads a transformation from PUMA by sending an exception code

    // Implementation details...
}
unsigned char CtrlByte;
unsigned char Temp, Temp1, Temp2;
unsigned char check = START_CHECK;
unsigned char AltStatus;
unsigned char SegmentNumberLow, SegmentNumberHigh;
unsigned char MotionPercentLow, MotionPercentHigh;

int LocalTransform[12];
int i,j,k;

// get bytes up to STX
while(!(Temp==m_cSTX)&&( !_kbhit())) PumaGetByte(&Temp); 
if (!(_kbhit()==0)) fprintf(test,"PumaGetTransformation - 1 keyboard was hit\n");
PumaGetByte(&CtrlByte); // get control byte
if(CtrlByte==m_cDLE) PumaGetByte(&CtrlByte);
check = check + CtrlByte;
AltStatus = check & MASK_VALUE;
if((AltStatus==0x00)||(AltStatus==0x20)) // alter starting or alter running
{
    PumaGetByte(&Temp); // get the segment state
    if(Temp==m_cDLE) PumaGetByte(&Temp);
    m_nSegmentState=(unsigned)Temp;
    check = check + Temp;
    PumaGetByte(&SegmentNumberLow); // get the segment number low byte
    if(SegmentNumberLow==m_cDLE) PumaGetByte(&Temp);
    check = check + SegmentNumberLow;
    PumaGetByte(&SegmentNumberHigh); // get the segment number high byte
    if(SegmentNumberHigh==m_cDLE) PumaGetByte(&Temp);
    check = check + SegmentNumberHigh;
    m_nSegmentNumber = (int)SegmentNumberHigh*Oxl00 + (int)SegmentNumberLow;
    PumaGetByte(&MotionPercentLow); // get the motion percent low byte
    check = check + MotionPercentLow;
    PumaGetByte(&MotionPercentHigh); // get the motion percent high byte
    if(MotionPercentHigh==m_cDLE) PumaGetByte(&Temp);
    check = check + MotionPercentHigh;
    m_nMotionPercent = (int)MotionPercentHigh*Ox100 + (int)MotionPercentLow;
    for(i=0;i<12;i++)
    {
        PumaGetByte( &Temp1);
        if(Temp1==m_cDLE) PumaGetByte(&Temp1);
        check = check+Temp1;
        PumaGetByte( &Temp2);
        if(Temp2==m_cDLE) PumaGetByte(&Temp2);
        check = check+Temp2;
        LocalTransform[i] = (int)Temp1 + (int)Temp2*Oxl00;
    }
    *marker = 0;
}
else
// alter pausing or stopping
{
    fprintf(test,"\nalter stopping\n"); // alter stopping
}
*marker = 1;
}
check=-check+1;

// get bytes up to ETX
while((!(Temp==m_cETX))&&!(_kbhit()==0)) PumaGetByte(&Temp);
if (!(_kbhit()==0)) fprintf(test, "nPumaGetTransformation - 2 keyboard was hit\n");
PumaGetByte(&Temp); // PumaGet checksum
while(check>=0x100)
{
    check=check-0x100;
}
while(check<ZERO)
{
    check=check+0x100;
}
if (((unsigned)Temp-check) != ZERO)
{
    fprintf(test,"Checksum error in message during PumaGetMessage!\n");
}
for(j=0;j<3;j++) // scale PUMA transformation matrix
{
    for(k=0;k<3;k++)
    {
        StartTransform[j*4+k] = (double)LocalTransform[k*3+j]/16384;
    }
    StartTransform[j*4+3] = (double)LocalTransform[j+9]/32;
    StartTransform[j+12] = 0.0;
}
StartTransform[15] = 1.0;

void JCPuma::PumaGetByte(unsigned char *Byt)
{
    // Purpose: Reads single byte from PUMA
    while
    {
        while(PerformInterrupt(m_nInterruptNumber,FN_RXREADY,m_nInterruptChannel,ZERO)<=
        (ZERO)&&(!(_kbhit()==ZERO))); // wait until character received
        if(_kbhit()==ZERO)
        {
            unsigned char ch = PerformInterrupt(m_nInterruptNumber,FN_RXCHAR,
            m_nInterruptChannel,ZERO); // read it
            *Byt = ch;
        }
        else
        {
            if (!(_kbhit()==0)) fprintf(test,"nPumaGetByte - keyboard was hit\n");
        }
    }
}

void JCPuma::PumaSendByte(unsigned char *Byt)
{
    // Purpose: Sends single byte to PUMA
    unsigned char ch = *Byt;
    while(PerformInterrupt(m_nInterruptNumber,FN_TXREADY,
    m_nInterruptChannel,ZERO)!=ZERO); // Wait until transmitter ready
if(PerformInterrupt(m_nInterruptNumber,FN_TXCHAR,m_nInterruptChannel,ch)!=ZERO)
    // then send character
    printf("\aTRANSMIT CHAR ERROR\n");

unsigned int JCpuma::PerformInterrupt(unsigned int i, unsigned int a, unsigned int b, unsigned int c)
{
    // Purpose: Sets up the interrupt handler to enable byte stuffing
    union REGS regs;       // REGS defined in dos.h
    regs.x.ax=a;
    regs.x.bx=b;           // Load registers
    regs.x.cx=c;
    _int86(i,&regs,&regs);   // Do interrupt
    return(regs.x.ax);     // Return result from AX
}
As output, each camera gives a voltage related to the \((x,y)\) co-ordinate of the target image as it falls on the image plane. Figure D.1 shows a simplified schematic of this assuming a pin-hole camera.

In order to be able to find the focal length, \(\lambda\), it was necessary to gather data at a number of known distances from the camera, \(L\). This was done by mounting a twenty point target of known geometry on the PUMA arm and stepping it through space along the \(Z\)-axis, recording the camera output at each step.

It was then possible to do a least squares minimisation on the data sets and calculate the image plane position, \(u\), and thus calculate \(a\), the distance from the front of the camera to the apparent focal point of the lens. It was therefore possible to convert the camera voltages recorded into a direction vector \((x,y,\lambda)\) emanating from the focal point of the camera, a point that was geometrically known relative to the camera body.

From figure K1 we can see that, by equal triangles, the following is true:

\[
x(\lambda - Z) = \lambda X
\]  

(1)
For each of the twenty target positions, the actual \((x,z)\) co-ordinate of the \(n^{th}\) target from the centre is given by:
\[
(x_n, y_n) \quad \text{i.e. } (x_1, y_1) = (120, 75)
\]

The co-ordinate from the centre of the camera is affected by the offset rotation of the target relative to the camera by angle \(\Phi - \phi\).
\[
X_n = x_n \cos \phi - y_n \sin \phi \quad (2)
\]
\[
Y_n = x_n \sin \phi + y_n \cos \phi \quad (3)
\]

Now, a simple first order fit for the voltage to the image plane displacement for each camera is given by:
\[
V_{x_n} = AX_n + B \quad (4)
\]
\[
V_{y_n} = CY_n + D \quad (5)
\]

where \(A, B, C, D\) are constants. Hence, the minimisation below enables us to solve for \(A, B, C, D, \phi\):
\[
\text{sum square error} = \sum \text{error}^2 = \sum (f_n - V_n)^2 \quad (6)
\]

let us call this function \(G\):
\[
G = \sum (fx_n - Vx_n)^2 + \sum (fy_n - Vy_n)^2 \quad (7)
\]
\[
G = \sum (fx_n - AX_n - B)^2 + \sum (fy_n - CY_n - D)^2 \quad (8)
\]
\[
G = \sum (fx_n - Ax_n \cos \phi + Ay_n \sin \phi - B)^2 + \sum (fy_n - Cx_n \sin \phi - Cy_n \cos \phi - D)^2 \quad (9)
\]

where \(fx_n, fy_n\) = the measured voltage

By differentiating partially with respect to \(A, B, C, D\) and \(\phi\) and setting the resultant equation to zero we can simply solve for \(A, B, C, D\) and \(\phi\) by standard matrix methods.
### APPENDIX E

**RESOLVED APPLIED LOADS TO THE DOCKED ROV**

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Fx (N)</th>
<th>Fy (N)</th>
<th>Fz (N)</th>
<th>Tx (Nm)</th>
<th>Ty (Nm)</th>
<th>Tz (Nm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>412</td>
<td>0</td>
<td>164</td>
<td>169</td>
<td>-575</td>
<td>-420</td>
</tr>
<tr>
<td>17</td>
<td>824</td>
<td>0</td>
<td>328</td>
<td>337</td>
<td>-1151</td>
<td>-841</td>
</tr>
<tr>
<td>18</td>
<td>1236</td>
<td>0</td>
<td>492</td>
<td>505</td>
<td>-1727</td>
<td>-1262</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>-363</td>
<td>253</td>
<td>-549</td>
<td>-372</td>
<td>532</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>-726</td>
<td>505</td>
<td>-1098</td>
<td>-744</td>
<td>1064</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>-1089</td>
<td>758</td>
<td>-1647</td>
<td>-1116</td>
<td>1595</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>-342</td>
<td>239</td>
<td>-518</td>
<td>-351</td>
<td>502</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
<td>-685</td>
<td>477</td>
<td>-1036</td>
<td>-702</td>
<td>1003</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
<td>-1027</td>
<td>715</td>
<td>-1554</td>
<td>-1053</td>
<td>1505</td>
</tr>
<tr>
<td>44</td>
<td>412</td>
<td>0</td>
<td>164</td>
<td>169</td>
<td>-575</td>
<td>-420</td>
</tr>
<tr>
<td>45</td>
<td>824</td>
<td>0</td>
<td>328</td>
<td>337</td>
<td>-1151</td>
<td>-841</td>
</tr>
<tr>
<td>46</td>
<td>1236</td>
<td>0</td>
<td>492</td>
<td>505</td>
<td>-1727</td>
<td>-1262</td>
</tr>
</tbody>
</table>

**TABLE E.1 - MAXIMUM IMPULSIVE LOAD APPLIED DURING IMPULSE TESTS**

<table>
<thead>
<tr>
<th>Test</th>
<th>Fx (N)</th>
<th>Fy (N)</th>
<th>Fz (N)</th>
<th>Tx (Nm)</th>
<th>Ty (Nm)</th>
<th>Tz (Nm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>92</td>
<td>0</td>
<td>37</td>
<td>38</td>
<td>-127</td>
<td>-93</td>
</tr>
<tr>
<td>17</td>
<td>183</td>
<td>0</td>
<td>73</td>
<td>75</td>
<td>-255</td>
<td>-186</td>
</tr>
<tr>
<td>18</td>
<td>274</td>
<td>0</td>
<td>109</td>
<td>112</td>
<td>-382</td>
<td>-279</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>-80</td>
<td>56</td>
<td>-121</td>
<td>-82</td>
<td>118</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>-160</td>
<td>112</td>
<td>-243</td>
<td>-164</td>
<td>236</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>-241</td>
<td>168</td>
<td>-364</td>
<td>-247</td>
<td>354</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>-80</td>
<td>56</td>
<td>-121</td>
<td>-82</td>
<td>118</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
<td>-160</td>
<td>112</td>
<td>-243</td>
<td>-164</td>
<td>236</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
<td>-241</td>
<td>168</td>
<td>-364</td>
<td>-247</td>
<td>354</td>
</tr>
<tr>
<td>44</td>
<td>92</td>
<td>0</td>
<td>37</td>
<td>38</td>
<td>-127</td>
<td>-93</td>
</tr>
<tr>
<td>45</td>
<td>183</td>
<td>0</td>
<td>73</td>
<td>75</td>
<td>-255</td>
<td>-186</td>
</tr>
<tr>
<td>46</td>
<td>274</td>
<td>0</td>
<td>109</td>
<td>112</td>
<td>-382</td>
<td>-279</td>
</tr>
</tbody>
</table>

**TABLE E.2 - FINAL STATIC LOAD APPLIED DURING IMPULSE TESTS**
<table>
<thead>
<tr>
<th>test</th>
<th>$F_x$ (N)</th>
<th>$F_y$ (N)</th>
<th>$F_z$ (N)</th>
<th>$T_x$ (Nm)</th>
<th>$T_y$ (Nm)</th>
<th>$T_z$ (Nm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>-160</td>
<td>112</td>
<td>-243</td>
<td>-164</td>
<td>236</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-241</td>
<td>168</td>
<td>-364</td>
<td>-247</td>
<td>354</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-160</td>
<td>112</td>
<td>-243</td>
<td>-164</td>
<td>236</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>-80</td>
<td>56</td>
<td>-121</td>
<td>-82</td>
<td>118</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>92</td>
<td>0</td>
<td>37</td>
<td>38</td>
<td>-127</td>
<td>-93</td>
</tr>
<tr>
<td>11</td>
<td>183</td>
<td>0</td>
<td>73</td>
<td>75</td>
<td>-255</td>
<td>-186</td>
</tr>
<tr>
<td>12</td>
<td>274</td>
<td>0</td>
<td>109</td>
<td>112</td>
<td>-382</td>
<td>-279</td>
</tr>
<tr>
<td>13</td>
<td>183</td>
<td>0</td>
<td>73</td>
<td>75</td>
<td>-255</td>
<td>-186</td>
</tr>
<tr>
<td>14</td>
<td>92</td>
<td>0</td>
<td>37</td>
<td>38</td>
<td>-127</td>
<td>-93</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>-80</td>
<td>56</td>
<td>-121</td>
<td>-82</td>
<td>118</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>-160</td>
<td>112</td>
<td>-243</td>
<td>-164</td>
<td>236</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>-241</td>
<td>168</td>
<td>-364</td>
<td>-247</td>
<td>354</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
<td>-160</td>
<td>112</td>
<td>-243</td>
<td>-164</td>
<td>236</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>-80</td>
<td>56</td>
<td>-121</td>
<td>-82</td>
<td>118</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>37</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>38</td>
<td>92</td>
<td>0</td>
<td>37</td>
<td>38</td>
<td>-127</td>
<td>-93</td>
</tr>
<tr>
<td>39</td>
<td>183</td>
<td>0</td>
<td>73</td>
<td>75</td>
<td>-255</td>
<td>-186</td>
</tr>
<tr>
<td>40</td>
<td>274</td>
<td>0</td>
<td>109</td>
<td>112</td>
<td>-382</td>
<td>-279</td>
</tr>
<tr>
<td>41</td>
<td>183</td>
<td>0</td>
<td>73</td>
<td>75</td>
<td>-255</td>
<td>-186</td>
</tr>
<tr>
<td>42</td>
<td>92</td>
<td>0</td>
<td>37</td>
<td>38</td>
<td>-127</td>
<td>-93</td>
</tr>
<tr>
<td>43</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**TABLE E.3 - LOADS APPLIED DURING STATIC TESTS**
Listing of ROV motion simulation program written in Matlab V 4.0

% Impulse Response of Mass-Spring-Damper ROV model
% SYSTEM VARIABLE DEFINITIONS & INITIAL VALUES

% Fk Force Input Function
% Xk, Xk1, Xk2 System Displacement Outputs
Fk = 0.0;
Xk = 0.0;
Xk1 = 0.0;
Xk2 = 0.0;

% SYSTEM PARAMETER DEFAULT VALUES
% K Linear Stiffness Coefficient [N/m]
% f Coeff of Viscous Friction [N/m/s]
% m Mass [kg]

% Assume initial Period of motion to be 3.5 secs
% Therefore frequency = 0.286 Hz or 1.8 rad/s.
% Mass = 1.5 tonnes, hence K = \( w^2 m = 4860 \) N/m.
% Also assume damping ratio, \( \zeta = 0.5 \)
% i.e \( f = 2700 \) Ns/m.

m = 1500;
k = 4860;
f = 2700;

% SIMULATION PARAMETER VALUES
% T Update Time Interval (s)
% n Number of iteration steps
T = 0.01;
n = 1000;

% DEFINE FORCE IMPULSE INPUT FUNCTION
% Based on Linearly ramping to 1000N in 0.1 sec and
% ramping back down to zero in a further 0.1 sec
% ie Fk is incremented by 100 each sample until 1000
% and then decremented by 100 until zero.
% NB This is main area of doubt in system model!!

% CONSTRUCT EQUATIONS AND UPDATE IN FOR LOOP
for i = 1:n,
    Fklog(i) = Fk;
    Xklog(i) = Xk;
    Tlog(i) = T+(i-1);

    % Eqn is:- mD^2X + fDX + kX = F
    % Where:
    % DX = (Xk1-Xk)/T and D^2X = (DX1-DX)/T
    if i<41
        Fk = Fk + 100;
    end
    if i>40
        if i<81
            Fk = Fk - 100;
        end
    end
    if i>80
        Fk = 200;
    end

    Xk2 = Fk*T*T/m + (2-f*T/m)*Xk1 + (f*T/m-k*T*T/m-1)*Xk;

% UPDATE VARIABLES AND REPEAT ITERATION
    Xk = Xk1;
    Xk1 = Xk2;
end
The chosen force/torque sensor for this thesis work was the six degrees of freedom Mini 40/2 manufactured by Associated Technologies Incorporated (ATI) of 503D Highway 70, East Garner, North Carolina 27528, USA.

The transducer uses silicon strain gauges that convert the force and torque to analogue electrical signals that are decoupled to give a digital (parallel or serial) or analogue signal proportional to the force or torque at each degree of freedom.

The transducer is rated for up to 40 N force and 2 Nm torque and is rated to survive temporary overloads of ten times that without permanent damage. However, the transducer uses a range of gauges that are shared amongst different degrees of freedom. This means that, if forces are low, higher torques than the device is rated for may be read. Similarly, if torques are low, higher forces than the device is rated for may be read. This may be quantified by these limits which may not be exceeded in normal operation:

\[
\left( \frac{F_{xy}^2 + T_{z}}{80} \right) + \left( \frac{T_{xy}}{220} \right) \leq 1 \]

where:

\[ F_{xy} = \sqrt{F_x^2 + F_y^2} \]

and:

\[ T_{xy} = \sqrt{T_x^2 + T_y^2} \]

The transducer measures 3.5 mm in thickness and normally resides between two interface plates that also act as limit stops and protective covers. This takes the overall dimensions to 12 x 44.7 x 31.75 mm.
H.1. Without Probe Offset

For the first computer model, it was assumed that the probe was in its neutral position with no initial rotation about the $x$ axis. The results tabled in chapter seven came from:

H.1.1. Program main3.m

```matlab
% calls function var_mod3 to find F/T vector from input vector [x y z Rx Ry Rz]

% Define the input position as six separate columns, each column has only one degree of freedom varying
inpos = [0.0005 0 0 0 0 0
0 0.0005 0 0 0 0
0 0 0.0005 0 0 0
0 0 0 0.02 0 0
0 0 0 0 0.02 0
0 0 0 0 0 0.02];

% Initialise solution matrix to zero
allval = zeros(8,16);

% Define the centre of rotation relative to the centre of force
Odash = [0 0 -0.04 0];

% Define central and side spring end locations
A = [0 -0.01 -0.01 1];
B = [0 0.01 -0.01 1];
C = [0 0.01 0.01 1];
D = [0 -0.01 0.01 1];
A1 = [0.032 -0.01 -0.01 1];
B1 = [0.032 0.01 -0.01 1];
C1 = [0.032 0.01 0.01 1];
D1 = [0.032 -0.01 0.01 1];

% Define the magnitude and range of variation of spring co-ordinates
% as the geometry of the compliant device is varied.
delta = 0.0025;
delly = [0 delta 0 0];
dellz = [0 0 delta 0];
pos1 = delly + dellz;
pos2 = delly - dellz;
pos3 = -delly + dellz;
pos4 = -delly - dellz;
```
% Loop through the various geometries
for pattern_a = 1:4
    % Re-define central spring end locations
    A = [0 -0.01 -0.01 1];
    B = [0 0.01 -0.01 1];
    C = [0 0.01 0.01 1];
    D = [0 -0.01 0.01 1];
    if pattern_a == 1
        A = A + pos4;
        B = B + pos2;
        C = C + pos1;
        D = D + pos3;
    elseif pattern_a == 2
        A = A + pos3;
        B = B + pos1;
        C = C + pos2;
        D = D + pos4;
    elseif pattern_a == 3
        A = A + pos2;
        B = B + pos4;
        C = C + pos3;
        D = D + pos1;
    elseif pattern_a == 4
        A = A + pos1;
        B = B + pos3;
        C = C + pos4;
        D = D + pos2;
    end

    for pattern_a1 = 1:4
        % Define side spring end locations
        A1 = [0.032 -0.01 -0.01 1];
        B1 = [0.032 0.01 -0.01 1];
        C1 = [0.032 0.01 0.01 1];
        D1 = [0.032 -0.01 0.01 1];
        % Define pattern to be sent to subroutine
        if pattern_a1 == 1
            A1 = A1 + pos4;
            B1 = B1 + pos2;
            C1 = C1 + pos1;
            D1 = D1 + pos3;
        elseif pattern_a1 == 2
            A1 = A1 + pos3;
            B1 = B1 + pos1;
            C1 = C1 + pos2;
            D1 = D1 + pos4;
        elseif pattern_a1 == 3
            A1 = A1 + pos2;
            B1 = B1 + pos4;
            C1 = C1 + pos3;
            D1 = D1 + pos1;
        elseif pattern_a1 == 4
            A1 = A1 + pos1;
            B1 = B1 + pos3;
            C1 = C1 + pos4;
            D1 = D1 + pos2;
        end

        counter = (pattern_a-1)*4 + pattern_a1
    end
for positions = 1:6
    temp_pos = inpos(:,positions);
    [total] = var_mod3(temp_pos,A,B,C,D,Al,Bl,Cl,Dl);
    local_k(positions, 1) = total(positions)/inpos(positions,positions);
end
allval(:,counter) = [pattern_a; pattern_a1; local_k];
end
allval = abs(allval);

H.1.2. Program var_mod3.m

function [total] = var_mod3(inpos,A,B,C,D,Al,Bl,Cl,Dl)
% Returns the total force and torque arising from spring positions defined A to D and Al to D1
% True 3 D vector model for eight spring version

% Define some constants
PlateSep = 0.032; % Define plate separation distance A1 to A at no motion = 32mm
% Define spring stiffness (N/m)
KA = 1920;
KB = KA;
KC = KA;
KD = KA;
KE = KA;
KF = KA;
KG = KA;
KH = KA;
O = [0 0 0]';

% Set terms defined by symmetry
E = A;
F = B;
G = C;
H = D;
PlateOffset = [ 2*PlateSep 0 0 0]';
E1 = A1 - PlateOffset;
F1 = B1 - PlateOffset;
G1 = C1 - PlateOffset;
H1 = D1 - PlateOffset;

% Calculate original spring lengths
lenA = sqrt(sum((Al-A).*(Al-A)));%N
lenB = sqrt(sum((Bl-B).*(Bl-B)));%N
lenC = sqrt(sum((Cl-C).*(Cl-C)));%N
lenD = sqrt(sum((Dl-D).*(Dl-D)));%N
lenE = sqrt(sum((El-E).*(El-E)));%N
lenF = sqrt(sum((Fl-F).*(Fl-F)));%N
lenG = sqrt(sum((Gl-G).*(Gl-G)));%N
lenH = sqrt(sum((Hl-H).*(Hl-H)));%N

% Define transformation matrix for XYZ rotations
cx = cos(inpos(4));
cy = cos(inpos(5));
cz = cos(inpos(6));
sx = sin(inpos(4));
sy = sin(inpos(5));
sz = sin(inpos(6));
R = [ cy*cx sx*sy*cx-cz*sx cz*sx+sy*sx
  sx*cx cy*sx+cz*sx cx*cz-sy*sx
  -sy*cx sx*cy+cz*sx cx*cy-sz*sx
  0 0 0 1];
\[
\begin{align*}
&cy*sz \quad sx*sy*sz+cx*cz \\
&sz*sy*sz-cx*cz \\
&-sy \quad sx*cy \\
&cx*cy
\end{align*}
\]

\[
T = [R_0;0 0 0 1];
\]

% Calculate new positions of nodes in origin co-ordinates

tempsum = [inpos(1 2 3)]; 0];
NA1 = T*A1 +tempsum;
NB1 = T*B1 +tempsum;
NC1 = T*C1 +tempsum;
ND1 = T*D1 +tempsum;
NE1 = T*E1 +tempsum;
NF1 = T*F1 +tempsum;
NG1 = T*G1 +tempsum;
NH1 = T*H1 +tempsum;

% Calculate spring extensions
vecA = NA1-A;
vecB = NB1-B;
vecC = NC1-C;
vecD = ND1-D;
vecE = NE1-E;
vecF = NF1-F;
vecG = NG1-G;
vecH = NH1-H;
magA = sqrt(sum(vecA.*vecA));
magB = sqrt(sum(vecB.*vecB));
magC = sqrt(sum(vecC.*vecC));
magD = sqrt(sum(vecD.*vecD));
magE = sqrt(sum(vecE.*vecE));
magF = sqrt(sum(vecF.*vecF));
magG = sqrt(sum(vecG.*vecG));
magH = sqrt(sum(vecH.*vecH));

extA = magA - lenA;
extB = magB - lenB;
extC = magC - lenC;
extD = magD - lenD;
extE = magE - lenE;
extF = magF - lenF;
extG = magG - lenG;
extH = magH - lenH;

% Calculate unit force direction vector
Ahat = vecA/magA;
Bhat = vecB/magB;
Chat = vecC/magC;
Dhat = vecD/magD;
Ehat = vecE/magE;
Fhat = vecF/magF;
Ghat = vecG/magG;
Hhat = vecH/magH;

% Calculate scalar and vector force on each member
fA = KA*extA;
fB = KB*extB;
fC = KC*extC;
fD = KD*extD;
fE = KE*extE;
fF = KE*extE;
fG = KG*extG;
fH = KH*extH;

234
FA = Ahat*fA;
FB = Bhat*fB;
FC = Chat*fC;
FD = Dhat*fD;
FE = Ehat*fE;
FF = Fhat*fF;
FG = Ghat*fG;
FH = Hhat*fH;

% Calculate total force
Alambda = (A(1)*Ahat(1) + A(2)*Ahat(2) + A(3)*Ahat(3)) / (Ahat(1)*Ahat(1) + Ahat(2)*Ahat(2) + Ahat(3)*Ahat(3));
AP = A([1 2 3]) + Alambda*Ahat([1 2 3]);
AT = cross(AP,FA([1 2 3]));
Afinal = [FA([1 2 3]);AT];
Blambda = (B(1)*Bhat(1) + B(2)*Bhat(2) + B(3)*Bhat(3)) / (Bhat(1)*Bhat(1) + Bhat(2)*Bhat(2) + Bhat(3)*Bhat(3));
BP = B([1 2 3]) + Blambda*Bhat([1 2 3]);
BT = cross(BP,FB([1 2 3]));
Bfinal = [FB([1 2 3]);BT];
Clambda = (C(1)*Chat(1) + C(2)*Chat(2) + C(3)*Chat(3)) / (Chat(1)*Chat(1) + Chat(2)*Chat(2) + Chat(3)*Chat(3));
CP = C([1 2 3]) + Clambda*Chat([1 2 3]);
CT = cross(CP,FC([1 2 3]));
Cfinal = [FC([1 2 3]);CT];
Dlambda = (D(1)*Dhat(1) + D(2)*Dhat(2) + D(3)*Dhat(3)) / (Dhat(1)*Dhat(1) + Dhat(2)*Dhat(2) + Dhat(3)*Dhat(3));
DP = D([1 2 3]) + Dlambda*Dhat([1 2 3]);
DT = cross(DP,FD([1 2 3]));
Dfinal = [FD([1 2 3]);DT];
Elambda = (E(1)*Ehat(1) + E(2)*Ehat(2) + E(3)*Ehat(3)) / (Ehat(1)*Ehat(1) + Ehat(2)*Ehat(2) + Ehat(3)*Ehat(3));
EP = E([1 2 3]) + Elambda*Ehat([1 2 3]);
ET = cross(EP,FE([1 2 3]));
Efinal = [FE([1 2 3]);ET];
Flambda = (F(1)*Fhat(1) + F(2)*Fhat(2) + F(3)*Fhat(3)) / (Fhat(1)*Fhat(1) + Fhat(2)*Fhat(2) + Fhat(3)*Fhat(3));
FP = F([1 2 3]) + Flambda*Fhat([1 2 3]);
FT = cross(FP,FF([1 2 3]));
Ffinal = [FF([1 2 3]);FT];
Glambda = (G(1)*Ghat(1) + G(2)*Ghat(2) + G(3)*Ghat(3)) / (Ghat(1)*Ghat(1) + Ghat(2)*Ghat(2) + Ghat(3)*Ghat(3));
GP = G([1 2 3]) + Glambda*Ghat([1 2 3]);
GT = cross(GP,FG([1 2 3]));
Gfinal = [FG([1 2 3]);GT];
Hlambda = (H(1)*Hhat(1) + H(2)*Hhat(2) + H(3)*Hhat(3)) / (Hhat(1)*Hhat(1) + Hhat(2)*Hhat(2) + Hhat(3)*Hhat(3));
HP = H([1 2 3]) + Hlambda*Hhat([1 2 3]);
HT = cross(HP,FH([1 2 3]));
Hfinal = [FH([1 2 3]);HT];

total = (Afinal+Bfinal+Cfinal+Dfinal+Efinal+Ffinal+Gfinal+Hfinal);
H.1.3. Program normdat.m

% normalise allval data from program main3 and put in a useful form for display

forces = allval([3:5],:);
MaxF = max(max(forces));
MinF = min(min(forces));
DeltaF = MaxF-MinF;
forces = (forces - MinF)/DeltaF;

 torque = allval([6:8],:);
MaxT = max(max(torque));
MinT = min(min(torque));
DeltaT = MaxT-MinT;
torque = (torque - MinT)/DeltaT;

newval = [forces;torque];

for i = 1:16
    Fx((allval(1,i)),(allval(2,i))) = newval(1,i);
    Fy((allval(1,i)),(allval(2,i))) = newval(2,i);
    Fz((allval(1,i)),(allval(2,i))) = newval(3,i);
    Tx((allval(1,i)),(allval(2,i))) = newval(4,i);
    Ty((allval(1,i)),(allval(2,i))) = newval(5,i);
    Tz((allval(1,i)),(allval(2,i))) = newval(6,i);
end
H.2. With Probe Offset

For the second computer model, the probe was rotated various angles about the x axis to find the local stiffness when in its operating position. The results were illustrated in chapter seven and came from:

H.2.1. Program mainreal.m

```matlab
% calls function var_real to find F/T vector from input vector [x y z Rx Ry Rz]

% Define the input position as six separate columns; each column has only one degree of freedom varying
inpos = [0.0005 0 0 0 0 0; 0 0.0005 0 0 0 0; 0 0 0.0005 0 0 0; 0 0 0 0.02 0 0; 0 0 0 0 0.02 0; 0 0 0 0 0 0.02];

% Initialise solution matrix to zero
allval = zeros(8,16);

% Define the centre of rotation relative to the centre of force
Odash = [0 0 -0.04 0]';

% Define central and side spring end locations
zero = [0 0 0 0]';
one = [1000 1000 1000 1000]';
offzero = [32 32 32 32]';
five = [5 5 5 5]';
ten = [10 10 10 10]';
fifteen = [15 15 15 15]';
a5to20 = [5 10 15 20]';
A = [zero-five-a5to20 one;zero -five -a5to20 one;zero -fifteen -a5to20 one]/1000;
B = [zero five -a5to20 one;zero ten -a5to20 one;zero fifteen -a5to20 one]/1000;
C = [zero five a5to20 one;zero ten a5to20 one;zero fifteen a5to20 one]/1000;
D = [zero -five a5to20 one;zero -ten a5to20 one;zero -fifteen a5to20 one]/1000;
A1 = [offzero -five -a5to20 one;offzero -ten -a5to20 one;offzero -fifteen -a5to20 one]/1000;
B1 = [offzero five -a5to20 one;offzero ten -a5to20 one;offzero fifteen -a5to20 one]/1000;
C1 = [offzero five a5to20 one;offzero ten a5to20 one;offzero fifteen a5to20 one]/1000;
D1 = [offzero -five a5to20 one;offzero -ten a5to20 one;offzero -fifteen a5to20 one]/1000;

% do two extra loops to gather data for three different values
% of Ty offset and Rx offset
for ii = 1:4
    Ty = (ii-1)*0.002;
    for jj = 1:4
        batchcount = (ii-1)*4 + jj;
        char = int2str(batchcount);
        Rx = ((jj-1)*5)*pi/180;
        offset_pos = [0 Ty 0 Rx 0 0];
        
        for inner = 1:12
            for outer = 1:12
                counter = (inner-1)*12 + outer;
```
batch_count = [batchcount counter]
% cycle through the six variations in displacement
for pos = 1:6
    temp_pos = inpos(:,pos);
    temp_pos = temp_pos+offset_pos;
    [total] = var_real(temp_pos,A(:,inner), B(:,inner),
                    C(:,inner), D(:,inner), A1(:,outer), B1(:,outer),
                    C1(:,outer), D1(:,outer));
    local_k(pos,l) = total(pos)/inpos(pos,pos);
end
end
end
eval(['allval' char '(:,counter) = [inner; outer; local_k;]']);
end
end
eval(['save allval' char ' dat allval' char ' -ascii'])
end

H.2.2. Program var_real.m

function [total] = var_real(inpos, A, B, C, D, A1, B1, C1, D1)
% Returns the total force and torque arising from spring positions defined A to D and A1 to D1
% True 3 D vector model for eight spring version

% Define some constants
PlateSep = 0.032;       % Define plate separation distance A1 to A at no motion = 32mm
% Define spring stiffness (N/m)
KA = 1920;
KB = KA;
KC = KA;
KD = KA;
KE = KA;
KF = KA;
KG = KA;
KH = KA;
O = [0 0 0]';
preload = 15;           % preload in Newtons on each spring

% Set terms defined by symmetry
E = A;
F = B;
G = C;
H = D;
PlateOffset = [2*PlateSep 0 0 0]';
E1 = A1 - PlateOffset;
F1 = B1 - PlateOffset;
G1 = C1 - Plate Offset;
H1 = D1 - PlateOffset;

% Calculate original spring lengths
lenA = sqrt(sum((A1-A).*(A1-A)));
lenB = sqrt(sum((B1-B).*(B1-B)));
lenC = sqrt(sum((C1-C).*(C1-C)));
lenD = sqrt(sum((D1-D).*(D1-D)));
lenE = sqrt(sum((E1-E).*(E1-E)));
lenF = sqrt(sum((F1-F).*(F1-F)));
lenG = sqrt(sum((G1-G).*(G1-G)));
lenH = sqrt(sum((H1-H).*(H1-H)));

% Define transformation matrix for XYZ rotations
\[ \begin{align*}
&cx = \cos(inpos(4)); \\
&cy = \cos(inpos(5)); \\
&cz = \cos(inpos(6)); \\
&sx = \sin(inpos(4)); \\
&sy = \sin(inpos(5)); \\
&sz = \sin(inpos(6)); \\
&R &= \begin{bmatrix}
  cy*cz & sx*sy*cz-cx*sz & cx*sy*cz+sx*sz \\
  cy*sz & sx*sy*sz+cx*cz & cx*sy*sz-sx*cz \\
  -sy & sx*cy & cx*cy
\end{bmatrix}; \\
&T = [R; 0 0 0 1];
\end{align*} \]

% Calculate new positions of nodes in origin co-ordinates
\[ \text{tempsum} = [\text{inpos}([1 2 3]); 0]; \]
\[ \text{NA1} = \text{T*A1} + \text{tempsum}; \]
\[ \text{NB1} = \text{T*B1} + \text{tempsum}; \]
\[ \text{NC1} = \text{T*C1} + \text{tempsum}; \]
\[ \text{ND1} = \text{T*D1} + \text{tempsum}; \]
\[ \text{NE1} = \text{T*E1} + \text{tempsum}; \]
\[ \text{NF1} = \text{T*F1} + \text{tempsum}; \]
\[ \text{NG1} = \text{T*G1} + \text{tempsum}; \]
\[ \text{NH1} = \text{T*H1} + \text{tempsum}; \]

% Calculate spring extensions
\[ \begin{align*}
&\text{vecA} = \text{NA1-A}; \\
&\text{vecB} = \text{NB1-B}; \\
&\text{vecC} = \text{NC1-C}; \\
&\text{vecD} = \text{ND1-D}; \\
&\text{vecE} = \text{NE1-E}; \\
&\text{vecF} = \text{NF1-F}; \\
&\text{vecG} = \text{NG1-G}; \\
&\text{vecH} = \text{NH1-H}; \\
&\text{magA} = \sqrt{\text{sum}(\text{vecA} \cdot \text{vecA})}; \\
&\text{magB} = \sqrt{\text{sum}(\text{vecB} \cdot \text{vecB})}; \\
&\text{magC} = \sqrt{\text{sum}(\text{vecC} \cdot \text{vecC})}; \\
&\text{magD} = \sqrt{\text{sum}(\text{vecD} \cdot \text{vecD})}; \\
&\text{magE} = \sqrt{\text{sum}(\text{vecE} \cdot \text{vecE})}; \\
&\text{magF} = \sqrt{\text{sum}(\text{vecF} \cdot \text{vecF})}; \\
&\text{magG} = \sqrt{\text{sum}(\text{vecG} \cdot \text{vecG})}; \\
&\text{magH} = \sqrt{\text{sum}(\text{vecH} \cdot \text{vecH})}; \\
&\text{extA} = \text{magA} - \text{lenA}; \\
&\text{extB} = \text{magB} - \text{lenB}; \\
&\text{extC} = \text{magC} - \text{lenC}; \\
&\text{extD} = \text{magD} - \text{lenD}; \\
&\text{extE} = \text{magE} - \text{lenE}; \\
&\text{extF} = \text{magF} - \text{lenF}; \\
&\text{extG} = \text{magG} - \text{lenG}; \\
&\text{extH} = \text{magH} - \text{lenH}; \\
\end{align*} \]

% Calculate unit force direction vector
\[ \begin{align*}
&\text{Ahat} = \text{vecA}/\text{magA}; \\
&\text{Bhat} = \text{vecB}/\text{magB}; \\
&\text{Chat} = \text{vecC}/\text{magC}; \\
&\text{Dhat} = \text{vecD}/\text{magD}; \\
&\text{Ehat} = \text{vecE}/\text{magE}; \\
&\text{Fhat} = \text{vecF}/\text{magF}; \\
&\text{Ghat} = \text{vecG}/\text{magG}; \\
&\text{Hhat} = \text{vecH}/\text{magH}; \\
\end{align*} \]

% Calculate scalar and vector force on each member
\[ \text{fA} = \text{KA*extA}; \]
\[ f_B = K_B \times \text{ext}_B; \]
\[ f_C = K_C \times \text{ext}_C; \]
\[ f_D = K_D \times \text{ext}_D; \]
\[ f_E = K_E \times \text{ext}_E; \]
\[ f_F = K_F \times \text{ext}_F; \]
\[ f_G = K_G \times \text{ext}_G; \]
\[ f_H = K_H \times \text{ext}_H; \]
\[ FA = Ahat \times f_A; \]
\[ FB = Bhat \times f_B; \]
\[ FC = Chat \times f_C; \]
\[ FD = Dhat \times f_D; \]
\[ FE = Ehat \times f_E; \]
\[ FF = Fhat \times f_F; \]
\[ FG = Ghat \times f_G; \]
\[ FH = Hhat \times f_H; \]

% Calculate total force
\[ \text{Alambda} = (A(1) \times Ahat(1) + A(2) \times Ahat(2) + A(3) \times Ahat(3))/\ (Ahat(1) \times Ahat(1) + Ahat(2) \times Ahat(2) + Ahat(3) \times Ahat(3)); \]
\[ \text{AP} = A([1 2 3]) + \text{Alambda} \times Ahat([1 2 3]); \]
\[ \text{AT} = \text{cross}(AP, FA([1 2 3])); \]
\[ \text{Afinal} = [FA([1 2 3]); \text{AT}]; \]
\[ \text{Blambda} = (B(1) \times Bhat(1) + B(2) \times Bhat(2) + B(3) \times Bhat(3))/\ (Bhat(1) \times Bhat(1) + Bhat(2) \times Bhat(2) + Bhat(3) \times Bhat(3)); \]
\[ \text{BP} = B([1 2 3]) + \text{Blambda} \times Bhat([1 2 3]); \]
\[ \text{BT} = \text{cross}(BP, FB([1 2 3])); \]
\[ \text{Bfinal} = [FB([1 2 3]); \text{BT}]; \]
\[ \text{Clambda} = (C(1) \times Chat(1) + C(2) \times Chat(2) + C(3) \times Chat(3))/\ (Chat(1) \times Chat(1) + Chat(2) \times Chat(2) + Chat(3) \times Chat(3)); \]
\[ \text{CP} = C([1 2 3]) + \text{Clambda} \times Chat([1 2 3]); \]
\[ \text{CT} = \text{cross}(CP, FC([1 2 3])); \]
\[ \text{Cfinal} = [FC([1 2 3]); \text{CT}]; \]
\[ \text{Dlambda} = (D(1) \times Dhat(1) + D(2) \times Dhat(2) + D(3) \times Dhat(3))/\ (Dhat(1) \times Dhat(1) + Dhat(2) \times Dhat(2) + Dhat(3) \times Dhat(3)); \]
\[ \text{DP} = D([1 2 3]) + \text{Dlambda} \times Dhat([1 2 3]); \]
\[ \text{DT} = \text{cross}(DP, FD([1 2 3])); \]
\[ \text{Dfinal} = [FD([1 2 3]); \text{DT}]; \]
\[ \text{Elambda} = (E(1) \times Ehat(1) + E(2) \times Ehat(2) + E(3) \times Ehat(3))/\ (Ehat(1) \times Ehat(1) + Ehat(2) \times Ehat(2) + Ehat(3) \times Ehat(3)); \]
\[ \text{EP} = E([1 2 3]) + \text{Elambda} \times Ehat([1 2 3]); \]
\[ \text{ET} = \text{cross}(EP, FE([1 2 3])); \]
\[ \text{Efinal} = [FE([1 2 3]); \text{ET}]; \]
\[ \text{Flambda} = (F(1) \times Fhat(1) + F(2) \times Fhat(2) + F(3) \times Fhat(3))/\ (Fhat(1) \times Fhat(1) + Fhat(2) \times Fhat(2) + Fhat(3) \times Fhat(3)); \]
\[ \text{FP} = F([1 2 3]) + \text{Flambda} \times Fhat([1 2 3]); \]
\[ \text{FT} = \text{cross}(FP, FF([1 2 3])); \]
\[ \text{Ffinal} = [FF([1 2 3]); \text{FT}]; \]
\[ \text{Glambda} = (G(1) \times Ghat(1) + G(2) \times Ghat(2) + G(3) \times Ghat(3))/\ (Ghat(1) \times Ghat(1) + Ghat(2) \times Ghat(2) + Ghat(3) \times Ghat(3)); \]
\[ \text{GP} = G([1 2 3]) + \text{Glambda} \times Ghat([1 2 3]); \]
\[ \text{GT} = \text{cross}(GP, FG([1 2 3])); \]
\[ \text{Gfinal} = [FG([1 2 3]); \text{GT}]; \]
\[ \text{Hlambda} = (H(1) \times Hhat(1) + H(2) \times Hhat(2) + H(3) \times Hhat(3))/\ (Hhat(1) \times Hhat(1) + Hhat(2) \times Hhat(2) + Hhat(3) \times Hhat(3)); \]
\[ \text{HP} = H([1 2 3]) + \text{Hlambda} \times Hhat([1 2 3]); \]
\[ \text{HT} = \text{cross}(HP, FH([1 2 3])); \]
\[ \text{Hfinal} = [FH([1 2 3]); \text{HT}]; \]

\[ \text{total} = (Afinal + Bfinal + Cfinal + Dfinal + Efinal + Ffinal + Gfinal + Hfinal); \]
H.2.3. Program normreal.m

% normalise allval data from program mainreal and put in a useful form

% first load all to find min and max values
MaxF = 0;
MinF = 1000000;
MaxT = 0;
MinT = 1000000;
for testcase = 1:16
testcase
clear allval
eval(['load allval int2str(testcase) .dat;'])
eval(['allval = allval int2str(testcase);'])

%----------------------
tran stiff = allval([3:5],:);
NewMaxF = max(max(tran stiff));
NewMinF = min(min(tran stiff));
if (NewMaxF>MaxF)
    MaxF=NewMaxF;
else
    MinF=NewMinF;
end
%----------------------
rot stiff = allval([6;8],:);
NewMaxT = max(max(rot stiff));
NewMinT = min(min(rot stiff));
if (NewMaxT>MaxT)
    MaxT=NewMaxT;
else
    MinT=NewMinT;
end

end

DeltaF = MaxF-MinF;
DeltaT = MaxT-MinT;
for ival = 1:16
clear allval
ival
eval(['load allval int2str(ival) .dat;'])
eval(['allval = allval int2str(ival);'])

%------------------------------------
tran stiff = allval([3:5],:);
tran stiff = (tran stiff - MinF)/DeltaF;
rot stiff = allval([6;8],:);
rot stiff = (rot stiff - MinT)/DeltaT;
newval = [tran stiff;rot stiff];
%------------------------------------

% put each force into a 12x12 grid
for i = 1:144
    Fx((allval(l,i)),(allval(2,i))) = newval(l,i);
    Fy((allval(l,i)),(allval(2,i))) = newval(2,i);
    Fz((allval(l,i)),(allval(2,i))) = newval(3,i);
    Tx((allval(l,i)),(allval(2,i))) = newval(4,i);
    Ty((allval(l,i)),(allval(2,i))) = newval(5,i);
    Tz((allval(l,i)),(allval(2,i))) = newval(6,i);
end
% put into a column vector that displays all the data in the form:
% inner outer KxT KyT KxR KyR KzR
newval = (round(newval*100))/100;
stiffvector = [allval(1,:)' allval(2,:)' newval'];
eval(['outfile = fopen("result' int2str(ival) '.txt","w")']);
for i1 = 1:12
  for i2 = 1:12
    count = (i1-1)*12 + i2;
    fprintf(outfile,' %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f
                          %5.2f %5.2f
',stiffvector(count,:));
  end
end
fclose(outfile);

H.2.4. Program plotgraf.m

% Program to plot stiffness variation as compliant device is translated and rotated

% Plot geometry location variables
spc = 0.02;
w = 0.225;
h = 0.225;
location = |
1*spc+0*w 4*spc+3*w h w
2*spc+1*w 4*spc+3*w h w
3*spc+2*w 4*spc+3*w h w
4*spc+3*w 4*spc+3*w h w
1*spc+0*w 3*spc+2*w h w
2*spc+1*w 3*spc+2*w h w
3*spc+2*w 3*spc+2*w h w
4*spc+3*w 3*spc+2*w h w
1*spc+0*w 2*spc+1*w h w
2*spc+1*w 2*spc+1*w h w
3*spc+2*w 2*spc+1*w h w
4*spc+3*w 2*spc+1*w h w
1*spc+0*w 1*spc+0*w h w
2*spc+1*w 1*spc+0*w h w
3*spc+2*w 1*spc+0*w h w
4*spc+3*w 1*spc+0*w h w |
shapevector = [1 4 9 12];
for innenum = 1:4
  inner = shapevector(innenum);
  for outenum = 1:4
    outer = shapevector(outenum);
    numcases = 4;
    term = (inner-1)*12 + outer
    for testcase = 1:16
      eval(['load result' int2str(testcase) '.txt;'])
      eval(['Matrix(testcase,:) = result' int2str(testcase) '(term,[3:8]);'])
      eval(['clear result' int2str(testcase) '])
    end
  end
end

% each subplot is a constant x rotation
figure(innenum)
for plotnum = 1:4

242
for i=1:4
    pattern = i*4-(4-plotnum);
    Tx(i) = Matrix(pattern,1);
    Ty(i) = Matrix(pattern,2);
    Tz(i) = Matrix(pattern,3);
    Rx(i) = Matrix(pattern,4);
    Ry(i) = Matrix(pattern,5);
    Rz(i) = Matrix(pattern,6);
end

xx = [0 2 4 6];
graphnum = (outernum-1)*4+plotnum;

% Section to custom plot data to figure
axes('position',location(graphnum,:))
plot(xx,Tx,'r',xx,Ty,'b',xx,Tz,'w',xx,Rx,'r:',xx,Ry,'b:',xx,Rz,'w:');
axis([0 6 -0.05 1])
set(gca,'XTick',[])
set(gca,'XTickLabels',[])
set(gca,'YTick',[])
set(gca,'YTickLabels',[])

end
end
end
APPENDIX I

ENGINEERING DRAWINGS FOR THE TRANSVERSE SPRING COMPLIANT DEVICE

The following drawings detail the design of the components used for the transverse spring compliant device:

<table>
<thead>
<tr>
<th>TITLE</th>
<th>QUANTITY</th>
<th>MATERIAL</th>
<th>FINISH</th>
<th>TOLERANCE</th>
<th>PROJECTION</th>
<th>ORIGINAL SCALE</th>
<th>DRAWN: Jason Tisdoll</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprung Compliant Device</td>
<td>1 off</td>
<td>Aluminum</td>
<td>Smooth</td>
<td>&lt;0.05</td>
<td></td>
<td>1:1</td>
<td>Jason Tisdoll</td>
</tr>
<tr>
<td>F/T attachment plate</td>
<td>1 off</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

25mm hex head M3 bolt with Inn hole drilled through a flat on the end

<table>
<thead>
<tr>
<th>TITLE</th>
<th>QUANTITY</th>
<th>MATERIAL</th>
<th>FINISH</th>
<th>TOLERANCE</th>
<th>PROJECTION</th>
<th>ORIGINAL SCALE</th>
<th>DRAWN: Jason Tisdoll</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprung Compliant Device</td>
<td>8 off</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ALL DIMENSIONS IN mm
Additionally, 20 pins, drilled as shown, for an interference fit with one of these three plates. Each pin to be pressed into one of the 20 holes on the larger plate section.
APPENDIX J
ENGINEERING DRAWINGS FOR THE TRANSVERSE POLYMER SPRING COMPLIANT DEVICE AND MOULD

The following drawings detail the design of the components used for the mould to manufacture the transverse polymer spring compliant device. The final three drawings show modifications to the aluminium frame from the designs shown in Appendix I.

D:\DRAWINGS\MOULD3.DWG – FASTENINGS FOR 8 MEMBER PERSPEX MOULD

8 off of 14mm M3 bolts

8 off of 45mm M5 bolts with nuts

16 off of 11mm M3 bolts with 1mm central shaft drilled through
**8 holes diameter 9mm**  
**8 holes clearance M5**

<table>
<thead>
<tr>
<th>FINISH</th>
<th>TOLERANCE</th>
<th>MATERIAL</th>
<th>PROJECTION</th>
<th>DRAWN: JPT</th>
<th>ORIGINAL SCALE</th>
<th>ALL DIMENSIONS IN mm</th>
<th>Q U A N T I T Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smooth</td>
<td>± 0.05</td>
<td>Perspex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 off</td>
</tr>
</tbody>
</table>

**D:\DRAWINGS\MOULD2.DWG - MOULD LIDS AND POLYMER POURING CHAMBER**

**Mould Lid**

Dimensions as for the other mould lids but with additional threaded holes placed around the perimeter as shown.

**Mould Lid**

**Polymer Pouring Chamber**

<table>
<thead>
<tr>
<th>FINISH</th>
<th>TOLERANCE</th>
<th>MATERIAL</th>
<th>PROJECTION</th>
<th>DRAWN: JPT</th>
<th>ORIGINAL SCALE</th>
<th>ALL DIMENSIONS IN mm</th>
<th>Q U A N T I T Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smooth</td>
<td>± 0.05</td>
<td>Perspex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 off</td>
</tr>
</tbody>
</table>
Modifications to the compliant device frame to accommodate the polymer spring elements.
<table>
<thead>
<tr>
<th>Finish</th>
<th>Tolerance</th>
<th>Material</th>
<th>Projection</th>
<th>Drawn by</th>
<th>Original Scale</th>
<th>ALL DIMENSIONS IN mm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smooth</td>
<td>± 0.05</td>
<td>Aluminium</td>
<td>Rubber compliant device</td>
<td>Jason Tisdall</td>
<td>1:1</td>
<td></td>
</tr>
</tbody>
</table>

Rubber compliant device
PUMA attachment plate
Jason Tisdall

Required eight brass M3 bolts with 9mm threaded length

<table>
<thead>
<tr>
<th>Finish</th>
<th>Tolerance</th>
<th>Material</th>
<th>Projection</th>
<th>Drawn by</th>
<th>Original Scale</th>
<th>ALL DIMENSIONS IN mm</th>
</tr>
</thead>
<tbody>
<tr>
<td>smooth</td>
<td>± 0.05</td>
<td>Aluminium</td>
<td>Spring Compliant Device</td>
<td>Jason Tisdall</td>
<td>1:1</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX K
SOFTWARE DEVELOPED TO COLLECT NEURAL NETWORK TRAINING DATA

K.1. PUMA Manipulator Control Programs

mak.moves is written in VAL II, the manipulator high level control language, and enables the manipulator end flange to go through a series of moves. Between each motion, the current manipulator position is transmitted by serial cable to the data logging PC by subroutine wer.

The flange rotations are specified as rotations about fixed axes (see chapter nine for an explanation) relative to the starting position. However, they must be given to the manipulator in its usual form as rotations about rotating axes relative to its zero (base) position. To this end, the additional programs, mak.new, mak.transform, mak.tool, mult.matrix and decom.matrix were written to transform the rotations into the correct form. These programs perform the transformation solution detailed here and illustrated in figure K.1:

1. \( bTC_{fa} \) = transformation created using initial start position of manipulator flange.
2. \( f_{fa}T_{fa} \) = transformation created using desired rotations about a fixed reference frame.
3. \( bT_{fa} = bTC_{fa}f_{fa}T_{fa} \) = resultant composite transformation matrix.
4. solution \( (bV_{fa}) \) = decomposition of \( bT_{fa} \) into a vector.
**Key:**

- **T** - Transformation
- **TC** - Constant Transformation
- **V** - Vector (of form xyzOAT)
- **VC** - Constant Vector
- **b** - base of PUMA (centre of global reference co-ordinate frame)
- **fa** - actual flange position
- **fs** - flange at start data gathering

**FIG. K.1 - TRANSFORMATION REQUIRED TO SHIFT FROM FIXED AXES TO ROTATING AXES FOR MANIPULATOR CONTROL**

<table>
<thead>
<tr>
<th>mak.moves</th>
<th>mak.transform</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 HERE thispos</td>
<td>1 DECOMPOSE xyzOAT[] = thispos</td>
</tr>
<tr>
<td>2 CALL mak.transform</td>
<td>2 t14 = xyzOAT[0]</td>
</tr>
<tr>
<td>3 CALL wer</td>
<td>3 t24 = xyzOAT[1]</td>
</tr>
<tr>
<td>4 DELAY 4</td>
<td>4 t34 = xyzOAT[2]</td>
</tr>
<tr>
<td>5 SPEED 50 MMPS ALWAYS</td>
<td>5 so = SIN(xyzOAT[3])</td>
</tr>
<tr>
<td>6 FOR loop4 = 0 TO 3</td>
<td>6 co = COS(xyzOAT[3])</td>
</tr>
<tr>
<td>7 rx = loop4 * 5 - 10</td>
<td>7 su = SIN(xyzOAT[4])</td>
</tr>
<tr>
<td>8 FOR loop5 = 0 TO 2</td>
<td>8 ca = COS(xyzOAT[4])</td>
</tr>
<tr>
<td>9 ry = loop5 * 5 - 5</td>
<td>9 st = SIN(xyzOAT[5])</td>
</tr>
<tr>
<td>10 FOR loop6 = 0 TO 4</td>
<td>10 ct = COS(xyzOAT[5])</td>
</tr>
<tr>
<td>11 rz = loop6 * 5 - 10</td>
<td>11 t11 = ((co * st) - (so * sa * ct))</td>
</tr>
<tr>
<td>12 rz = -10</td>
<td>12 t21 = ((so * st) + (co * sa * ct))</td>
</tr>
<tr>
<td>13 FOR loop1 = 0 TO 4</td>
<td>13 t31 = -(ca * ct)</td>
</tr>
<tr>
<td>14 dx = loop1 * 2</td>
<td>14 t12 = ((co * ct) + (so * sa * st))</td>
</tr>
<tr>
<td>15 FOR loop2 = 0 TO 3</td>
<td>15 t22 = ((so * ct) - (co * sa * st))</td>
</tr>
<tr>
<td>16 dy = loop2 * 2 - 4</td>
<td>16 t32 = (ca * st)</td>
</tr>
<tr>
<td>17 FOR loop3 = 0 TO 4</td>
<td>17 t13 = (so * ca)</td>
</tr>
<tr>
<td>18 dz = loop3 * 2 - 4</td>
<td>18 t23 = -(co * ca)</td>
</tr>
<tr>
<td>19 CALL mak.new</td>
<td>19 t33 = -sa</td>
</tr>
<tr>
<td>20 MOVES BASE:TRANS(x,y,z,o,a)</td>
<td>20 RETURN</td>
</tr>
<tr>
<td>21 DELAY 0.4</td>
<td></td>
</tr>
<tr>
<td>22 CALL wer</td>
<td></td>
</tr>
<tr>
<td>23 DELAY 0.1</td>
<td></td>
</tr>
<tr>
<td>24 END</td>
<td></td>
</tr>
<tr>
<td>25 END</td>
<td></td>
</tr>
<tr>
<td>26 END</td>
<td></td>
</tr>
<tr>
<td>27 ; END</td>
<td></td>
</tr>
<tr>
<td>28 END</td>
<td></td>
</tr>
<tr>
<td>29 END</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mak.new</th>
<th>mak.tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 dtool[0] = dx</td>
<td>1 p14 = dtool[0]</td>
</tr>
<tr>
<td>2 dtool[1] = dy</td>
<td>2 p24 = dtool[1]</td>
</tr>
<tr>
<td>4 dtool[3] = rx</td>
<td>4 sx = SIN(dtool[3])</td>
</tr>
</tbody>
</table>
5 \text{doool}[4] = ry
6 \text{doool}[5] = rz
7 \text{CALL mak.tool}
8 \text{CALL mul matrix}
9 \text{CALL decom matrix}
10 \text{RETURN}

5 \text{cx} = \text{COS}(\text{doool}[3])
6 \text{sy} = \text{SIN}(\text{doool}[4])
7 \text{cy} = \text{COS}(\text{doool}[4])
8 \text{sz} = \text{SIN}(\text{doool}[5])
9 \text{cz} = \text{COS}(\text{doool}[5])

\text{mult.matrix}
1 \text{s11} = t11*p11 + t12*p21 + t13*p31
2 \text{s12} = t11*p12 + t12*p22 + t13*p32
3 \text{s13} = t11*p13 + t12*p23 + t13*p33
4 \text{s14} = t11*p14 + t12*p24 + t13*p34
5 \text{s21} = t21*p11 + t22*p21 + t23*p31
6 \text{s22} = t21*p12 + t22*p22 + t23*p32
7 \text{s23} = t21*p13 + t22*p23 + t23*p33
8 \text{s24} = t21*p14 + t22*p24 + t23*p34
9 \text{s31} = t31*p11 + t32*p21 + t33*p31
10 \text{s32} = t31*p12 + t32*p22 + t33*p32
11 \text{s33} = t31*p13 + t32*p23 + t33*p33
12 \text{s34} = t31*p14 + t32*p24 + t33*p34
13 \text{RETURN}

\text{decom.matrix}
1 \text{nn} = (s13*s13 + s23*s23)
2 \text{nn} = \text{SQRT}(\text{nn})
3 \text{a} = \text{ATAN2}(-s33, \text{nn})
4 \text{IF} \text{nn < 0.001} \text{THEN}
5 \text{o} = \text{ATAN2}(\text{s22}, \text{s12})
6 \text{t} = 0
7 \text{ELSE}
8 \text{o} = \text{ATAN2}(\text{s13}, -\text{s23})
9 \text{t} = \text{ATAN2}(\text{s32}, -\text{s31})
10 \text{ENDIF}
11 \text{x} = \text{s14}
12 \text{y} = \text{s24}
13 \text{z} = \text{s34}
14 \text{RETURN}

\text{wer}
1 \text{DECOMPOSE n[] = HERE}
2 \text{byte} = 0
3 \text{FOR} \text{jt} = 0 \text{TO} 5
4 \text{data[byte]} = '-+((\text{SIGN(n[Jj])} == 1)*2); \text{n[j]} = \text{ABS(n[Jj])}
5 \text{integer} = \text{INT}(\text{n[j]})
6 \text{frac} = \text{INT}(\text{FRACT(n[j])}*1000)
7 \text{byte} = \text{byte} + 1
8 \text{IF} \text{integer} >= 1000
9 \text{data[byte]} = '0
10 \text{byte} = \text{byte} + 1
11 \text{ELSE}
12 \text{data[byte]} = '0
13 \text{END}
14 \text{END}
15 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
16 \text{IF} \text{integer} >= 1000
17 \text{data[byte]} = '0
18 \text{byte} = \text{byte} + 1
19 \text{ELSE}
20 \text{data[byte]} = '0
21 \text{END}
22 \text{integer} = \text{integer} - \text{INT}(\text{integer/1000})*1000
23 \text{IF} \text{integer} >= 1000
24 \text{data[byte]} = '0
25 \text{byte} = \text{byte} + 1
26 \text{ELSE}
27 \text{data[byte]} = '0
28 \text{byte} = \text{byte} + 1
29 \text{END}
30 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
31 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
32 \text{IF} \text{integer} >= 1000
33 \text{data[byte]} = '0
34 \text{byte} = \text{byte} + 1
35 \text{ELSE}
36 \text{data[byte]} = '0
37 \text{byte} = \text{byte} + 1
38 \text{END}
39 \text{integer} = \text{integer} - \text{INT}(\text{integer/1000})*1000
40 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
41 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
42 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
43 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
44 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
45 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
46 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
47 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
48 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
49 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
50 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
51 \text{integer} = \text{integer} \text{INT}(\text{integer/1000})*1000
52 \text{screen} = ^177566
53 \text{buffer} = \text{j713}
54 \text{jt13} = ^176526
K.2. Data Logging Software

These consist of two programs written in C. CTERM.C is the main program and reads the serial port and the analogue to digital (A/D) card at the appropriate moments. This data is written to a file called rs.dat. The program perform.c enables the software interrupt such that incoming serial messages may be read.

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <dos.h>
#include "functs.h"
#include "address.h"
#include "daqbook.h"

FILE *rs;

int main(void)
{
    // initialise variables
    int chnl = 0;       // Serial Channel to communicate with
    int maxchnls;      // Maximum number of channels
    int intno = 99;     // Software interrupt number
    char ch;
    int x = 0;
    int i = 0;
    int j = 0;
    float frequency = 2000;
    unsigned int nDataHundred[600];
    unsigned int nData[6];
    double temp = 0;
    double temp2[6];
    unsigned int count = 100;
    unsigned int level = 0;
    unsigned char oneshot = 0x0;

    rs = fopen("rs.dat","w");  // open data print file
    printf("Press any key to halt data logging\n"); // print message to screen
    ...
initialising subroutines

daqInit(PORT_ADDRESS,DMA_ADDRESS);  // initialise daqboard (A/D board)
daqAdcSetTag(DISABLE_TAG);          // disable channel tagging
if(perform_int(intno,FN_EXISTS,22,33) != 55)  // Check Serial Bios exists AX = BX+CX = 55
{
    printf("\nCOMBIOS NOT FOUND AT INT %u ABORTING......",intno);
    exit(0);
}
maxchnls=perform_int(intno,FN_PORTNO,0,0);  // Check number of active channels
if (perform_int(intno,FN_START,0,0) != 0) // Start Hardware interrupt. Say if error detected
    printf("\nSTART INTERRUPT ERROR!\n");
    printf("\n\nDATA LOGGING STARTED\n");  // State data logging started

main program loop during data logging

do
{
    if(perform_int(intno,FN_RXREADY,chnl,0)>0)  // If character received
    {
        ch=perform_int(intno,FN_RXCHAR,chnl,0);  // read it
        if ((int)ch==13)
        {
            putch(13);

            // read A/D card with a multiple averaged scan to reduce the effect of noise
            daqAdcRdScanN(START_CHANNEL,END_CHANNEL_6,nDataHundred,count,
            DtsPacerClock,oneshot.level,frequency,CHANNEL_GAIN);
            for(j=0;j<6;j++) temp2[j] = 0;
            for(i=0;i<100;i++)
            {
                for(j=0;j<6;j++)
                {
                    temp2[j] = temp2[j] + (double)nDataHundred[i*6+j];
                }
            }
            for(j=0;j<6;j++)
            {
                nData[j] = (unsigned int)(temp2[j]/100);
                fprintf(rs,"%5d ",nData[j]);
            }
        }
        else if ((int)ch==0)
        {
            x=x+1;
        }
        else
        {
            putch(ch);  // place on screen
            fprintf(rs,"%c",ch);  // place file
        }
    }
}while (kbhit()==0);  // Stop when user hits keyboard
if(perform_int(intno,FN_STOP,0,0)!=0)  // Stop hardware interrupt
    printf("\n\n\nSTOP INTERRUPT ERROR!\n");
    printf("\n\nfinished\n");
    fclose(rs);
}
// Program: PERFORM.C
// Description: Executes a software interrupt using 8086 chip

// Arguments:
//   i = Interrupt number
//   a = register ax
//   b = register bx
//   c = register cx
// Returns: Result from register ax

#include <dos.h>

unsigned int perform_int(unsigned int i, unsigned int a, unsigned int b, unsigned int c)
{
    union REGS regs;  // REGS defined in dos.h
    regs.x.ax=a;       // Load registers
    regs.x.bx=b;
    regs.x.cx=c;
    int86(i,&regs,&regs);  // Do interrupt
    return(regs.x.ax);   // Return result from AX
}
APPENDIX L
MATLAB SOFTWARE USED TO TRAIN THE ARTIFICIAL NEURAL NETWORK

L.1 - PUMA Manipulator Control Programs

This program uses functions from the Matlab Neural Network Toolbox Version 1.0.

```matlab
% program anntrain.m
% non-linear 2 layer net (11,6 neurons)
clear

% load INPUT AND TARGET VECTORS
load T.dat
load P.dat

% INITIALISE SYSTEM
S1 = 11; % No. of input neurons
[R,Q] = size(P); % size of input matrix
[S2,Q] = size(T); % size of target matrix indicates number of output neurons, S2

% SET TRAINING PARAMETERS
disp freq = 50; % training epochs between visual displays
max epoch = 4000; % maximum number of training epochs
err_goal = 1.9; % sum square error target
lr = 0.0005; % learning rate starting value
lr_inc = 1.05; % learning rate increment
lr_dec = 0.7; % learning rate decrement
momentum = 0.95; % momentum decrement
err_ratio = 1.04; % momentum increment

[W1,B1] = ntan(S1,R); % assigns random initial weights and biases
[W2,B2] = rands(S2,S1); % assigns random initial weights and biases 'neural nets initialised'

% TRAINING USING MOMENTUM & ADAPTIVE METHOD
TP = [disp_freq max epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,TE,TR] = trainbpx(W1,B1,'tansig',W2,B2,'purelin',P,T,TP);

% CALCULATE RMS ERROR
rms_err = sqrt(TR(I,TE+1)/(R*Q))

% WRITE DATA SETS TO FILE
save W1_11.dat W1 -ascii -double
save W2_11.dat W2 -ascii -double
save B1_11.dat B1 -ascii -double
save B2_11.dat B2 -ascii -double
save err_11.dat rms_err -ascii -double
```
% program mis dal1.m

% LOAD TRAINING AND VERIFICATION DATA FILES
load P.dat
load P1.dat
load T.dat
load T1.dat
load T_orig.dat
load T1_orig.dat

% LOAD NEURAL NETWORK WEIGHT AND BIAS VALUES FROM DATA FILES
load min_T.dat
load rat_T.dat
load W1_11.dat
load W2_11.dat
load B1_11.dat
load B2_11.dat

% DEFINE ANN VALUES
MIN_T = min_T;
RAT_T = rat_T;
W1 = W1_11;
W2 = W2_11;
B1 = B1_11;
B2 = B2_11;

% ===== calculate ANN output from training data ========
nl=W1*P;
outl=tansig(nl,B1);
n2=W2*out1;
T_net=purelin(n2,B2);

% ===== transform output back to original range ========
[m,n]=size(T_net);
if (m<n)
m=n;
end
allind_T=1:m;
for loop = 1:6
    T_net(loop,allind_T) = ((T_net(loop,allind_T)+1)/RAT_T(loop))+MIN_T(loop);
end

% ===== calculate ANN output from verification data ========
nl=W1*P1;
out1=tansig(n1,B1);
n2=W2*out1;
T1_net=purelin(n2,B2);
% ===== transform output back to original range ==========
[m,n]=size(T1_net);
if (m<n)
    m=n;
end
allind_T=1:m;
for loop = 1:6
    T1_net(loop,allind_T) = ((T1_net(loop,allind_T)+1)/RAT_T(loop))+MIN_T(loop);
end

% ===== calculate max and rms errors for training set =====
err = T_orig - T_net;
max_train_err = max(err)
max_abs_train_err = max(abs(err))
train_err_rms_dat1 = sqrt(mean(err' * err'))

% ===== calculate max and rms errors for verification set =====
errl = T1_orig - T1_net;
max_verif_err = max(errl)
max_abs_verif_err = max(abs(errl))
verif_err_rms_dat1 = sqrt(mean(err1' .* err1'))
APPENDIX N
MATLAB LINEAR SPRING MODEL

The inverting linear spring model consists of several functions:

1. inv_spr3.m: This is the main program and will call the function converg3.m for each force data vector. i.e. a force is applied and the appropriate input position, for the manipulator relative to the probe, to create that force is calculated.

2. def_geom.m: This defines the physical geometry of the spring end points within the linear spring model.

3. converg3.m: This will move the model in space. At each position, sol_vec3.m is called to find the resulting force. This is repeated with the input positions adjusted until the force reaches its desired value.

4. sol_vec3.m: This will solve the forward solution of the linear spring model. Given an input position, this will output the force generated by the resulting interaction.

5. trans.m: This creates a homogeneous transformation matrix given an input vector. It can be about fixed or rotating axes and a range of axis orders.

6. retrans.m: The position data from inv_spr3.m has rotations defined about fixed axes rotating about XYZ. The data used in the neural network is for changes in rotation relative to a non zero start point. These rotations are about rotating axes in the order ZYZ. It is therefore necessary to transform the position vectors from one form to the other such that the solutions can be compared.

N.1. Program inv_spr3.m

```
% program inv_spr3.m
% This will invert the linear spring compliant device
% model supplied by function sol_vec.m
% This is done in function converg3.m

% define spring geometry
```
def_geom

% load relevant data
load T1_orig.dat  % positions
load P1_orig.dat  % forces
load T1_net.dat   % positions calculated by the neural network for the forces in P1_orig

% define constants to convert force data to units of N and Nm
s1 = [409.6 409.6 409.6 409.6 409.6 409.6]';
s2 = [8.2089 8.0684 8.6804 0.3813 0.3857 0.4093]';
s3 = [0.0353 0.0372 0.0676 0.0006 0.0011]';
% define constant to ensure that co-ordinate frames match
signl = [1 1 -1 1 -1 -1]';
% pre-assign large matrices to speed up computation
allforce = zeros(6,2700);
allinpos = zeros(6,2700);
allerrflag = zeros(1,2700);

% Main loop to solve for position for each force in P1_orig
for index = 1:2700
    targetforce = ((((P1_orig(:,index))./sl).*s2)+s3).*signl;
    [newinpos,force,errflag] = converg3(targetforce);
    allforce(:.index) = force;
    allinpos(:.index) = newinpos;
    allerrflag(index) = errflag;
end

N.2. Program def_geom.m

%program def_geom.m
% defines the geometry of the eight spring compliant device to as nearly as possible
% re-create the real device characteristics

% define global variables
global INNER
global OUTER
global KSTIFF

% define true central and side spring contact positions in metres relative to the centre of rotation
A = [0 -0.005 -0.015 1]';
B = [0 0.005 -0.015 1]';
C = [0 0.005 0.015 1]';
D = [0 -0.005 0.015 1]';
A1 = [0.028 -0.005 -0.005 1]';
B1 = [0.028 0.005 -0.005 1]';
C1 = [0.028 0.005 0.005 1]';
D1 = [0.028 -0.005 0.005 1]';

% define geometrical constants
PlateSep = 0.028;  % distance from A1 to A at no motion = 28mm
centreXoffset = [0.003 0 0 0]';
K = ones(1,8)*1920;  % spring stiffnesses in N/m

% define spring positions defined by symmetry
E = A - centreXoffset;
F = B - centreXoffset;
G = C - centreXoffset;  
H = D - centreXoffset;  
A = A + centreXoffset;  
B = B + centreXoffset;  
C = C + centreXoffset;  
D = D + centreXoffset;  
PlateOffset = [2*PlateSep 0 0 0]';
E1 = A1 - PlateOffset;  
F1 = B1 - PlateOffset;  
G1 = C1 - PlateOffset;  
H1 = D1 - PlateOffset;  

% define global variables
INNER = [A B C D E F G H];  
OUTER = [A1 B1 C1 D1 E1 F1 G1 H1];  
KSTIFF = K;

N.3. Program converg3.m

function [newinpos,force,errflag] = converg3(targetforce)
% function converg3.m
% Given an input target force, this will rotate and translate
% the spring model about fixed axes to minimise the error.
% errflag = 1 is returned if the force error target is not reached
% within the prescribed maximum number of iterations.

clear allforce  
threshold = 0.01;  
brakethreshold = 100;

% minimisation gains for PID minimisation
kp = 3*[0.01 0.01 0.01 0.01 0.01 0.01 0.01];
kd = [0 0 0 0 0 0];
ki = [0.05 0.05 0.05 0.5 0.05 0.05];

% preset variables
oldc = zeros(6,1);  
e1 = zeros(6,1);  
e2 = zeros(6,1);  
err = ones(6,1);  
newinpos = [0 0 0 0 0 0];

% solve spring model for starting value
[force] = sol_vec3(newinpos);

% minimisation loop
i=0;  
while((max(abs(err))>threshold)&(i<=breakthreshold))  
i=i+1;  
[force] = sol_vec3(newinpos);  
err=(targetforce-force);  
e = err;  
c = oldc + (kp+kd+ki).*e - (kp+2*kd).*e1 + kd.*e2;  
oldc = c;  
e2 = e1;  
e1 = e;  
newinpos = c;  
allforce(:,i) = force;
\begin{verbatim}
end
crflag=0;
if(i==breakthreshold+1)
    crflag=1;
end

N.4. Program sol_vec3.m

function [force] = sol_vec3(inpos)
% True 3 D vector model for eight spring version.
% for an input position will return the force acting at the
% centre of the force sensor.

% global variables defined in gef_geom.m
global INNER
global OUTER
global KSTIFF

% inner and outer define the spring end points
inner = INNER;
outer = OUTER;
K = KSTIFF;

% convert inpos to m from mm
inpos = inpos./[1000 1000 1000 1 1 1]';

% define original spring lengths
len = ones(1,8)*0.0177;

% define transformation matrix for XYZ rotations about fixed axes
T = trans(inpos,'pre','xyz');
NAH = T*outer;

% calculate spring extensions
vec = NAH-inner;
mag = sqrt(sum(vec.*vec));
ext = mag-len;

% calculate the unit force direction vector
hat = vec./[mag;mag;mag;mag];

% calculate the scalar and vector force on each member
f = K.*ext;
f=[f;f;f;f;f];
FAH = hat.*f;

% set inner co-ordinates relative to centre of force
CFrelCR = [0 0.014 0.0465 0]';
CFrelCR = [CFrelCR CFrelCR CFrelCR CFrelCR CFrelCR CFrelCR CFrelCR CFrelCR];
inner = inner-CFrelCR;

lambda = sum(inner([1:3],:).*hat([1:3],:))./sum(hat([1:3],:).*hat([1:3],:));
lambda = [lambda;lambda;lambda;lambda];
P = inner([1:3],:) + lambda.*hat([1:3],:);
TT = cross(P,FAH([1:3],:));
final = [FAH([1:3],:);TT];

force = (sum(final))';
\end{verbatim}
function [T] = trans(inpos, def_string1, def_string2)

% TRANS - converts euler type co-ordinates to a homogeneous transformation matrix function:
% [T] = trans(inpos, def_string1, def_string2)
% It returns a 4x4 transformation matrix, T, given an euler type input position of the form:
% 'xyz O A T' or 'xyz X Y Z'
% WHERE THE ANGLES MUST BE IN RADIANS.
% The function will set default values if the second and/or the third input arguments are absent.
% The def_string1 allows either 'pre' or 'post' (default 'pre'). 'pre'-multiplying if rotating about
% fixed axes. 'post'-multiplying if rotating about rotating axes.
% The def_string2 allows 'xyz', 'oat' or 'puma' (default 'xyz').
% 'xyz'- uses Euler angles in that order.
% 'oat'- uses Euler angles as for PUMA (z>z) but in true global axes.
% 'puma'- uses Euler angles as the PUMA does i.e. PUMA rotation = TRUE rotation +[90 -90 0].

% check for correct function call including default
if nargin==1
    def_string1 = 'pre';
    def_string2 = 'xyz';
end
if nargin==2
    def_string1 = 'pre';
    def_string2 = 'xyz';
end
errorchk = 0;
if nargin==3
    if strcmp('pre', def_string1)==0, errorchk=errorchk+1; end
    if strcmp('post', def_string1)==0, errorchk=errorchk+1; end
    if strcmp('xyz', def_string2)==0, errorchk=errorchk+1; end
    if strcmp('oat', def_string2)==0, errorchk=errorchk+1; end
    if strcmp('puma', def_string2)==0, errorchk=errorchk+1; end
    if errorchk>=4, 'error in arguments'
        return
    end
end

% DEFINE TRANSFORMATION MATRIX FOR X Y Z ROTATIONS
[m,n]=size(inpos);
if m>n
    O = inpos([1 2 3]);
else
    O = inpos([1 2 3])';
end
cx = cos(inpos(4));
cy = cos(inpos(5));
cz = cos(inpos(6));
sx = sin(inpos(4));
sy = sin(inpos(5));
sz = sin(inpos(6));
if strcmp('xyz', def_string2)==1
RI = \[
\begin{bmatrix}
1 & 0 & 0 \\
-cx & 0 & sx \\
0 & sx & cx
\end{bmatrix};
\]
R2 = \[
\begin{bmatrix}
cy & 0 & sy \\
0 & 1 & 0 \\
-sy & 0 & cy
\end{bmatrix};
\]
R3 = \[
\begin{bmatrix}
cz & -sz & 0 \\
sz & cz & 0 \\
0 & 0 & 1
\end{bmatrix};
\]
elseif strcmp('oat', def_string2) == 1
RI = \[
\begin{bmatrix}
cx & -sx & 0 \\
sx & cx & 0 \\
0 & 0 & 1
\end{bmatrix};
\]
R2 = \[
\begin{bmatrix}
cy & 0 & sy \\
0 & 1 & 0 \\
-sy & 0 & cy
\end{bmatrix};
\]
R3 = \[
\begin{bmatrix}
cz & -sz & 0 \\
sz & cz & 0 \\
0 & 0 & 1
\end{bmatrix};
\]
else
% puma solution
RI = \[
\begin{bmatrix}
sx & 0 & -sx \\
cx & 0 & sx \\
0 & 0 & 1
\end{bmatrix};
\]
R2 = \[
\begin{bmatrix}
-cy & 0 & cy \\
0 & 1 & 0 \\
-cy & 0 & -cy
\end{bmatrix};
\]
R3 = \[
\begin{bmatrix}
cz & -sz & 0 \\
sz & cz & 0 \\
0 & 0 & 1
\end{bmatrix};
\]
end

if strcmp('pre', def_string1) == 1
% premultiplying for the case where subsequent rotations are about the original axis
RR1 = R3*R2*R1;
% equivalent to
% R = \[
\begin{bmatrix}
cy*cz & sx*sy*cz-cx*sz & cx*sy*cz+sx*sz \\
sy*cz & sy*cz*sz & sy*cz*sz
\end{bmatrix};
\]
else
% postmultiplying for the case where subsequent rotations are about the rotated axes
RR1 = R1*R2*R3;
end

% put rotation into transformation matrix
T = [RR1 0 0 0 1];

N.6. Program retrans.m

---

% program retrans.m.
% transforms position data calculated in inv_sp3.m into a form which is equivalent to the original inpos data form gathered by the PUMA manipulator

def_geom
load allinpos.dat
pumastart = [0 0 0 90*pi/180 90*pi/180 -90*pi/180]';
bTs = trans(pumastart,'post','oat');
newinpos=zeros(6,2700);
for i=1:2700
norminpos = allinpos(:,i)
sTa = trans(norminpos,'pre','xyz');
bTa = bTs*sTa;

% decompose bTa
T=bTa;
N=sqrt((T(1,3)*T(1,3)+(T(2,3)*T(2,3)));
Y=atan2(-T(3,3),N);
if (N<0.001)
X=atan2(T(1,2),T(2,2));
Z=0;
else
X=atan2(T(1,3),-T(2,3));
Z=atan2(T(3,2),-T(3,1));
end
---

266
bVa = [T([1:3],4):X;Y;Z];
% move into global frame
inpos = bVa - [0 0 0 90*pi/180 -90*pi/180 0]';

% transform into position relative to start position
inpos = inpos - pumastart;
if (inpos(4)<pi)
    inpos(4) = 2*pi-inpos(4)
end
newinpos(:,i) = inpos;
end
[force]= sol_vec3(norminpos)
[force]= sol_vec(inpos)
<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Year</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahmad S.</td>
<td>1993</td>
<td>Constrained Motion (Force/Position) Control of Flexible Joint Robots</td>
</tr>
<tr>
<td>Aleksander I.</td>
<td>1989</td>
<td>Neural computing Architectures</td>
</tr>
<tr>
<td>American Petroleum Institute</td>
<td>1993</td>
<td>Recommended Practice for planning, designing and constructing fixed offshore platforms</td>
</tr>
<tr>
<td>Asada H. and Slotine J. E.</td>
<td>1986</td>
<td>Robot analysis and control</td>
</tr>
<tr>
<td>Ashby M.F. and Jones D.R.H.</td>
<td>1983</td>
<td>Engineering Materials - An Introduction to their Properties and Applications</td>
</tr>
<tr>
<td>Backes P.G., Leininger G.G. and Chung C.H.</td>
<td>1985</td>
<td>Real Time Cartesian Co-ordinate Hybrid Control of a PUMA 560 Manipulator</td>
</tr>
<tr>
<td>Reference</td>
<td>Author(s)</td>
<td>Title</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>Broome 96</td>
<td>Broome D.R. and Tisdall J.P.</td>
<td>Measuring the Force Induced Motion of an ROV Attached to an Underwater Structure by Three Vacuum Feet</td>
</tr>
<tr>
<td>De Schutter 88a</td>
<td>De Schutter J. and Van Brussel H.</td>
<td>Compliant Robot Motion: I. A Formalism for Specifying Compliant Motion Tasks</td>
</tr>
</tbody>
</table>
De Schutter 88b  
De Schutter J. and Van Brussel H.  
*Compliant Robot Motion: II. A Control Approach Based on External Control Loops*  
The International Journal of Robotics Research 1988 Vol. 7 No. 4 pp 18-33

DEC 78  
Version 4.0, 1978

Denavit 55  
Denavit J. and Hartenberg R.S. (1955)  
*A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices.*  

Duffy 90  
Duffy J.  
*The Fallacy of Modern Hybrid Control Theory that is Based on 'Orthogonal Complements' of Twist and Wrench Spaces*  
Editorial: Journal of Robotic Systems 1990

Dunnigan 96  
Dunnigan M.W., Lane D.M., Clegg A.C. and Edwards I  
*Hybrid Position/Force Control of a Hydraulic Underwater Manipulator*  

Edwards 95  
Edwards G.R.  
*Inspection of North Sea Oil Platforms; 25 Years of Equipment Development*  
Proceedings SPIE - The International Society for Optical Engineering 1995 pp 98-103

Elosegui 90  
Elosegui P., Daniel R.W. and Sharkey P.M.  
*Joint Servoing For Robust Manipulator Force Control*  
Proceedings IEEE 1990 pp 246 - 251

Elsharkawi 92  
Elsharkawi A. and Reinisch H.  
*Neural Based Controller for Robotic Fine Manipulation*  

Eppinger 87  
Eppinger S.D. & Seering W.P.  
*Understanding Bandwidth Limitations in Robot Force Control*  
Proceedings IEEE 1987 pp 904 - 909

Eppinger 89  
Eppinger S.D. & Seering W.P.  
*Three Dynamic Problems in Robot Force Control*  
Proceedings IEEE 1989 pp 393 - 397

271
Fearing 91  
Fearing R. S.  
*Using a cylindrical tactile sensor for determining curvature*  
IEEE Transactions on Robotics and Automation 1991 Vol. 7 No. 6 pp 806 - 817

Fu 87  
Fu K.S., Gonzalez R.C. & Lee C.S.G.  
*Robotics: Control, Sensing, Vision and Intelligence*  
Published McGraw-Hill International 1987  
ISBN 0-07-100421-1

Goddard 92  
Goddard Ralph E., Zheng Yuan F. and Hooshang Hemami  
*Dynamic Hybrid Velocity/Force Control of Robot Compliant Motion over Globally Unknown Objects*  
IEEE Transactions on Robotics and Automation Vol. 8 No. 1 Feb. 1992

Greig 92  
Greig A. R.  
*Automatic Inspection of Complex Geometry Welds*  
Ph.D. Thesis. Dept. of Mechanical Engineering, University College London. 1992

Greig 94  
Greig A.R. and Broome D.R.  
*Development of a robotic underwater manipulator*  
Transactions if the Institute of Marine Engineers 1994 Vol. 106 Part 5 pp 217-229

Greig 96  
Greig A.R. and Lovell S.P.  
*Improvement of the Accuracy of a PUMA 560 Industrial Manipulator by Calibration*  

Hebb 49  
Hebb D.O.  
*The organization of behaviour*  
New York: Wiley, 1949

Hogan 85  
Hogan N.  
*Impedance Control: An Approach to Manipulation part I, II and III*  

Horne 90  
Horne B., Jamshidi M. and Vadiee N.  
*Neural Networks in Robotics: A Survey*  
Journal of Intelligent and Robotic Systems 1990 Vol. 3 pp 51-66

272
Hyde 93
Hyde J.M. and Cutkosky M.R.
*Contact Transition Control: An Experimental Study*

Irwin 95
Irwin G. W., Warwick K. and Hunt K.J.
*Neural Network Applications in Control*
Pub. IEE, 1995

Kazerooni 86 a
Kazerooni H., Sheridan T.B. and Houpt P.K.
*Robust Compliant Motion for Manipulators: Parts I and II*

Kazerooni 86 b
Kazerooni H., Bausch J.J. and Kramer B.M.
*An Approach to Automated Deburring by Robot Manipulators*

Khatib 87
Khatib O.
*A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation*

Kim 90
Kim W.S. and Bejczy A.K
*A Stability Analysis of Shared Compliance Control*
USA Symposium on Flexible Automation 1990 Japan pp 567-572

Kohonen 87
Kohonen T.
*Self-organization and associative memory*
Berlin: Springer-Verlag, 1987

Lane 97
Lane D.M., Dunnigan M.W., Clegg A.C., Dauchez P. and Cellier L.
*A Comparison Between Robust and Adaptive Hybrid Position/Force Control Schemes for Hydraulic Underwater Manipulators*

Langrock 94
Langrock and Broome D.R.
*Development of a Telerobotic Underwater Manipulator*
Insight journal 1994 Vol. 36 No. 9 pp 666-671

Lian 91
Lian K.Y., Jean J.H. and Fu L.C.
*Adaptive Force Control of Single-Link Mechanism with Joint Flexibility*
IEEE Transactions on Robotics and Automation 1991 Vol. 7 No. 4 pp 540-545

273
Lipkin 88
Lipkin H. and Duffy J.
*Hybrid Twist and Wrench Control for a Robotic Manipulator*
transactions of the ASME journal of Mechanisms, Transmissions and Automation in Design 1988 Vol. 110 pp138-144

Lippmann
Lippmann R.P.
*A Survey of Neural Network Models*
MIT Lincoln Laboratory, Lexington, MA

Lugg 96
Lugg M.
*UNDERWATER ACFM: Inspection Procedure for use with the Model U11 ACFM Microgauge*
Milton Keynes, UK, revision 44, 1996

Marth 93
Marth G.T., Tarn T.J. and Bejczy A.K.
*Stable Phase Transition Control for Robot Arm Motion*

Mason 81
Mason M. T.
*Compliance and Force Control for Computer Controlled Manipulators*

Massey 89
Massey B. S.
*Mechanics of Fluids*

Matlab NN 93
Matlab Neural Networks Toolbox Manual 1993

Matlab UG 92

Morellan 89
Morellan J. R.
*Robust back propagation: connectionist learning algorithms resistant to various types of noise*

Nguyen 90
Nguyen D. and Widrow B.
*Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights*
International Joint Conference on Neural Networks, Vol. 3, pp. 21-26, July 1990

274
Nicolodi 90  
Nicolodi S., Visentin R. and Rotella A. 
Supervisory controlled telemanipulation with automatic compensation of manipulator base movements 
Proceedings of the 8th International ROV Conference, 1990

Ole 93  
Ole Jakob Elle, Trygve Thomessen and Terje Lien 
Tele-Manipulation using Guided Contour Tracking 
ISOPE - 1993  Singapore

Patel 89  
Patel M.H. 
Dynamics of Offshore Structures 
Published by Butterworth & Co. Ltd. 1989 
ISBN 0-408-01074-6

Pati 92  
Pati Y.C., Krishnaprasad P.S. and Peckerar M.C. 
An Analogue Neural Network Solution to the Inverse Problem of "Early Taction" 
IEEE Transactions on Robotics and Automation 1992 Vol. 8 No. 2 pp 196-211

Quian 92  
Quian H.P. and De Schutter J. 
Introducing Active Linear and Non-Linear Damping to Enable Stable High Gain Force Control in Case of Stiff Contact 

Raibert 81  
Raibert M.H. and Craig J.J. 
Hybrid Position/Force Control of Manipulators 
Transactions of the ASME, Vol 102, pp 126 - 133, June 1981

Roberts 85  
Roberts R.K., Paul R.P. and Hillberry B.M. 
The Effect of Wrist Force Sensor Stiffness on the Control of Robotic Manipulators 
Proceedings of the IEEE conference on Robotics and Automation 1985 pp 269-274

Rosenblatt 61  
Rosenblatt F. 
Principles of Neurodynamics 
1961 Washington DC, Spartan Press

Rumelhart 86  
Rumelhart D.E., Hinton G.E. and Williams R.J. 
"Learning internal representations by error propagation" 
Rumelhart D.E. and McClelland J. editors of 
Parallel Data Processing 
Cambridge, MA, the MIT press 
Sinha P. R., Yangsheng X., Ruzena K. B. & Paul R. P.
Robotic Exploration of Surfaces with a Compliant Wrist Sensor

Stacey A. and Sharp J.V.
Re-assessment of offshore structures using the revised HSE fatigue guidance

Stonham T. J.
Neural networks - how they work - their strengths and weaknesses
Proceedings of Conference of neural networks, neuro-fuzzy and other learning systems for engineering applications and research London 1994 (EPSRC information technology awareness in engineering)

Tan P.Y., Lim G., Chua K., Wong F.S. and Neo S.
Comparative Studies Among Neural Nets, Radial Basis Functions and Regression Methods

Tao J.M. and Luh J.Y.S.
Application of Neural Network with Real-Time Training to Robust Position/Force Control of Multiple Robots

Townsend W. T. and Salisbury J. K.
The Effect of Coulomb Friction and Stiction on Force Control

Tsujimura T. and Yabuta T.
Object detection by tactile sensing method employing force/torque information
IEEE Transactions on Robotics and Automation 1989 Vol.5 No. 4 pp 444 - 450

Unimation PUMA 500 Mk II Technical Manual 1985

Unimation PUMA 500 Mk II - Users guide to VAL II. Ver. 1.4b 1985
Venkataraman 93
Venkataraman S.T., Gulati S., Barhen J. & Toomarian N.
A Neural Network Based Identification of Environments Models for Compliant Control of Space Robots
IEEE Transactions on Robotics and Automation 1993 Vol. 9 No. 5 pp 685-696

Volpe 92
Volpe R. and Khosla P.
An experimental Evaluation and Comparison of Explicit Force Control Strategies for Robotic Manipulators.

Waibel 91
Waibel B.J. and Kazerooni H.
Theory and Experiments on the Stability of Robot Compliance Control
IEEE Transactions on Robotics and Automation 1991 Vol. 7 No. 1 pp 95-104

Walter 93
Walter J.A. and Schulten K.J.
Implementation of Self-Organising Neural Networks for Visuo-Motor Control of an Industrial Robot
IEEE Transactions on Neural Networks 1993 Vol. 4 No. 1 pp 86-95

Wang 91
Wang Q., Broome D.R. and Daycock I.T.W
Adaptive Force Control of the PUMA 560 Industrial Robot Manipulator.
1991 IEE International Conference on Control pp 358 - 362

Wang 92
Wang Q.
Study on the Dynamic and Force Control of a Robotic Manipulator.

West 85
West H. and Asada H
A Method for the Design of Hybrid Position/Force Controllers for Manipulators Constrained by Contact with the Environment
Proceedings of the IEEE conference on Robotics and Automation 1985 pp 251-259

White 79
White F.M.
Fluid Mechanics
pub. McGraw-Hill Kogakusha 1979

Whitney 87
Whitney D.E.
Historical Perspective and State of the Art in Robot Force Control.
International Journal of robotics research. Vol. 6 No 1 spring 1987 pp 3 - 13

277
Widrow 85
Widrow B. and Sterns S.D.
*Adaptive signal processing*

Wilfinger 93
Wilfinger L.S. et. al.
*Integral Force Control with Robustness Enhancement.*
Proceedings 1993 IEEE International Conference on robotics and automation. pp 100 - 105

Zhang 85
Zhang H. and Paul R.P.
*Hybrid Control of Robot Manipulators*
IEEE 1985

Zhou 91
Zhou Y.
*On the Hard Contact Adaptive Control of a Manipulator*
IEEE International Conference on Systems Engineering 1991 pp 105-108

Zhou 92
Zhou Y. and Han Z.J.
*Adaptive Force Control of Construction Robot*
Asia-Pacific Engineering Journal (part A) 1992 Vol. 2 No. 1 pp 133-159