Transfer of manipulation skills from human to machine through demonstration in a haptic rendered virtual environment

Yuxin Chen
University of Wollongong

UNIVERSITY OF WOLLONGONG
COPYRIGHT WARNING
You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced, transmitted, adapted, performed, broadcast, published or displayed in any way or in any medium, electronically or mechanically, including photocopying and recording, without the prior written permission of the copyright owner.

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Recommended Citation
Chen, Yuxin, Transfer of manipulation skills from human to machine through demonstration in a haptic rendered virtual environment, PhD thesis, School of Electrical, Computer and Telecommunications Engineering, University of Wollongong, 2005.
http://ro.uow.edu.au/theses/283
Transfer of manipulation skills from human to machine through demonstration in a haptic rendered virtual environment

Yuxin Chen

This thesis is presented as full requirements for the award of a PhD at the University of Wollongong

January 2005
ABSTRACT

Robots are widely used as automation tools to improve productivity in industry. Force sensitive manipulation is a generic requirement for a large number of industrial tasks, especially those associated with assembly. One of the major factors preventing greater use of robots in assembly tasks to date has been the lack of availability of fast and reliable methods of programming robots to carry out such tasks. Hence robots have in practice been unable to economically replicate the complex force and torque sensitive capability of human operators.

A new approach is explored to transfer human manipulation skills to a robotics system. The teaching of the human skills to the machine starts by demonstrating those skills in a haptic-rendered virtual environment. The experience is close to real operation as the forces and torques generated during the interaction of the parts are sensed by the operator. A skill acquisition algorithm utilizes the position and contact force/torque data generated in the virtual environment combined with a priori knowledge about the task to generate the skills required to perform such a task. Such skills are translated into actual robotic trajectories for implementation in real time.

The peg-in-hole insertion problem is used as a case study.

A haptic rendered 3D virtual model of the peg-in-hole insertion process is developed. The haptic or tactile rendering is provided through a haptic device. A multi-layer method is developed to derive and learn the basic manipulation skills from the virtual manipulation carried out by a human operator. The force and torque data generated through virtual manipulation are used for skill acquisition.

The skill acquisition algorithm primarily learns the actions which result in a proper change of contact states. Both optimum sequences and normal operation rules are learned and stored in a skill database. If the contact state is not among or near any state in the optimum sequences stored in the skill database, a corrective strategy is applied until a state among or near a state in the optimal space is produced. On-line incremental learning is also used for new cases encountered during physical manipulation.

The approach is fully validated through an experimental rig set up for this purpose and the results are reported.

Keywords: Phantom, haptics, skill learning, virtual reality, peg-in-hole insertion
ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my supervisor, Associate Professor Fazel Naghdy, for providing great help and support during the course of my study. Thanks are also due to Professor Chris Cook, Dr Steven Gower and my co-supervisor Associate Professor Golshah Naghdy for their support of the project. Likewise, I wish to express my gratitude to CRC for providing me with a scholarship and University of Wollongong for further financial support. I would also like to thank the staff members in School of Electrical, Computer and Telecommunications Engineering for their assistance. Special thanks are given to University of Wollongong. Last but not least, I wish to thank my family, friends and all helpful people.
TABLE OF CONTENTS

ABSTRACT .................................................................................................................. i
ACKNOWLEDGEMENTS ........................................................................................... ii
TABLE OF CONTENTS .............................................................................................. iii
LIST OF FIGURES ...................................................................................................... vi
LIST OF TABLES ......................................................................................................... x
ABBREVIATION ........................................................................................................ xi

CHAPTER 1  Introduction .......................................................................................... 1
  1.1  Problem Statement ....................................................................................... 1
  1.2  Overall Approach ......................................................................................... 3
  1.3  Aim and Contributions of Thesis ................................................................. 5
  1.4  Thesis Organization ..................................................................................... 6

CHAPTER 2  Background .......................................................................................... 8
  2.1  Introduction .................................................................................................. 8
  2.2  Overview of Robot Programming ................................................................ 8
  2.3  Text Programming ..................................................................................... 10
  2.4  Off-Line Simulation-Based Programming ................................................. 12
  2.5  Inductive Learning ..................................................................................... 13
  2.6  Teaching by Guiding ................................................................................. 14
  2.7  Teaching by Showing ................................................................................. 17
  2.8  Teaching by Demonstration through A Virtual Environment ................... 20
  2.9  Chapter Summary ....................................................................................... 22

CHAPTER 3  Virtual Manipulation Environment ................................................... 24
  3.1  Introduction ................................................................................................ 24
  3.2  Virtual Peg-in-Hole Insertion Environment ............................................... 25
      3.2.1  Graphic Model ................................................................................... 26
      3.2.2  Haptic Rendered Model ..................................................................... 29
          3.2.2.1  Spring-Damper System .............................................................. 33
          3.2.2.2  Geometric Model ....................................................................... 33
          3.2.2.3  PointShell Method .................................................................... 35
          3.2.2.4  Normal Force ............................................................................. 38
          3.2.2.5  Friction Force ............................................................................. 39
      3.2.3  Collision Detection and Depth Calculation ....................................... 40
      3.2.4  Data from Virtual Environment ......................................................... 45
  3.3  Another Haptic Rendering Method ............................................................. 51
  3.4  Chapter Summary ....................................................................................... 53

CHAPTER 4  Skill Transfer System ......................................................................... 55
  4.1  Introduction ................................................................................................ 55
  4.2  Imitation and Skill Transfer ........................................................................ 55
  4.3  Steps in Skill Transfer to Machines ............................................................ 57
7.6.1 Higher Degree of Freedom Haptic Device ...................................... 141
7.6.2 Skill Acquisition Algorithm ............................................................. 142
7.6.3 Transportability of the Approach ..................................................... 142

REFERENCES ............................................................................................. 143

APPENDIX A Experimental Rig and Rig Control Programs .................... 164
A. 1 Introduction ....................................................................................... 164
A. 2 Hardware Components ....................................................................... 164
   A.2.1 Sanyo Denki 2 Phase Stepper Motor ......................................... 164
   A.2.2 Gecko G201 7A, 80V 10 Microstep Drive ................................... 165
   A.2.3 DC Servo Motor W/Encoder ....................................................... 167
   A.2.4 DC Servo Controller ................................................................. 169
   A.2.5 DAQ-802 Board and DaqEZ Professional Software .................. 170
   A.2.6 Decoder ..................................................................................... 172
   A.2.7 F/T Sensor ................................................................................ 173
A. 3 Rig Control Programs ......................................................................... 174
   A.3.1 Program for Get the Force/Torque Information .......................... 175
      A.3.1.1 Using Djgpp as A Compiler ............................................... 175
      A.3.1.2 Using VC++ as A Compiler ............................................... 177
   A.3.2 Head File for Position Getting, Hole Turning and Peg Moving ..... 179

APPENDIX B Related Publications .............................................................. 183
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Fig. 1-1</th>
<th>Overall model of the system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 2-1</td>
<td>System enabling a robot to follow a behaviour prescribed by the user [Ahmad(online)]</td>
</tr>
<tr>
<td>Fig. 2-2</td>
<td>Levels of robot programming [Lawson(on-line)]</td>
</tr>
<tr>
<td>Fig. 2-3</td>
<td>Visual programming -- ToonTalk Append Example [Visual(online)]</td>
</tr>
<tr>
<td>Fig. 2-4</td>
<td>Text Programming: A simple program to count from 1 to 10 [Mechatronics(online)]</td>
</tr>
<tr>
<td>Fig. 2-5</td>
<td>Basic architecture of the model based off-line system supplied by AMROSE [MOBARC(online)]</td>
</tr>
<tr>
<td>Fig. 2-6</td>
<td>Screenshot of IGRIP, by Deneb [Lawson(on-line)]</td>
</tr>
<tr>
<td>Fig. 2-7</td>
<td>Direct teaching of tool manipulation skills (a) Measure of hand motion; (b)Replication of human expert; [Asada88]</td>
</tr>
<tr>
<td>Fig. 2-8</td>
<td>Language-aided robotic teleoperation system [Sato87]</td>
</tr>
<tr>
<td>Fig. 2-9</td>
<td>Using a virtual slave arm to overcome force feedback instabilities due to transmission delays [Kotoku92]</td>
</tr>
<tr>
<td>Fig. 3-1</td>
<td>Graphic or geometric model used for both visual rendering and haptic rendering [Mendoza(online)]</td>
</tr>
<tr>
<td>Fig. 3-2</td>
<td>Graphic polygons in the peg and the hole. (a) Graphic model of the peg (8 sides are assumed); (b) Graphic model of the hole</td>
</tr>
<tr>
<td>Fig. 3-3</td>
<td>Peg-in-hole insertion virtual environment and coordinates of the real rig</td>
</tr>
<tr>
<td>Fig. 3-4</td>
<td>Facet, force normal to the plane and ambiguity. (a) Facet; (b) force normal to the plane; (c) ambiguity</td>
</tr>
<tr>
<td>Fig. 3-5</td>
<td>Spring-damper system</td>
</tr>
<tr>
<td>Fig. 3-6</td>
<td>Triangle polygon mesh of the hole</td>
</tr>
<tr>
<td>Fig. 3-7</td>
<td>Cylindrical approach to building the geometrical model of the hole</td>
</tr>
<tr>
<td>Fig. 3-8</td>
<td>PointShell and singular points</td>
</tr>
<tr>
<td>Fig. 3-9</td>
<td>Voxmap. (a) Voxmap (shown only in 2D); (b) Collision (penetration)</td>
</tr>
<tr>
<td>Fig. 3-10</td>
<td>The first approach used in haptic rendering modelling</td>
</tr>
<tr>
<td>Fig. 3-11</td>
<td>The second method used in haptic rendering modelling</td>
</tr>
<tr>
<td>Fig. 3-12</td>
<td>Normal force and friction force</td>
</tr>
<tr>
<td>Fig. 3-13</td>
<td>Definition of the strain $z$</td>
</tr>
<tr>
<td>Fig. 3-14</td>
<td>Definition of depth</td>
</tr>
<tr>
<td>Fig. 3-15</td>
<td>Collision detection and response for haptic rendering</td>
</tr>
<tr>
<td>Fig. 3-16</td>
<td>Force and torque generation</td>
</tr>
<tr>
<td>Fig. 3-17</td>
<td>Program blocks of collision detection and depth calculation for the peg-in-the hole insertion</td>
</tr>
</tbody>
</table>
Fig. 3-18 Depth calculation ................................................................. 45
Fig. 3-19 A virtual insertion data set (initial angles: \( \theta_x = 0 \) and \( \theta_z = 0.267035 \) rad).
(a) Positions; (b) Forces; (c) Torques; (d) Angles ................................ 48
Fig. 3-20 Another virtual insertion data set (initial angles: \( \theta_x = -0.01571 \) rad and \( \theta_z = 0.298451 \) rad).
(a) Positions; (b) Forces; (c) Torques; (d) Angles ... 49
Fig. 3-21 A third virtual insertion data set (initial angles: \( \theta_x = -0.00135 \) rad and \( \theta_z = -0.02709 \) rad).
(a) Positions; (b) Forces; (c) Torques; (d) Actions ... 50
Fig. 3-22 Haptic rendering of the hole using cylindrical approach ............... 51
Fig. 3-23 Previous and current points in cylindrical approach.......................... 52
Fig. 3-24 Data obtained from the virtual environment using cylindrical method.
(a) Positions; (b) Forces; (c) Torques; ........................................... 53
Fig. 4-1 Relationship between the skill transfer steps and the system modules. 59
Fig. 4-2 Basic structure of perception module ............................................ 62
Fig. 4-3 Forces in a successful insertion in the virtual environment .......... 66
Fig. 4-4 Accommodation and generation of RBFUs .................................... 69
Fig. 4-5 An example of a one-dimension dynamic system ......................... 69
Fig. 4-6 Combination of sequences illustrated in 2 dimensions (Assuming that
the longer displacement takes longer time) ........................................ 70
Fig. 4-7 Manipulator task planner module ............................................. 71
Fig. 5-1 Skill learning paradigm: (a) Learning through SVM for Type 1 skills;
(b) Learning through Lwpr for Type 2 skills ..................................... 78
Fig. 5-2 Skill learning process ................................................................. 79
Fig. 5-3 Decision margin for an oriented hyper-plane classifier .................... 82
Fig. 5-4 One-to-one classifier ................................................................. 88
Fig. 5-5 Forces/torque of jamming states ............................................... 93
Fig. 5-6 Position of peg at jamming states (measured in the number of pulses). 93
Fig. 5-7 Actions at jamming states .......................................................... 94
Fig. 5-8 Two models of SVM classification ............................................. 96
Fig. 5-9 Performance of Model x ............................................................ 96
Fig. 5-10 Performance of Model y ........................................................... 96
Fig. 5-11 Cross correlation between original steps and output steps of Model x.. 97
Fig. 5-12 Cross correlation between original steps and output steps of Model y.. 97
Fig. 5-13 Regional of validity of a linear model and its approximation bias \( \theta_k \)... 100
Fig. 5-14 Lwpr algorithm model ............................................................. 105
Fig. 5-15 First learning results (normalised) ............................................ 105
Fig. 5-16 Cross correlation between original steps and output steps of the first
leaning results ................................................................................. 106
Fig. 5-17  Second learning results with the model updated by the new data (normalised) ......................................................................................... 106

Fig. 5-18  Cross correlation between original steps and output steps of the second learning results with the model updated by the new data .......... 107

Fig. 6-1  Real rig................................................................................................. 109

Fig. 6-2  Block diagram of the experimental rig ................................................. 109

Fig. 6-3  Definition of turning directions ........................................................... 112

Fig. 6-4  Displacement between the axis of the transducer and that of the hole 113

Fig. 6-5  Jamming example 1.............................................................................. 116

Fig. 6-6  Jamming example 2.............................................................................. 116

Fig. 6-7  Example 1 under Condition 2, (a) Forces/torque; (b) Actions; (c) Position................................................................................................. 119

Fig. 6-8  Example 2 under Condition 2. (a) Forces/torque; (b) Actions; (c) Position................................................................................................. 120

Fig. 6-9  Example 1 in Case 2 (initial angles: X-axis Misalignment = 4.5° and Y-axis Misalignment = -4.5°), (a) Forces/torque; (b) Actions. .............. 123

Fig. 6-10 Example 2 in Case 2 (initial angles: X-axis Misalignment = -4.5° and Y-axis Misalignment = -4.5°), (a) Forces/torque; (b) Actions. .............. 124

Fig. 6-11 Example 3 in Case 2 (initial angles: X-axis Misalignment = -4.5° and Y-axis Misalignment = 4.5°), (a) Forces/torque; (b) Actions. .............. 125

Fig. 6-12 Example 4 in Case 2 (initial angles: X-axis Misalignment = 4.5° and Y-axis Misalignment = 4.5°). (a) Forces/torque; (b) Actions. .............. 126

Fig. 6-13  Forces/torque of jamming states .......................................................... 127

Fig. 6-14  Positions of the jamming states.......................................................... 127

Fig. 6-15  Angles of the jamming states .............................................................. 128

Fig. 6-16 Jamming states ..................................................................................... 128

Fig. 6-17  Forces/torque (Example 1 of Case 3).................................................... 129

Fig. 6-18  Position (Example 1 of Case 3)........................................................... 129

Fig. 6-19  Actions (Example 1 of Case 3)............................................................ 130

Fig. 6-20  Forces/torque (Example 2 of Case 3).................................................... 130

Fig. 6-21  Positions (Example 2 of Case 3)......................................................... 131

Fig. 6-22  Actions (Example 2 of Case 3)............................................................. 131

Fig. A-1  Sanyo Denki 2 phase stepper motor [Farnell(online)]......................... 164

Fig. A-2  Connections for stepper motors ............................................................ 165

Fig. A-3  Gecko microstep drive [Gecko(online)].............................................. 166

Fig. A-4  Connections for stepper motor drives (Dashed lines are for the step drive which drives the motor mounted on the X-axis of the hole)............. 167

Fig. A-5  MTE-2250-AMACN DC servo motor W/Encoder [Baldor(online)] . 168
Fig. A-6  DC Servo Controller (TSD-050-05-1-I) [Baldor (online)] .................. 169
Fig. A-7  Connection of TSD with other parts .................................................... 170
Fig. A-8  DAQ-802 board .................................................................................... 171
Fig. A-9  D37 pin diagram ................................................................................. 171
Fig. A-10 DaqEZ professional software ............................................................... 172
Fig. A-11 Connection of HCTL-2016 decoder with other parts ......................... 173
LIST OF TABLES

Table 3-1  Scaling factors between the virtual environment and the real rig........ 46
Table 3-2  Units in the virtual and physical environments................................. 47
Table 4-1  Inputs, outputs and privilege for models of Types 1 and 2 skills ........ 67
Table 5-1  Training data ..................................................................................... 95
Table 5-2  Pairs of angles (measured in step) between the hole and the peg....... 105
Table 6-1  Typical forces and torques used in acquiring Type 1 & Type 2 skills 114
Table 6-2  Data from a successful insertion with Type 1 skills only ................... 114
Table 6-3  Corrective actions .............................................................................. 117
Table 6-4  First series of corrective actions (Example 2 of Situation 2)............. 121
Table 6-5  Second series of corrective actions (Example 2 of Situation 2) ....... 121
Table A-1  Functions of the pins in Fig. A-5(c)................................................ 168
<table>
<thead>
<tr>
<th>ABBREVIATION</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D</td>
<td>Three Dimension</td>
</tr>
<tr>
<td>ALVINN</td>
<td>Autonomous Land Vehicle In a Neural Network</td>
</tr>
<tr>
<td>ARMA</td>
<td>Auto Regressive Moving Average</td>
</tr>
<tr>
<td>ASMOD</td>
<td>Adaptive Spline Modelling of Observed Data</td>
</tr>
<tr>
<td>BDPP</td>
<td>Bidirectional Dynamic Path Planning</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer Aided Design</td>
</tr>
<tr>
<td>CMAC</td>
<td>Cerebellar Model Arithmetic Computer</td>
</tr>
<tr>
<td>CMU</td>
<td>Carnegie Mellon University</td>
</tr>
<tr>
<td>DAQ</td>
<td>Data Acquisition</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>DOF</td>
<td>Degree of Freedom</td>
</tr>
<tr>
<td>FA</td>
<td>Function Approximator</td>
</tr>
<tr>
<td>HMM</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>HSOL</td>
<td>Hierarchically Self-Organizing Learning</td>
</tr>
<tr>
<td>i.i.d.</td>
<td>Independently Identically Distributed</td>
</tr>
<tr>
<td>ISA</td>
<td>Industry Standard Architecture</td>
</tr>
<tr>
<td>KDS</td>
<td>Kinetic Data Structure</td>
</tr>
<tr>
<td>KL</td>
<td>Kullback-Leibler</td>
</tr>
<tr>
<td>LARTS</td>
<td>Language-Aided Robotic Teleoperation System</td>
</tr>
<tr>
<td>LWL</td>
<td>Locally Weighted learning</td>
</tr>
<tr>
<td>LWPLS</td>
<td>Locally Weighted Partial Least Squares Regression</td>
</tr>
<tr>
<td>LWPR</td>
<td>Local Weighted Projection Regression</td>
</tr>
<tr>
<td>LWR</td>
<td>Locally Weighted Regression</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-Layer Perceptron</td>
</tr>
<tr>
<td>MRCCN</td>
<td>Multi-resolution Radial basis Competitive and Cooperative Network</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network</td>
</tr>
<tr>
<td>NURBS</td>
<td>Non-Uniform Rational B-Splines</td>
</tr>
<tr>
<td>OBB</td>
<td>Oriented Bounding Box</td>
</tr>
<tr>
<td>p.d.f.</td>
<td>Probability Density Function</td>
</tr>
<tr>
<td>PbD</td>
<td>Programming by Demonstration</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>PLS</td>
<td>Partial Least Squares Regression</td>
</tr>
<tr>
<td>QP</td>
<td>Quadratic Programming</td>
</tr>
<tr>
<td>RBF</td>
<td>Radial Basis Function</td>
</tr>
<tr>
<td>RBFU</td>
<td>Radial Basis Function Units</td>
</tr>
<tr>
<td>RFWR</td>
<td>Receptive Field Weighted Regression</td>
</tr>
<tr>
<td>RMS</td>
<td>Root Mean Square</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SV</td>
<td>Support Vector</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>VR</td>
<td>Virtual Reality</td>
</tr>
<tr>
<td>VRML</td>
<td>Virtual Reality Modelling Language</td>
</tr>
<tr>
<td>WTK</td>
<td>WorldToolKit</td>
</tr>
</tbody>
</table>
CHAPTER 1   INTRODUCTION

1.1 Problem Statement
This thesis addresses the problem of efficient and simple generation of robotics skills to perform a constrained motion manipulation task which is an essential process in any realistic manipulation environment such as automatic assembly. In such applications, a grasped object is manipulated while in contact with the surrounding environment or other objects towards achieving a specific goal. Hence the manipulator, the object and the environment must collectively perform compliant motion along certain directions.

In compliant motion, a task is not usually well structured and there is uncertainty. A human operator deals with uncertainty in compliant motion and adapts to new circumstances mainly through kinaesthetic and tactile information received from the hands [Nguyen(thesis)]. Similarly, a robot performing compliant motion depends on sensory information measured from the environment for a successful manipulation. Sensory integration in robot programming has proved difficult and challenging. This has been a major obstacle in deploying robots in automatic assembly.

Through training and practice, a human operator develops skills to perform a task. A skill consists of actions that a person performs competently to achieve a goal. While skills are usually developed in the context of a particular task, they usually have generic elements which can be applied to other tasks whether in the same class or completely different. For example, the skill of opening a door applies to any type of door regardless of its size and type. Such skill can be however, used to open the lid of a box.

In this study, feasibility of skill acquisition by a robotic manipulator to perform a constrained motion manipulation task as an alternative approach to programming is
explored. Skill acquisition in human takes place in psychomotor domain and is best achieved by learning from another human being. For a robot, acquiring skills from a human operator has been a viable approach to pursue.

Direct transfer of skills from a human operator to a machine in an interactive environment has been studied by a number of research groups. Similar to the processes involved in supervised human learning, two critical issues need to be addressed in the transfer of skills from a human to a machine:

(a) The transfer of control strategies

(b) The evolution and enhancement of the control strategies and the manipulation processes as the task is repeated.

Handleman and Lane [Handleman96] have provided some initial work on (a) and have suggested a knowledge-base “tell” approach to describe the task to be carried out by the robot and the corrective control measures to be taken up. The task is defined by a rule-based goal directed strategy. The proposed method has been verified only through computer simulation for a typical peg-in-hole insertion problem. The development of the rule-based system has been intuitive and rather complicated. The developed rules are very much context based and have to be built from scratch for any new application.

The same authors have also developed some strategies to achieve the goals described in (b) [Handleman90] [Hendleman92a] [Hendleman92b], using integrating declarative rule-based systems and reflexive neural networks in highly adaptive systems.

The focus of this work is on developing a more systematic and simpler method for the transfer of skills from a human operator to a machine. In this approach, the instructor demonstrates the manipulation task in a haptic rendered virtual
environment using a haptic device. The use of the virtual environment will simplify the process as the training data will be directly extracted from the haptic system [Mussa-Ivaldi85]. The extraction of the rules and the development of the manipulation skills database take place systematically through a learning algorithm as the operation of the instructor is observed.

The approach pursued in this work has some similarities to the work reported in the literature on learning from examples, which studies feasible methodologies to acquire knowledge. It then attempts to structure, analyse, and formalise that knowledge to extract rules in building an expert, knowledge-base system. As an example, in the work conducted by Pitas et al [Pitas92] a learning approach using a minimum entropy criterion is developed to obtain rules that fit a set of examples and counter examples. The parameters of the rules can also be varied to minimise the entropy. The major difference is that in this work manipulation skills performed in psychomotor domain are transferred from human to machine, rather than the transfer of knowledge in the cognitive domain.

1.2 Overall Approach

A manipulation skill is the ability to transfer, physically transform or mate a part with another part. A specific manipulation skill consists of a number of basic skills that when sequenced and integrated can achieve the desired manipulation outcome.

The manipulation task ($M_s$) is applied to the part by the human operator through an action $u^h(t)$, transferring the part from an initial state of $x^h(t_i)$ to a final state of $x^h(t_f)$. The control action command $u^h$, provides position and force/torque settings. Depending on the type of manipulation, the state vector can represent position, orientation, and dimension of the part or its contact forces/torques with the environment. The measured state variables at any instant of time $t$ will represent the
output of the manipulation system $y^h(t)$. The variables $x$, $u$ and $y$ are vectors. The superscripts ‘$h$’ and ‘$m$’ of $y$ and $u$ mean ‘haptic virtual environment’ and ‘manipulator’ respectively.

Fig. 1-1 describes the structure of the system under study in this research. As shown in this figure the robotics manipulator mimics the behaviour of the human operator by acquiring the skills and producing the machine control action $u^m(t)$ from $y^h(t)$ as illustrated in Fig. 1-1. The system is designed to closely emulate and facilitate the relevant stages of human motor learning taxonomy for a robotics manipulator as described below:

(a) The human operator performs the manipulation task in a virtual environment using a haptic device. The haptic device provides the operator with contact forces and torques similar to those in a real life operation.

(b) The information produced in the virtual environment, $y^h(t)$, is used by the perception module to identify the basic skills and functions employed in the operation and, to extract the algorithm sequencing the applied skills. This is perception stage of the taxonomy.

(c) The information produced in (b) is passed to the Manipulator Task Planner to be translated into position/force trajectories and associated control algorithms for the robotics manipulator. Initially $u^m$ is generated based on the information received from the perception module, the output of the machine manipulation system $y^m(t)$, and prior knowledge about the task. The performance of the manipulator under $u^m$ is then compared with the expected behaviour. The manipulator trajectory and $u^m$ are adjusted according to the error to produce a behaviour as close as possible to the manipulation performed by the human.
(d) After satisfactory imitation, information from the learning module will be taken into account to calculate $u^m$. The learning module performs various optimisation processes to enhance the performance.

![Diagram of the system](image)

Fig. 1-1 Overall model of the system

Such a system will be most effective when the perception and learning modules are generic. The manipulation virtual environment will be dependent on the application and the task planner will be dependent on the manipulator employed.

1.3 Aim and Contributions of Thesis

The primary aim of the work has been to explore the feasibility of a practically achievable transfer of physically constrained manipulation skills from a human operator to a manipulator, through a simple show-based system utilising a haptic rendered virtual training environment.

The work has made the following contributions:

(a) A comprehensive and critical review of the historically significant and recent projects related to robot programming has been carried out. This represents a survey of work in this field while setting the background for any new initiative.
A technically valid haptic rendered model of the peg-in-hole insertion process, representing a typical assembly process, has been developed in a virtual environment. The force and position data produced from the manipulation of the model have been successfully linked to construction of skill acquisition algorithm and eventually the actual physical assembly process. This clearly demonstrates that haptic rendered models can be fine tuned to reflect the characteristics of a constrained motion manipulation task.

An important and unique contribution of the thesis is the development of a hierarchical paradigm for acquisition of manipulation skills from a haptic rendered virtual manipulation model. This methodology, though developed based on the peg-in-hole insertion, has potential for deployment in other applications.

The effectiveness of the approach has been validated through carefully designed experimental work.

The thesis overall demonstrates that machine intelligence and machine learning can be employed to replicate the significant stages of human psychomotor learning such as perception, planning and imitation learning.

1.4 Thesis Organization

The content of the thesis is structured as follows. A literature review of the previous work on programming of a robotics manipulator is carried out in Chapter 2. In this study the evolution of the methodologies and advances made particularly in the area of constrained motion manipulation are highlighted.

In Chapter 3, graphic and haptic rendered modelling of the peg-in-hole insertion in the virtual environment is described. The virtual haptic rendered model is used in the virtual manipulation of the task. The algorithm used to calculate collision between
the peg and the hole and the generated forces and torques is also described. Some examples of the forces and torques generated during virtual manipulation are provided.

Chapter 4 provides an overview of the process through which manipulation skills are acquired from the manipulation performed by the operator in a haptic rendered virtual environment and transferred to the physical manipulation system. Initially the overall process is described. Then three modules developed in this work to carry out the acquisition and the transfer of the skills including the perception module, manipulator task planner module and learning module are described.

The focus of Chapter 5 is on manipulation skill acquisition. Some classical learning methods are discussed. The methods employed in this thesis including Incremental Multi-class Support Vector Machine (off-line), and Locally Weighted Projection Regression (on-line) are introduced.

The developed methodology is validated in Chapter 6. Initially the experimental rig and the experimental set up are introduced. Then experiments carried out to evaluate the performance of the system, in particular in comparison with other methodologies, are described. It will be shown that the proposed method works successfully if sufficient training data is used in the learning algorithm. The experimental study also reveals that the outcomes depend not only on the algorithm employed but also experimental settings and hardware design of the system.

Chapter 7 provides conclusions on the work and offers some recommendations for further research.
CHAPTER 2  BACKGROUND

2.1 Introduction
The overall aim of this work is to explore the development of a simple and efficient method of programming a robotics manipulator through transfer of manipulation skills from human to machine through demonstration in a haptic rendered virtual environment. It also represents one of the latest paradigms proposed for the programming of a robotic manipulator. In order to highlight the significance and contribution of the work, the historically significant and recent works related to robot programming are reviewed in this chapter.

The project is multidisciplinary and represents the synergy and integration of a number of emerging technologies such as haptic rendering, virtual modelling, and motor skill acquisition. The detailed literature review associated with each topic will be carried out in its corresponding chapter. An overview of these approaches is provided in this chapter in the context of robot programming.

2.2 Overview of Robot Programming
The programming of a robotic manipulator has been an important area of research and development since the inception of robotics. Autonomous manipulators are capable of following a particular type of behaviour prescribed by the user (see Fig. 2-1 [Ahmad(online)]). For simple pick and place tasks, the points of action can be easily programmed into the robot in terms of translation and rotation that each joint should perform. Programming the robot to perform complex tasks is, however, very different and demands a different level of interaction between the user and the robot.
Fig. 2-1 System enabling a robot to follow a behaviour prescribed by the user [Ahmad(online)]

Historically, as the complexity of the robotics applications has increased, new methods for their programming have also been introduced. These methods can be categorised in different ways. For example, a programming language can instruct a robot to drive its joints to specific locations or achieve a particular task at its tool tip. Accordingly, the programming language used can be referred to as a joint level or a task level language respectively [Pettinaro96] [Gray89]. One typical categorization is illustrated in Fig. 2-2 [Lawson(on-line)].

In this chapter, the robotics programming languages are reviewed chronologically and categorized according to the complexity of the paradigm they are based on, as follows:

- Text programming,
- Off-line simulation-based programming,
- Inductive learning,
Teaching by guiding and
Teaching by showing.

```
| Task level          | "assemble a PCB of type A"
|---------------------|--------------------------
| Object level        | "pick up IC No. 20"
| Manipulator level   | "move to position X"
| Joint level         | "drive joint 3 by 30 degrees"
```

Fig. 2-2  Levels of robot programming [Lawson(on-line)]

### 2.3 Text Programming

While visual programming systems such as LabView [labview(online)] and ToonTalk [ToonTalk(online)] (see Fig. 2-3 for examples) are based on the creation of an application using the mouse, graphical symbols, algorithms and visual structures, text programming (see Fig. 2-4 for examples) is based on entering keywords in a text editor. Text programming can be applied to complex applications, but the development time is long and special skills and much effort is required to produce a complete program. This has resulted in the development of task-level robot languages (see Fig. 2-2 for examples) [Perez77] [Matsubara85].

Usually, a robot programming language defines robot motion at joint and manipulator levels. Task-level programming enables the user to specify the desired goals of the tasks without defining every movement of the robot in detail. The task planner will express each task in terms of necessary manipulator motions and actions. This scheme requires the system to have the ability to perform many planning tasks automatically. In addition, task-level programming tools require a
great deal of information about the workcell, the robots, the objects, the initial state of the environment and the final goal to reach, which can be extremely tedious and time consuming.

At this point the code is:

```
append(Arg1, Arg2, Arg3) :-
  Arg1 = [X1 | X2],
  X1 = a,
  X2 = [X3 | X4],
  X3 = b,
  X4 = [],
  Arg2 = [X5 | X6],
  X5 = c,
  X6 = [],
  Arg3 = $bird(Answer) | Floor1 = Arg1
```

The code for the recursive base case after vacuuming up the “c” becomes:

```
append(Arg1, Arg2, Arg3) :-
  Arg1 = [],
  Arg3 = $bird(Answer) | Answer = Arg2 /* from thought bubble box*/

// set off bomb(step M)
```

Fig. 2-3 Visual programming -- ToonTalk Append Example [Visual(online)]

Fig. 2-4 Text Programming: A simple program to count from 1 to 10 [Mechatronics(online)]
2.4 Off-Line Simulation-Based Programming

Off-line simulation-based methods usually integrate text-programming and model-based motion planners in one common platform [Derby82] [Naylor87]. Model-based motion planners can automatically generate motions from a CAD model and therefore the tedious task of teaching the robot is avoided. An example of model-based motion planners -- AMROSE [AMROSE(online)] motion planner -- is shown in Fig. 2-5 [MOBARC(online)]. In principle, the off-line model-based motion planner thus enables one-of-a-kind production.

![Diagram of AMROSE motion plan net](image)

**Fig. 2-5** Basic architecture of the model based off-line system supplied by AMROSE [MOBARC(online)]

Off-line simulation-based programming is powerful but requires special hardware and a complete description of the real world, both of which are costly. Some off-line programming environments such as IGRIP (a trademark of Deneb, Inc.) (Fig. 2-6), CimStation (a trademark of Adept Technology, Inc.), and RobCad (a trademark of Technomatic Technologies Limited) [Wittenberg95] offer graphical simulation platforms, in which the programming and execution processes are illustrated using models of real objects [Rembold93]. Thus, the programmer is required only to learn the simulation language. In off-line programming, libraries of pre-defined high-level
commands for certain types of applications can be built, the kinematics feasibility of a move can be assessed, and the cycle time for a sequence of movements can be determined.

Fig. 2-6  Screenshot of IGRIP, by Deneb [Lawson(on-line)]

The off-line programming environments can also provide a set of primitives commonly used by various robot vendors, and produce a sequence of robot manipulator language primitives such as “move” or “open gripper”, However, the current state-of-the-art off-line systems do not address the issue of sensor-guided robot actions. They are also mostly limited to kinematics or dynamics simulation of a robot, with no provisions for advanced reasoning functionality, nor flexibility in the tasks.

2.5 Inductive Learning

In inductive learning, a robot arm masters appropriate motion and sensing strategies through trial and error [Dufay84]. This is an effective method when used to refine
other programming methods. It is not suitable on its own for very complex tasks. The inductive learning is also called learning from examples, which is one of the most important approaches among the learning paradigms. In this approach, a learning approach is employed to acquire or to generalize a “concept” or decision rules from a set of training examples [Bhavsar00].

The performance of inductive learning methods can be measured by the learning curve which shows prediction accuracy as a function of the number of observed examples. Inductive learning can be seen as a form of optimization or search method reducing the ‘badness’ of the hypothesis. It can be viewed as learning a function by selecting a function (the hypothesis) from a set of functions (the hypothesis class).

Examples of Inductive Learning [Dumais(online)] are Find Similar (a variant of Rocchio’s method for relevance feedback) [Rocchio71], Decision Trees [Chickering97], Naïve Bayes, Bayes Nets [Heckerman95], and Support Vector Machines (SVM) [Vapnik95].

2.6 Teaching by Guiding

In “teaching by guiding” a human operator drives a robotics arm in the real world to perform the task while the characteristics of the motion are recorded. In spite of its simplicity, the method is not generic, flexible or robust, and is not applicable to complex tasks. It cannot accommodate extensive sensory interaction and can be dangerous for the operator. For example, in long distance or space telerobotics, the transmission time delay up to a few seconds may cause instability in the system [Ferrel66]. In another example of teaching-by-guiding system, human intention is inferred from tactile gestures measured through force sensors mounted on the robot [Voyles95]. This method is not flexible since it requires modification of the robot
manipulator, ie. the operator physically nudges the robot into the desired trajectory shape with his or her hands.

The ultimate method of teaching by guiding is direct telemanipulation. Some of the previous work in this area falls between the two extremes of a fully autonomous robot arms and direct telemanipulation. In such applications, the human can specify the desired robot arm motion at a higher level of abstraction without having to deal with some of the mapping and communication issues of direct telemanipulation. In the teleautonomous system developed by Conway, Volz et al. [Conway87], the robot acts autonomously, but the human is able to make adjustments during run time if the situation warrants. In [Graves95], decisions are received and integrated from multiple sources, such as a human and an autonomous controller. The weight placed upon each decision source can be modified dynamically according to the mode of control, such as shared, teleoperative, and supervisory. Thus the system can benefit from the human control while still utilizing the robot capabilities.

There have been a number of attempts to overcome some of the shortcomings of “teaching by guiding” approach. Summers and Grossman [Summers84] embedded the collection of the sensory information and interaction with the operator in the task instruction procedure. Asada and Assari [Asada88] extracted the control rules to perform a particular assembly motion from the position and force data generated during operation of a human operator. The control strategy employed in this approach follows the relationship between the force data and the tool motion demonstrated by the human operator (Fig. 2-7).

Sator and Hirai [Sato87] integrated direct teaching with task level languages through master-slave manipulators. This language-aided robotic teleoperation system (LARTS) has two sets of teleoperation languages -- a text/menu commander as a
symbolic communication channel to the operator, and a master-slave manipulator as an analog communication channel to the operator (Fig. 2-8). The force feedback instability due to transmission delays has been overcome by using a virtual slave arm [Kotoku92]. This implies that the slave environment is virtually constructed in the computer to solve the time delay problem caused by long distance telemanipulation (Fig. 2-9). The simulator can display simulated arm on the graphic display and estimate the interaction forces between the arm and the environment without time delay. With the help of this virtual visual and kinaesthetic information, the operator will be able to perform the tasks without feeling the delayed feedbacks.

![Diagram](image1.png)

Fig. 2-7 Direct teaching of tool manipulation skills (a) Measure of hand motion; (b) Replication of human expert; [Asada88]

![Diagram](image2.png)

Fig. 2-8 Language-aided robotic teleoperation system [Sato87]
2.7 Teaching by Showing

The concept of “teaching by showing” has been another extension of “teaching by guiding”, in which a robotics system learns a particular task by watching a human operator performing it. The learning methodologies were initially developed for computer scene understanding [Suji77] and automatic perception of actions [Thibadeau86] [Kuniyoshi87]. The best method would seem to be one which allows the user to make natural human motions that can be mapped intelligently to robot commands.

Some recent developments have significantly advanced the “teaching by showing” approach in programming a robotics manipulator. Ikeuchi and Sehiro developed a system that could extract fine motion sequence from transitions of face contact states obtained by a range sensor [Ikeuchi91]. Haas [Haas91] integrated a symbolic recogniser and play back module using a visual servo for two-dimensional pick and

Fig. 2-9 Using a virtual slave arm to overcome force feedback instabilities due to transmission delays [Kotoku92]
place operation. Yamada and Uchiyama conducted a study to determine essential
features of human physical skills based on multi-sensory data and the possibility of
transferring them to robots by focusing on two tasks of crank rotation and side
matching [Sato97].

Direct transfer of skills from a human operator to a machine in an interactive
environment has been the next stage in the programming and training of a robotics
system. In the field of mobile robots, Pomerleau used a 3 layer perceptron network to
control the CMU (Carnegie Mellon University) ALVINN (Autonomous Land
Vehicle In a Neural Network) autonomous vehicle [Pomerleau93]. Grudic and
Lawrence used an approximation method as a means for creating the robot’s
mapping from sensor inputs to actuator outputs in transfer of skills to a mobile robot
[Grudic96].

In acquisition of manipulation skills, particularly in constrained motion, the work
carried out by Kaiser and Dillman [Kaiser96] is of significance. The work proposes a
general approach to the acquisition of sensor-based robot skills from human
demonstration. An adaptation method is also proposed to optimise the operation with
respect to the manipulator. The method is validated for two manipulation tasks of the
peg-in-hole insertion and opening a door.

Handleman and Lane [Handleman96] have carried out some preliminary work on a
knowledge-based “tell” approach to describe the task to be carried out by the robot
and the corrective control measures to be taken up. The task is defined by a rule-
based goal directed strategy. The proposed method has been verified through
computer simulation only for a typical peg-in-hole insertion problem. The
development of the rule-based system has been intuitive and rather complicated. The
developed rules are very much context based and have to be built from scratch for any new application.

Acquisition of force-based low-level assembly skills from human demonstration has been studied by Skubic [Skubic98]. The contact information during demonstration is derived from force sensor patterns rather than position and geometric data [Skubic00]. Force-based discrete states are used to model qualitatively the contact made with the environment. A change in state triggers a new control command to the robot. The focus of the work has been primarily on the identification of contact formations from force sensor patterns. The main difference between guiding and showing is that during guiding the human operator teleoperates or guides the robotic manipulator to do the task while during showing the human operator does the task first and then the robotic manipulator learns human skills from the recorded data to do the task.

In many cases, the desired position of an object is quite important to be known by a robot. By utilizing a vision system, Kang and Ikeuchi [Kang94] taught grasping to a robot through demonstrating the task by a human operator performing it. The human fingers are equivalent to robot fingers in terms of their functionality. An algorithm produces a kinematically feasible manipulator grasp trajectory by analysing the acquired visual information. Kuniyoshi et al developed a robotics system that could learn reusable task plans in real time by watching a human performing the assembly tasks [Kuniyoshi94]. The method was based on visual recognition and analysis of human action sequences. The effectiveness of the method was demonstrated for a block assembly task.


2.8 Teaching by Demonstration through A Virtual Environment

As discussed in previous sections, text programming has been popular and effective in programming simple robotics tasks. The Robotics text-based languages are usually dependent on the particular manipulator used [Blume83] [Rehg97]. Simulation based programming is a powerful approach and is more suited to sensor-intensive tasks. It, however, requires expensive special purpose hardware together with either a custom environment, or an accurate description of the real world. Inductive learning needs the least interference from the user, but it is not suitable for complex tasks. The approach “teaching by guiding” requires no additional hardware, and little programming. However, there may be safety issues during the use of real manipulators and mistakes in the movements can be costly. In “teaching by showing”, a robot learns a reusable task plan by observing the task performed by a human operator. Human demonstration can be performed by guiding a robot with a force sensor to do a task [Wang96] or by using a teach device which acquires the sensorimotion of the human [Koeppe96]. However, sometimes a vision system, a teach device or a local copy of a robot and the robot’s environment is not available or not easy to construct. In this case the user can demonstrate the task in a virtual setting. Robot programming can be improved by using a virtual robot and a virtual environment.

Fraunhofer IPA in Germany [Strommer93] enabled the programmer interaction being mediated by Virtual Reality (VR) I/O devices (eg. sensing glove). The programmer can navigate and look at the scene from any angle, and observe details that may not be visible in real life, and specify a trajectory very easily. This graphical user interface allows the user to specify the dynamic behaviour of components such as their paths, accelerations, velocities, interpolations, etc. The code is automatically
stored using a special-purpose toolkit called “VR4” [Flaig96]. Once the program is completed, it is downloaded to a real robot connected to the same VR engine to execute the same task. Feedback from the sensors on the real robot during the real-life task is then used to fine-tune the program. Kunii and Hashimoto [Kunii97] use Dynamic Force Simulator from [Hashimoto93] a human to interact with a virtual object. A robot learns the task from the human motion. Learning methods involving neural nets and radial basis functions are used to compensate for small errors.

Bruns [Bruns98] [Bruns99] combined the virtual manipulation with the physical demonstration by allowing the users to wear an instrumented glove to interact with a concrete object which has a corresponding virtual object in a virtual environment. The virtual objects often represent conveyors and objects in a manufacturing plant, or stages in a process planning diagram, while the physical objects provide real feedback to the user.

VR can provide immersibility, interactivity, and dynamics [Flaig94]. Based on programming by demonstration (PbD) [Asada89] [Delson94] [Kaiser95] and programming by watching [Kuniyoshi94] [Lee94] [Suji77] [Thibadeau86] [Kuniyoshi87], virtual environments have been used for demonstration in skill learning and transfer for safety, convenience, fast programme modification and testing, and inexpensive user training [Troy95]. Virtual environments with haptic feedback, which can provide not only position but also contact force and torque data, have been used for skill acquisition or skill transfer [Boud97] [Kunii96] [Yokokohji98]. With the help of the latest developments in virtual reality and computer simulation, the human operators can demonstrate the skills in a virtual environment with tactile sensing (haptics). The robotics manipulator mimics the behaviour of the human operator by acquiring the skills and producing the machine
control action. In general, the skills should be task centred, not robot centred. Yokokohji et al [Yokokohji96] [Yokokohji98] provided several methods for effective hand-eye coordination training via virtual environment based on the idea of “record-and-replay”. However, the skill transfer is only from human to human or from one (expert) to others (trainee) via virtual environment. In [Kunii96], a Dynamic Force Simulator is used to simulate object dynamics, contact model and friction characteristics of the human hand interacting with the object in virtual reality. Recently, more work on the application of haptic rendered VR in human-to-robot skill transfer has been reported in the literature [Brunner95] [Voyles97] [Bayazit99] [Kawasaki00]. The scope, approach and application of the proposed methods vary significantly. The feedback sensory information is provided by gloves, specific built haptic devices or vision systems. Overall, it is difficult to generalize the methods or apply one approach to another situation.

A new paradigm for programming of robotics manipulator to perform complex constrained motion tasks is being studied in this thesis. The teaching of the manipulation skills to the machine starts by demonstrating those skills in a haptic-rendered virtual environment. Position and contact force and torque data directly extracted from the haptic virtual environment combined with a priori knowledge about the task is used to identify and learn the skills in the newly demonstrated tasks and then to reproduce them in the robotics system. This is also a special machine training or machine learning approach which takes advantage of recent developments in virtual reality and computer simulation.

2.9 Chapter Summary

In this chapter a review of various programming techniques developed for a robotic manipulator was carried out. Programming a robot to perform a constrained motion
task is quite challenging. In this work the task is demonstrated in a haptic rendered virtual environment.
CHAPTER 3  VIRTUAL MANIPULATION ENVIRONMENT

3.1 Introduction

The data used to acquire basic manipulation skills is generated through a haptic-rendered virtual environment. This approach offers a number of advantages compared to other methods of obtaining training data including:

(a) The training data (e.g. velocities, angles, positions, forces and torque) can be extracted and recorded directly which simplifies the data collection process. [Mussa-Ivaldi et al., 1985].

(b) The environment can be easily modified and changed as the manipulation process and its requirements are changed.

(c) The risk of breakdown and breakage of the system is very low.

(d) Dangerous and costly environments can be easily constructed and simulated.

(e) A user-friendly environment for the human operator can be developed.

It is, however, essential that the developed virtual environment accurately reflects the real assembly process. This may not be always possible. For example, the maximum force generated in a haptic rendered environment may be limited compared to the actual forces generated in a real assembly process.

Such shortcomings can be overcome through further generalization of the skill learning methods and enhancement of the skills database by employing complimentary on-line methods such as on-line incremental learning.

A haptic rendered virtual environment can be manipulated in a stable manner in real time using a haptic interface. The haptic interface used in this research is a 3 degree-of-freedom generic device called Phantom manufactured by Sensable. It allows users to directly interact with digital objects as they do in the real world. The software package used, GHOST® SDK, supplied with Phantom, can handle complex
computations and allows developers to deal with simple, high-level objects and physical properties like location, mass, friction and stiffness [SensAble(online)].

In order to produce a haptic rendered virtual environment, a number of problems such as collision detection, force/torque and dynamics generation, geometric modelling of haptic objects and graphic rendering should be resolved [Hollerbach98]. This chapter describes the methodologies used to address these issues in the development of a haptic rendered VR models for the peg-in-hole insertion which represents a typical manipulation task in assembly.

The previous work conducted by other groups on different stages of development including collision detection, force/torque and dynamics generation, geometric modelling of haptic object, and graphic rendering [Hollerbach98] will be also reviewed.

3.2 Virtual Peg-in-Hole Insertion Environment

The peg-in-hole insertion with tight fit is a classic example demonstrating a typical assembly task. It has been used extensively in the study of issues associated with automatic assembly.

The real experimental peg and the hole are both round. The radius of the peg is 10mm and that of the hole is 10.05mm. The hole has two degrees of freedom (the pitch and yaw angles) controlled by two stepper motors. The peg has one degree of freedom of translation along the axis of the peg and it is controlled by a DC servo motor. The virtual environment simulates this experimental rig.

Graphic and geometric models are the key components of the virtual model. The former has been called visualisation model or visual rendering model whereas the latter has also been referred to in the literature as haptic geometric model. The graphic and geometric models of an object should coincide in the virtual space.
While graphic model displays the visual properties of an object surface, the geometric model represents the haptic characteristics of its surface.

The rendered graphic objects can also provide geometrical information for collision detection and force calculation using standard methods such as NURBS (Non-Uniform Rational B-Splines) [Thompson97]. In fact, most of the graphic rendering methods (including the NURBS) can render powerful graphics for both vision feedback and collision detection/force calculation. Haptic update rate must be around 1 KHz to maintain stable force interactions while graphic update rate must be between 20-30Hz to meet the real-time constraints (Fig. 3-1).

![Fig. 3-1 Graphic or geometric model used for both visual rendering and haptic rendering [Mendoza(online)]](image)

3.2.1 Graphic Model

Graphic modelling is the first step in the process of building the virtual environment. It provides vision feedback to the user. The final version of the graphic model of the peg-in-hole insertion virtual environment is constructed using OpenGL [SensAble(online)] supplied with the Ghost SDK [SensAble(online)]. In this work,
the information provided by the graphic model is not used for collision detection and force calculation.

In the first implementation of the peg-in-hole insertion virtual model, WTK (WorldToolKit) [Millersville(online)] was used to build the graphic model and to calculate collision detection and force generated during the manipulation. WTK is a cross-platform software development system used for building high-performance, real-time, integrated 3D applications for scientific and commercial use [Sense8(online)]. It generates objects based on polygons and has been used previously in conjunction with Phantom and Ghost SDK in many applications such as medical suturing and operation.

The graphic and geometric models of the virtual peg-in-hole insertion developed based on WTK proved problematic. The force feedback produced by the virtual model exhibited oscillation due to the complexity of the routines used and the intensity of the computing. Hence, this approach was abandoned.

A haptic object or geometric model created through Ghost SDK can be felt but not seen, i.e., it is touchable but invisible. Ghost SDK uses OpenGL to render visible graphic objects coinciding with corresponding haptic objects and to transform graphic objects accordingly when the haptic objects are transformed. The force calculation and generation are only based on the invisible haptic objects (e.g. geometric model) not on the visible graphic objects (e.g. graphic model). In this case, the graphic rendering is just for the visual convenience. Without the graphic rendering, the haptic objects still can be touched but it is similar to touching objects with closed eyes.

OpenGL uses quadrangle polygons to build the graphic models. The quadrangles used for building the graphics of the peg and the hole are shown in Fig. 3-2. If the
direction of the normal of the polygons is opposite to the viewer, these polygons are invisible, otherwise they are visible.

The normal of a quadrangle is determined by applying the right hand rule to the quadrangle. The order of the quadrangle vertexes at the time of its creation determines the direction along which the four fingers of the right hand should be curled. Then the thumb indicates the direction of the normal to the quadrangle.

For example in Fig. 3-2, if the vertexes of the shaded quadrangle are created in the order of 1-2-3-4, the normal is along \( A \) direction; whereas for 1-4-3-2 the normal is along \( B \) direction.

The quadrangles created for the inner wall of the front half of the hole are invisible since their normal vectors are pointed to the centre axis of the hole and hence they are in the opposite direction of the viewer.

![Graphic model of the peg and hole](image)

(a) (b)

Fig. 3-2  Graphic polygons in the peg and the hole. (a) Graphic model of the peg (8 sides are assumed); (b) Graphic model of the hole.

The virtual environment is discrete as at each sampling interval, the graphic model is redisplayed and the forces and torque are recalculated and re-rendered. In order to reduce computing time in each sampling interval, only the top of the hole and the visible parts of the inner wall are created graphically in the virtual environment. The
virtual environment and its coordinates are shown in Fig. 3-3(a). The coordinate frame attached to the virtual environment is not the same as that of the real rig (see Fig. 3-3(b) and Fig. 6-1) relative to the configuration of the peg and the hole. Hence, the training data from the virtual environment is transformed to match the coordinate frame of the physical rig.

![Fig. 3-3 Peg-in-hole insertion virtual environment and coordinates of the real rig](image)

3.2.2 Haptic Rendered Model

Haptic rendered model provides haptic or force/torque feedback to the user. The most basic and common functions of haptic rendering algorithms include modelling objects with facets (e.g. triangle meshes), and generating forces normal and tangent to the plane, respectively proportional to penetration (stiffness) and velocity (damping) (see Fig. 3-4 (a) and (b) ). This is not different from generating the haptic rendered model from the geometric model by setting the tactile and force feedback simulation parameters for each virtual plane in the geometric model. In another approach, the virtual prototype logical models are constructed from existing or new logical model components, called virtual components [Salmela(online)].
The maximum stiffness of any virtual object is limited by the inherent mechanical compliance of haptic interface devices. Thus, the user’s contact point often penetrates the simulated object volume to a greater distance than would be possible in real life. This can lead to ambiguity in identifying the penetration plane and consequently the magnitude and the direction of the reactive force (see Fig. 3-4 (c)) [Salisbury95].

Collision detection in a virtual environment with many facets can be too computationally intensive for real time. This results in reduced force computational rates and increased latency. Force computational rate is the frequency at which the latest force or haptic information is calculated. Latency is the time required to update haptic rendering. As a result, hard surfaces in the virtual environment may feel soft and the system may become unstable. This was the case when WTK was used to calculate the collision detection and haptic rendering facets of the peg and the hole.

![Fig. 3-4 Facet, force normal to the plane and ambiguity. (a) Facet; (b) force normal to the plane; (c) ambiguity.](image)

A number of methodologies have been developed in the literature to overcome the problems mentioned above. In the god-object method [Salisbury95], a virtual contact point is maintained on the surface of the object during its virtual manipulation. The location of this point removes any ambiguity on which the force vector should be applied to the user.
In order to provide a smooth feeling for complex surfaces that are composed of many smaller, planar surfaces, the surface can be force-shaded \cite{Salisbury95} \cite{Ruspini97}. Force Shading permits the mapping of a shape or texture onto a polygon, so that they may be used in haptic rendering in the same way that texture mapping and colour shading are used in graphics rendering \cite{MIT(online)}. Arbitrary 3D polyhedral models can be displayed by a ray-based haptic rendering technique \cite{Basdogan97} or the intermediate plane method, which provides a planar approximation to a surface at the point of contact \cite{Adachi95} \cite{Mark96}.

Gillespie and Colgate \cite{Gillespie97} reviewed the methods available for the display of the moving objects dynamics. Most of the simulators with haptic display to date are designed based on the “Coupled Force Balance” scheme, which includes the following methods:

- Coupling through Spring Damper Pairs
- Coupling through Bilateral Constraint Forces
- Coupling through Unilateral Constraint Forces
- Coupling through Repeated Impulse

For Coupling through Spring Damper Pairs, each link or contact between bodies is modelled as a spring or spring-damper connection. The forward dynamics of each body is computed independently according to the Newton's laws. Although theoretically feasible, this approach is computationally expensive to implement. The presence of highly disparate time constants creates “stiff” differential equations, requiring specialized differential equation solvers. For Coupling through Repeated Impulse \cite{Chang97}, if a collision occurs, the collision response module must find the correct impulses to prevent interpenetration. Also, the time of collision for all pairs
of objects involving either of these two objects needs to be recomputed and updated in the heap.

The impulse-based method is more appropriate for non-linked (unilaterally constrained) rigid body environments than for linked (bilaterally constrained) rigid body systems. In impulse-based simulation, multiple contacts at a fixed time step are not allowed. Instead, a variable time step integration method is used and impulses are treated as a sequence of collisions over time [Mirtich94]. Hence, the impulses can be handled individually and there is no need to solve the Linear Complementarity Problem [Baraff94] for a multiple contact situation.

In another approach, impedance display is used to sense motion and to produce the force feedback. An image of the manipulandum handle is placed in the virtual environment and attached to a certain virtual object through the virtual coupler (i.e. a spring and damper together). Virtual coupling is a bridge between the haptic interface and the virtual environment. It allows stability to be guaranteed without any simulation-specific parameter tuning [Colgate95]. There are two key elements in the approach,

(a) The simulation method ensures that the environment is fully or approximately discrete time passive;

(b) The handle of the virtual tool is connected to the handle of the haptic display via a multi-dimensional coupling consisting of stiffness and damping.

In this work, the spring-damper system is used to couple the peg with the phantom to reduce the vibration. A geometric model is used to build the haptic surface of the hole and the PointShell method is employed for haptic rendering. The overall approach used for haptic rendering is described in the following subsections.
3.2.2.1 Spring-Damper System

The peg is coupled with the phantom (i.e., the manipulation point) through a spring-damper system as shown in Fig. 3-5. The dynamic of this system can be defined by Equation 3-1 [McNeely99]. The hole is static in the environment while the peg can be translated and rotated. The peg is a dynamic rigid object in the virtual environment and the forces and torques generated on it as the result of the interaction with the hole are transferred to PHANTOM through the spring-damper system.

\[ F = a \cdot x + b \cdot \dot{x} + m \cdot \ddot{x} \]  (3-1)

In this relationship

- \( x \) is the displacement of the peg
- \( \dot{x} \) is the velocity of the peg
- \( \ddot{x} \) is the acceleration of the peg
- \( a \) is the coefficients of the displacement
- \( b \) is the coefficient of the velocity
- \( m \) is the mass of the peg

![Figure 3-5 Spring-damper system](image)

3.2.2.2 Geometric Model

Polygonal models are widely used since they are simple [Zachmann97] [Gottschalk96] for geometrical modelling. Implicit functions have also been used as an alternative method to model the general shape of an object by describing the object surface. The tangent to the surface is used to determine the collision/penetration [Salisbury97].
NURBS (Non-Uniform Rational B-Splines) [Thompson97] and other models developed for CAD environments have been used as geometric models for haptic rendering. Ruspini, et al. [Ruspini97] created a haptic interface library "HL" for rapid incorporation of haptic environments into graphic models using "virtual proxy" and "force-shading" methods.

Avila and Sobierajski [Avila96] found that similar to pixels in 2D pictures, voxels (an array of 3D rectilinear volume elements) could be used to represent three-dimensional environments, with each voxel corresponding to a force vector. This volumetric data representation is further explored for haptic display, especially in complex virtual environments such as teeth drilling and surgery [Yagel96a] [Yagel96b].

In this work, the geometric model is developed using the Ghost modelling package supplied with PHANTOM haptic device. The geometric model of the hole is constructed by employing triangle polygon mesh. The mesh is formed by first defining 4 triangle polygons and then rotating the 4 polygons around the centre of the hole (Fig. 3-6). This reduces the computation time during haptic rendering as only the polygons representing the top and inner surfaces of the hole are considered during haptic rendering.
The geometric model of the hole can be also constructed using two cylinders forming the inner and outer surfaces of the hole, while the top and the bottom surfaces of the inner cylinder are either removed or not considered, as shown in Fig. 3-7. This method can reduce the computational time further.

![Cylindrical approach to building the geometrical model of the hole](Image)

**Fig. 3-7** Cylindrical approach to building the geometrical model of the hole

### 3.2.2.3 PointShell Method

The haptic rendered model of the peg-in-hole insertion generating force data is constructed using the PointShell method [McNeely99]. The PointShell of an object is the collection of all the points forming the surface of the object. A normal vector directing from the surface of the object to its inside is assigned to every point on the PointShell as the direction of the contact force [Renz01]. Fig. 3-8(a) illustrates the normal vectors of a PointShell.

Similarly, in the PointShell model developed for the peg-in-hole insertion virtual environment, the directions of the vectors assigned to singular points depend on the normal of the contact surface (Fig. 3-8(b)). They are assigned when the peg and hole are in contact. A point \( P(x, y, z) \) is a singular point of the curve \( C(x, y, z) = 0 \) if \( C_x(P) \)
\[ C_y(P) = C_z(P) = 0, \] where \( C_x, C_y \) and \( C_z \) denote the partial derivatives of \( F \) relative to \( x, y \) and \( z \).

Due to regular geometric forms of the peg and the hole, the depth of the peg penetrated into the hole can be easily calculated. Assignment of the normal vectors is also quite straightforward. This removes the need to use voxmap which is created by discretising the static environment space and used for collision detection by probing it with the PointShell of a dynamic object [Renz01] [McNeely99]. The voxmap is represented by its bounding box and a curve surface inside the bounding box (see Fig. 3-9).

![Fig. 3-8 PointShell and singular points](image)

![Fig. 3-9 Voxmap. (a) Voxmap (shown only in 2D); (b) Collision (penetration).](image)

Two different PointShell approaches have been explored to build the haptic rendered model. In the first approach, the peg is defined as a dynamic object described by a collection of points as shown in Fig. 3-10(a). The dotted points are singular points at
both the tip and side of the peg. The black points are not singular and are located on the side of the peg.

According to Friction Law, the frictional force, $f_{\text{friction}}$, is a fixed fraction of normal force, $f_{\text{normal}}$. If friction is not considered, the direction of the force generated at a dotted point will be normal to the surface of the hole at that point. Whereas the direction of the force generated at a black point will be opposite to the surface normal of the peg, as shown in Fig. 3-10(b). This implies that the direction of the force at a singular point (the dotted point) depends on the direction of the normal of the hole surface while the direction of the force at a non-singular point (the black point) relies on the direction of the normal of the peg surface. The force signals are generated only for the points located within the seat of the hole as shown in Fig. 3-10(c).

![Diagram](image)

**Fig. 3-10** The first approach used in haptic rendering modelling

In the second method, only the points at the two ends of the peg and the hole are considered (Fig. 3-11(a)). It is possible to reduce the points further by considering the points on the sides where the hole and peg come in contact with each other as illustrated in Fig. 3-11(b). The directions of the force vectors at dotted and black points are the same as the first method. The second method is used in this work as it requires less computation time. The force generated at each point is the sum of the normal force and the friction force exerted at that point, as shown in Fig. 3-12.
Normal Force

As shown in Fig. 3-12, the direction of the normal force is perpendicular to the contact surface and points to the object on the surface. Its magnitude at each point is calculated by

\[ f_n = k \cdot d + c \cdot d_a + b \cdot \nu \]  

(3-2)

where

- \( d \) is the depth of the point in the opposite object
- \( d_a \) is the accumulated depth during a continuous contact between the point and the static object
- \( \nu \) is the velocity of the object and is calculated by the current depth minus the last depth divided by the sampling time
- \( k \) is the stiffness coefficient
- \( b \) is the damping coefficient
- \( c \) is the coefficient for the accumulated depth

The above formula is used for the normal force generated in a 3 DOF phantom. The first element, \( k \cdot d \) generates a stronger force for a larger penetration depth. The third
element \( b \cdot v \) produces stronger force when the collision velocity is higher. The second element \( c \cdot d_a \) generates a stronger force for a larger accumulated depth. The depth, the accumulated depth and the magnitude of the force at each point are calculated at each loop. In continuous loops, if the depth of the point in the opposite object \( d \) is nonzero, it is added to the accumulated depth at each loop. This will increase the value of the accumulated depth \( d_a \) and the magnitude of the normal force \( f_n \).

The accumulated force can push the two objects in contact further away until the depth \( d \) becomes zero which in turn stops the increase of the accumulated depth \( d_a \). In this state, there is no depth or deformation between the two objects but the normal force still exists to balance the forces as the accumulated depth is not zero. Thus \( c \cdot d_a \) maintains the rigidity of an object and prevent deformation.

3.2.2.5 Friction Force

The direction of the friction force is the same as the contact surface and opposite to the moving object. The magnitude of the friction force generated at each point is calculated by

\[
f = \sigma z
\]

(3-3)

where

\( z \) is the strain describing micro-movements between the two objects, which is not allowed to exceed a small value called the breakaway distance \( z_{\text{max}} \).

\( \sigma \) is the stiffness relating force to strain

Assuming \( x_i \) is a fixed point on the moving object, and \( y_i \) is an adhesion point on the fixed object as shown in Fig. 3-13, the value of \( z_i \) is calculated by

\[
\begin{align*}
  z_i &= x_i - y_i \\
  y_i &= x_i \pm z_{\text{max}}, \quad \text{if} \ |x_i - y_{i-1}| > z_{\text{max}} \\
  y_i &= y_{i-1}, \quad \text{otherwise}
\end{align*}
\]

(3-4)
When the object moves from position $x_0$ to $x_1$, where the distance between $x_0$ and $x_1$ is less than the breakaway distance $z_{\text{max}}$, then $y_0 = y_1$ and the friction will be $f = \sigma^*(x_1 - y_1)$. When the object moves from position $x_0$ to $x_2$ where the distance between $x_0$ and $x_2$ is greater than the breakaway distance $z_{\text{max}}$, then $y_2 = x_2 - z_{\text{max}}$ and the friction force will be $f = \sigma^* z_{\text{max}}$.

![Fig. 3-13 Definition of the strain $z$](image)

### 3.2.3 Collision Detection and Depth Calculation

Collision detection, interference detection or contact determination has been an area of study in the field of computer graphics [Lin98]. There are a number of algorithms developed for collision detection such as the Lin-Canny closest features algorithm [Lin93], V-Clip (Voronoi-clip) [Mirtich96] [Mirtich97], I-Collide [Cohen95], OBB (Oriented Bounding Box) –tree [Gottschalk96], KDS (Kinetic Data Structure) [Basch99], Q-Collide [Chung96] and QuickCD [Klosowski98].

I-Collide library and V-Clip system all use bounding volume hierarchies for effective collision detection. I-Collide library is an interactive and exact collision detection library for environments with many convex polyhedral objects. Many non-convex polyhedra may be decomposed into a set of convex polyhedra, which may then be used with this library. V-Clip also handles penetrating polyhedra, which allows the algorithm to be extended to non-convex polyhedra when they are represented as groups of convex polyhedra. Voronoi-clip system can be used with any polyhedral object, which can be non-convex or even disconnected. It can return the penetration depth if the objects penetrate and can be used within larger architectures such as the
V-Collide library. Hubbard [Hubbard96] and Quinlan [Quinlan94] present bounding volume hierarchies using spheres. Gottschalk, et al. [Gottschalk96] use a rectangular box as the bounding volume (Oriented Bounding Box), while Krishnan, et al. [Krishnan98] use the Spherical Shell bounding volume, which looks somewhat like a truncated cone.

There are also collision detection methods for dynamic systems [Cameron90], for rigid body object in motion [Canny86] [Schoemer95] and for non-rigid bodies [Herzen90] [Liu96].

In this work, collision detection is performed using a function supplied by GHOST. The depth is calculated by projecting the vector which points from the current point to the intersection point along the direction of the force at the intersection point (see Fig. 3-14). The flowcharts illustrating the generation of forces and torques are shown in Fig. 3-15 and Fig. 3-16. The collision detection and depth calculation are used primarily to calculate the normal force. The flowchart of the algorithm used for collision detection and calculation of the depth of the peg is shown in Fig. 3-17.

![Definition of depth](image)

Fig. 3-14 Definition of depth
In order to illustrate the approach, collision detection and depth calculation procedures for a black point on the peg are explained here. Initially, the `checkIfPointIsInside_WC` function is executed to check whether a black point on the hole is inside the peg. If it is, then the depth of the black point inside the peg and the direction of the normal force reacting to the peg at this point can be found using the `intersect` function provided by the Ghost SDK for a cylinder. This function detects the intersection between an object and a line defined by the current and previous locations of a point on the other object. This is illustrated in Fig. 3-14. For intersection between the peg and the hole, the depth is calculated by projecting the vector which points from the current point to the intersection point along the normal of the peg at the intersection point.
Go to the next dotted point and the next black point (The number of the dotted points is the same as that of the black points)

Fig. 3-16  Force and torque generation

- Calculate the normal force for the current dotted point
- Calculate the friction force for the current dotted point
- Generate the force and torque for the current dotted point

- Calculate the normal force for the current black point
- Calculate the friction force for the current black point
- Generate the force and torque for the current black point
Fig. 3-17  Program blocks of collision detection and depth calculation for the peg-in-the hole insertion
In the virtual environment, the black points are static since the hole is static. Hence, the current and previous locations of a black point are identical. However, in the `intersect` function, depth is calculated based on intersection of the line connecting the previous and current points and the peg. This constraint is overcome by assuming a previous black point which has the same coordinates as the current point but defined in the previous coordinate frame of the peg (Fig. 3-18). The assumed point will be the same distance \( L \) from the centre of the peg’s tip as the current point and makes the same angle with it. The depth of the black point on the peg can be determined according to Fig. 3-14, as mentioned before.

3.2.4 Data from Virtual Environment

Phantom 1.0 provides 3 DOF only along X, Y and Z axes. The data produced by the haptic virtual environment at each sampling interval is stored in a file. Each record consists of

- \( x(k), y(k) \) and \( z(k) \): Coordinates of the point
- \( F_x(k), F_y(k) \) and \( F_z(k) \): Reactive forces to the peg
- \( T_x(k) \) and \( T_z(k) \): Reactive torques to the peg
\( \theta_x(k), \theta_z(k) \) : Angles of the peg relative to \( x \) and \( z \) axes respectively

\( \Delta \theta_x(k), \Delta \theta_z(k) \) : Angular control signals or actions about \( X \) and \( Z \) axes

\( \Delta y(k) \) : Position control signal along \( Y \) axis

where \( \Delta \theta_x(k) = \theta_x(k) - \theta_x(k-1) \), \( \Delta \theta_z(k) = \theta_z(k) - \theta_z(k-1) \) and \( \Delta y(k) = y(k) - y(k-1) \).

The data units used in the virtual environment and in the real rig are different. Hence, the data obtained from the virtual environment is scaled according to Table 3-1 and Table 3-2. The \( Y \) and \( Z \) axes in the virtual environment correspond to the \( Z \) and \( Y \) axes of the physical rig, respectively (see Fig. 3-3). The coordinate frame of the Force/torque transducer is rotated by 45° about the \( Z \)-axis relative to the coordinate frame of the hole in the physical rig (see Section 6.3). Hence, the forces and torques produced in the virtual environment should be transformed to the coordinate frame of the force/torque transducer of the physical rig.

Table 3-1  Scaling factors between the virtual environment and the real rig

<table>
<thead>
<tr>
<th>Column</th>
<th>Real Rig</th>
<th>Virtual Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( F_x )</td>
<td>( F_x(k)/0.056448 )</td>
</tr>
<tr>
<td>2</td>
<td>( F_y )</td>
<td>( F_y(k)/0.056448 )</td>
</tr>
<tr>
<td>3</td>
<td>( F_z )</td>
<td>( F_z(k)/(-0.056448) )</td>
</tr>
<tr>
<td>4</td>
<td>( M_x )</td>
<td>( T_x(k)/0.056448*0.039 )</td>
</tr>
<tr>
<td>5</td>
<td>( M_y )</td>
<td>( T_y(k)/0.056448*0.039 )</td>
</tr>
<tr>
<td>6</td>
<td>( Pos )</td>
<td>( y(k)*500 )</td>
</tr>
<tr>
<td>7</td>
<td>( step_x )</td>
<td>( \Delta \theta_x(k)*1000/\pi )</td>
</tr>
<tr>
<td>8</td>
<td>( step_y )</td>
<td>( \Delta \theta_z(k)*1000/\pi )</td>
</tr>
</tbody>
</table>
Table 3-2 Units in the virtual and physical environments

(100 uf = 5.6448N, 50 pules = 0.1 mm, 1 mm = 0.039 inch, 0.18º = \pi /1000)

<table>
<thead>
<tr>
<th>Virtual Environment</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>$F_x(k)$</td>
<td>$T_x(k)$</td>
</tr>
<tr>
<td>unit</td>
<td>N</td>
<td>mm</td>
</tr>
<tr>
<td></td>
<td>$F_y(k)$</td>
<td>$T_y(k)$</td>
</tr>
<tr>
<td></td>
<td>$F_z(k)$</td>
<td>$T_z(k)$</td>
</tr>
<tr>
<td></td>
<td>$y(k)$</td>
<td>$\Delta \theta_x(k)$</td>
</tr>
<tr>
<td></td>
<td>$\Delta \theta_y(k)$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Real Rig</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>$F_x$</td>
<td>$M_x$</td>
</tr>
<tr>
<td>unit</td>
<td>uf</td>
<td>uf-in</td>
</tr>
<tr>
<td></td>
<td>$F_y$</td>
<td>$M_y$</td>
</tr>
<tr>
<td></td>
<td>$F_z$</td>
<td>Pos</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$step_x$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$step_y$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Step number</td>
</tr>
</tbody>
</table>

Examples of different training data sets extracted from the virtual environment for the perception module are shown in Fig. 3-19 ~ Fig. 3-21. Friction is considered in this model. At each sampling point the forces/torque, position and angles are recorded into the training data file. The sampling time between two neighbouring sampling points can be changed according to the dynamics of the manipulation. A high sampling frequency can be required if the forces and torques generated during the manipulation change rapidly. In this thesis, the sampling time keeps same for each run of an insertion in the virtual environment, which is 0.1s.
Fig. 3-19  A virtual insertion data set (initial angles: $\theta_x = 0$ and $\theta_z = 0.267035$ rad).
(a) Positions; (b) Forces; (c) Torques; (d) Angles.
Fig. 3-20  Another virtual insertion data set (initial angles: $\theta_x = -0.01571$ rad and $\theta_z = 0.298451$ rad).  (a) Positions; (b) Forces; (c) Torques; (d) Angles.
Fig. 3-21  A third virtual insertion data set (initial angles: $\theta_x = -0.00135$ rad and $\theta_z = -0.02709$ rad).  (a) Positions; (b) Forces; (c) Torques; (d) Actions.
3. 3 Another Haptic Rendering Method

The haptic model of the hole in the virtual environment has also been constructed using the cylindrical approach. The forces at each dotted point in Fig. 3-11 are calculated according to the flowchart provided in Fig. 3-22. The method does not use polygons to approximate the circle and hence can prove more accurate.

![Flowchart](image-url)

Fig. 3-22 Haptic rendering of the hole using cylindrical approach
The method to find the intersection and the depth in this method is illustrated in Fig. 3-23. The forces at the black points are calculated according to the method described in Fig. 3-18.

An example of training data extracted from the virtual environment using the cylindrical method is shown in Fig. 3-24. In a successful insertion, the position of the peg along the Y axis varies between 110mm and 60mm compared to 70-20 mm achieved in the previous method (Fig. 3-20(a)).
Fig. 3-24  Data obtained from the virtual environment using cylindrical method. (a) Positions; (b) Forces; (c) Torques;

3.4 Chapter Summary

Graphic modelling and haptic rendered modelling of the peg-in-hole insertion in the virtual environment have been studied in this chapter. The virtual haptic rendered model is used in the virtual manipulation of the task. The algorithm used to calculate
collision between the peg and the hole and the generated forces and torques are also described. Some examples of the forces and torques generated during virtual manipulation are presented.
CHAPTER 4  SKILL TRANSFER SYSTEM

4.1 Introduction
This chapter provides an overview of the process through which manipulation skills are acquired from the manipulation performed by the operator in a haptic rendered virtual environment and transferred to the physical manipulation system. Initially the overall process is described. Then three modules developed in this work to carry out the acquisition and the transfer of the skills including the Perception Module, Manipulator Task Planner Module and Learning Module will be described.

4.2 Imitation and Skill Transfer
Imitation is the result of interaction between perception, memory, and motor control. In addition to self-discovery and observation, learning of skills in humans generally takes place through training by an instructor [Adams71] in the psychomotor domain, where ‘motor’ is an observable movement response to a stimulus [Harrow72]. According to Smith and Smith [Smith62], there are three types of movements. The first is the postural movement which regulates body positioning. The second is locomotor movements, which translate and rotate a body, and the third category includes manipulative movements. The manipulative movements are the focus of this study as they are the type of movements emulated by robotics manipulators in automatic assembly.

Simpson [Simpson66] has proposed a model with seven hierarchical classification levels for the human behaviour in the psychomotor domain. The model is developed as a taxonomy and consists of:
(a) Perception, which deals with sensory stimulation, cue selection and translation.
(b) Set, which deals with mental, physical and emotional sets.
(c) **Guided response**, which deals with imitation and trial and error learning.

(d) **Mechanism**, which deals with the mechanics and habituation of movements.

(e) **Complex overt response**, which represents a higher level of skill in performing the motor skill by the learner.

(f) **Adaptation**, which adapts the skills to variation in the environment and task.

(g) **Origination**, which develops an original skill or new movement patterns to fit a particular situation or specific problem.

The first two levels do not represent easily observable behaviour. The next three levels represent a learning sequence inherent in many motor skills. The last two levels can be interpreted as refinement of basic motor skills and the creation of new movement patterns to achieve the same objectives.

In the context of manipulative movements, in the last two levels, slow, stiff and cautious movements transform to smooth, ballistic trajectories requiring much less mental concentration [Jeannerod88] [Rosenbaum91] [Holding89]. This transformation from a cognitive form of processing to a more automatic one has been described by terms such as *explicit* versus *implicit* [Mathews89], *deliberative* versus *reactive* [Schoppers87], *declarative* versus *procedural* [Anderson82], and *declarative* versus *reflexive* [Kupfermann85].

The emulation of such taxonomy in machine learning is not fully achievable due to basic differences between humans and machines. In this thesis, the extent to which the manipulative learning can be emulated through demonstration is explored with particular focus on stages (a), (c), and (d) – (f), as these are of particular relevance to manufacturing applications of machines and robots.
The overview of the skill transfer system in this research will be provided and the skill transfer procedure will be discussed. Then, the Perception Module, Manipulator Task Planner Module and Learning Module will be described.

4.3 Steps in Skill Transfer to Machines

A review of the work conducted on the transfer of skills from the human to machine reveals that the process is multi-stage and consists of the following steps:

(a) **Data collection:** This is the data representing the characteristics of the manipulation. It can be generated analytically when possible, or extracted from the manipulation carried out by a human expert in virtual or real environments. Examples of the demonstration should be carefully chosen in order to maximize the quality of the skill. Data should contain as much realistic information as possible to maximize the effect of learning. Although the cost of data collection is high, virtual environments or simplified tasks can be used for data reduction. Data should be of minimized dimension to reduce learning efforts.

(b) **Skill model or skill representation:** The skill model can be used to give on-line advice to robots or less-skilled operators. Skill representation is an essential issue since it determines the choice of skill learning algorithm. Skills can be represented or modelled by mathematical formalism, or by expert rules. The skill learning method can be regarded as an optimization algorithm which lets the skill model learn in the form of an optimization of a nonlinear function. An appropriate function approximator (FA) is chosen according to the nature of the skill model, i.e. parametric or nonparametric, as well as, local or global. Task decomposition introduced in learning procedures reduces the dimensionality of a learning space [Bondi88] [Sanger95] [Speeter91].
(c) **Skill learning**: Skill learning is quite important in the skill acquisition. The choice of learning algorithms is based on the nature of the task, available sensory system and the skill representation method. Learning algorithms can be unsupervised or supervised. In unsupervised algorithms, an agent learns from interaction with its environment rather than from the correct decisions paired with specific sensory input. Unsupervised algorithms are autonomous but slower than supervised, especially in high-dimensional tasks. An example of unsupervised algorithm, associative reinforcement learning learns to respond to each input with the action that has the highest expected evaluation. Through repeated manipulation attempts, the admittance mapping can be learned to work robustly despite the high degree of uncertainty [Gullapalli94]. In reinforcement learning, the agent uses system payoffs as a guide to form effective decision policies rather than learning from examples of correct behavior [Sardağ(online)].

(d) **Task planning**: This is the planning of the manipulation task either based on geometric path planning techniques or contact states. In the latter method, different contact states are identified using neural networks which take into account the torques and forces generated in each type of contact. The output of the system is the classification of contact states. In real-life manipulation, geometrical methods may be complex and failure may occur due to misalignment caused by uncertainty in position. Hence, contact states can identify the quality of manipulation at a sufficiently early stage of the process [Cervera95].

(e) **Skill refinement**: This consists of modifying actions that are associated to known states (skill adaptation) and assigning appropriate actions to states that are not encountered yet. There are methods which do not rely on a model of the plant but directly estimate on-line the necessary changes in the calculated actions
Skill representation should support an incremental learning algorithm for skill refinement.

(f) **Skill transfer or application:** This is the final stage in which skills are transferred or applied to a physical machine to perform a task by reusing and combining the acquired skills.

These steps are in line with the overall approach described in Section 1.2. This correspondence is further highlighted by Fig. 4-1.

![Diagram of skill transfer steps and system modules](attachment:image.jpg)

**Fig. 4-1** Relationship between the skill transfer steps and the system modules

### 4.4 Contact States

Assembly can be defined as the process of mating work-pieces in a desired configuration. In the peg-in-hole insertion, the robotics system moves the peg along the inner surface of the hole. Contact states are used in this work to model the peg-in-hole insertion process and associated manipulation skills in the virtual environment.

A contact state represents a particular instant of the assembly. Some contact states known as critical states indicate a significant change or event in the process. Transition from one critical state to another is referred to as state transition.
The following critical states are monitored and identified in the modelling of the virtual manipulation:

- **Jamming states**: the states at which the absolute values of forces/torques applied on the peg exceed certain limits (e.g., 5.6N for force)
- **Goal or final states**: the states at which the peg is fully inserted into the hole. Insertion is considered complete when the goal state is reached.
- **Better states**: the states which are nearer to the goal state
- **Worse states**: the states which are further away from the goal state

4.5 Perception Module

As mentioned before, the basic skills are derived and structured by the perception module. It derives the basic skills from the training data produced through manipulation in the virtual environment and stores them in models which can generate output according to inputs. These models can also be viewed as databases which store the skills. The skills stored in the database or in the model are used by the planning module to control the manipulator. The skills in the database can be further augmented by the skills learned on line during physical manipulation by the manipulator.

The human performs manipulation by choosing from a limited but possibly large repertoire of movement primitives or basic skills. A manipulation task usually consists of a sequence of basic skills. Identification of these basic skills and mapping them on to equivalent series of robot manipulation primitives, form the core of an algorithm for skill acquisition and transfer of those skills from the human to a robotic manipulator. Such skill-based manipulation is an effective way for a robotic manipulator to execute a complex task.
The basic skills are identified and modelled based on the contact states and state transitions taking place in the virtual manipulation [Takamatsu99] [Onda95]. This ensures that the basic skills are associated with the task, rather than the configuration of the physical manipulator performing the task [Nakamura96]. Using this approach, the basic skills can be automatically extracted from the manipulation carried out in the virtual environment.

The skill set associated with a task can be defined in a hierarchical structure. For example, the whole insertion process can be divided into search and insertion phases. The search and insertion skills are two high level skills which result in critical state changes by driving the peg from the initial state to touch the hole and insertion of the peg. Each high level skill can be divided into low level skills resulting in minor state changes.

The basic structure of the perception module is illustrated in Fig. 4-2. The training data are preprocessed, classified and perceived to acquire skills to store in the skill database. Type 1 skills are based on optimal state sequence method which can drive the peg to the goal state in minimum number of actions. Thus state classification is needed for this method to determine the difference between the current state and the next state or the goal state. Type 2 skills are based on Locally Weighted Projection Regression (Lwpr) method [Lwpr(online)] which drives the peg according to the actions performed by an operator under the same current state. In the following sections, different components of this module will be described.
4.5.1 Preprocessing of Training Data

The training data produced by the haptic rendered virtual environment may include inconsistent or unintended actions. Such data should be identified and removed from the training data before processing the data for skill acquisition. It will be shown that the learning algorithm primarily learns the actions that result in an improvement of state.

Ineffective actions do not improve the state significantly. For example, if a peg is jammed in the hole and after some actions the peg cannot be inserted further, then such actions are ineffective. Unintended actions might be due to accidental errors. For example, when the operator manipulates the peg, he/she might use too much force because of distraction.
The actions which result in worse states should be also removed. A worse state is further away from the goal state than its predecessor. An insertion is termed successful when the peg is inserted into the hole to a pre-defined depth. The position error and the maximum magnitude of forces can be used to indicate whether the state is closer to or further away from the goal state. The closer the position of the peg to its desired position and the smaller the maximum magnitude of forces, the higher will be the reward. An optimum sequence has both short insertion time and high action reward. The reward can be defined as [Gullapalli92]:

\[
    r = \begin{cases} 
    r_{\text{base}} & \text{if all forces are } \leq F_{\text{th}} \\
    r_{\text{base}} - 0.1F_{\max} & \text{otherwise}
    \end{cases} 
\]

(4-1)

where \( r_{\text{base}} = \max(0, 1 - 0.01(\|X\| + \|Y\| - |\Theta|)) \), \( F_{\max} \) is the largest magnitude of force component among \( F_x, F_y \), and \( F_z \), and \( X, Y \) & \( \Theta \) are the position errors and angle errors between the peg and the hole. In the 3 DOF peg-in-hole insertion, \( |X| \) and \( |Y| \) are nearly zero and \( |\Theta| \) is the magnitude of the angles between the peg and the hole. The value of \( F_{\text{th}} \) is set to 0.5 N in the virtual manipulation environment.

In order to remove ineffective and unintended actions, the following algorithm is applied to the training data [Kaiser96], where \( u \) is the action vector and \( x \) is the state vector of the sensor inputs.

1. Remove irrelevant perceptions, i.e., the data which does not satisfy the constraint:

\[
    \|x(t_1) - x(t_2)\| \leq \delta \|u(t_1) - u(t_2)\|, \ 0 < \delta, \ \text{where } \delta \text{ is a scale-factor or coefficient.}
\]

This constraint ensures that two states with similar associated actions are identical.

2. Remove all actions that do not contribute to state changes, i.e., remove samples \((x, u)\) with

\[
    \|u\| \leq \delta_s, \ \delta_s \geq 0, \ \text{where } \delta_s \text{ is an application specific threshold.}
\]
If the modal $||.||$ of an action is very small, then the action does not contribute to state changes.

3. Remove the operator’s rough control by applying

$$u(t) = u(t+1) = (u(t) + u(t+1))/2$$

If the difference between two continuous actions is large then both actions are replaced by their average value.

4.5.2 Skill Types

As illustrated in Fig. 4-2, perception module acquires two types of skills from the pre-processed training data based on state changes:

(a) Skills based on both the current state and the next state (obtained by optimal state sequence method)

(b) Skills based on the current state only (obtained by Locally Weighted Projection Regression method)

The first type of skills is learned during task sequence planning or trajectory optimization which finds the best state change sequence. In order to apply this skill, the next desired state or the method of choosing the next state should be known. State changes with the same current state but different next states might result in different output actions.

The second type of skills is acquired during performing a task with no obvious or fixed state change sequence. It is only based on the current state to simplify the skill learning process. It does not require to find an optimum state change sequence or to follow a pre-defined state change sequence. The second type of skills can be derived from the virtual environment and further enhanced online during physical manipulation.
The inputs for Type 1 skills are jamming states while the inputs for Type 2 skills are any state where the human operator takes effective actions. The outputs of Type 1 skills are actions defined according to the angles of rotation of the hole, while those for Type 2 skills are human actions. Skills of Type 1 are applied when the peg is jammed during the peg-in-hole insertion in physical rig. Skills of Type 2, on other hand, are employed when the peg is moved away from the jamming state and skills of Type 1 have proved ineffective after a number of trials.

The skills of Type 1 are acquired from virtual manipulation data by employing the Incremental Support Vector Machine (SVM) classification model [Fung02] [Amund03] whereas skills of Type 2 are modelled by the Locally Weighted Projection Regression (Lwpr) [Lwpr(online)]. The skills database consists of the classification models obtained from SVM and the Lwpr models. The details of the approach will be provided in Chapter 5.

The Lwpr model learns human actions. When the peg is jammed the peg will be moved up for a small distance, resulting in a state different from jamming state with smaller absolute values of $F_x, F_y, F_z, M_x$ and $M_y$. This state can be used as inputs in generating Type 2 skills.

The classification model learns about the jamming states and actions associated with them. In the real rig, an action is the number of steps required to drive the hole about any of the axes to align the axis of the hole and that of the peg in one step. In the virtual environment, the action is the angles between the peg and the hole which will be converted to stepper motor steps to correspond to those in the real rig. Jamming states are classified according to the outstanding forces/torques and the angles between the hole and the peg axes. The classification is used during physical
manipulation to decide on the amount of rotation of the hole about each axis when
the peg is jammed.

During the insertion, the amount of rotation of the hole in each step is recorded. The
total indicates the amount of rotation required to reach a final state from a jamming
state. This information is learned by the system and applied when a jamming state is
encountered in the future manipulations.

The manipulation actions of a human operator are not exclusive to jamming states.
For example, in the insertion process performed in the virtual environment as
illustrated in Fig. 4-3, the operator has manipulated the peg even though the
generated force has an absolute value less than 5.6 N (the threshold indicating a
jamming state).

![Fig. 4-3 Forces in a successful insertion in the virtual environment](image)

In real-time online trajectory planning, skills of Type 1 are considered and applied
first. If they do not result in states closer to goal state, skills of Type 2 are employed.
Although useful, it is rather difficult to acquire all the jamming states in the database.
The differences between the forces and torque from the real rig and those from the
virtual environment will also create jamming states not represented in the database.
Hence, online incremental learning is incorporated in the approach to enhance the
efficiency of the process by compensating for cases not identified in the offline learning process. If none of the skills of Type 1 or Type 2 can produce a satisfactory outcome, actions produced from a fuzzy algorithm [Nguyen(thesis)] or heuristic rules learned on line can be applied.

The differences between the two types of skills are shown in Table 4-1.

Table 4-1  Inputs, outputs and privilege for models of Types 1 and 2 skills

<table>
<thead>
<tr>
<th>Inputs for skill model</th>
<th>Model of Type 1 Skills</th>
<th>Model of Type 2 Skills</th>
<th>Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jamming states</td>
<td>any state where the human operator takes effective actions</td>
<td>$F_x$</td>
<td>Reactive force to the peg along X-axis</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$F_y$</td>
<td>Reactive force to the peg along Y-axis</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$F_z$</td>
<td>Reactive force to the peg along Z-axis</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$M_x$</td>
<td>Reactive torque to the peg about X-axis</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$M_y$</td>
<td>Reactive torque to the peg about Y-axis</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pos</td>
<td>Position of the peg along Z-axis</td>
<td></td>
</tr>
<tr>
<td>Outputs to learn</td>
<td>Actions calculated according to the rotation angles of the hole</td>
<td>Human actions</td>
<td>$x_{step}$</td>
<td>Rotation of the hole about X-axis</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$y_{step}$</td>
<td>Rotation of the hole about Y-axis</td>
</tr>
<tr>
<td>Privilege</td>
<td>First</td>
<td>Second</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.5.3 Optimum Sequence

Some skill acquisition methods can find the best sequence [Lee95] [Lee92] in the augmented state space formed by the current state and the next state. For a one-dimensional dynamic system, the augmented state space is a plane with the current
states as the X-axis and the next states as the Y-axis. A point in the augmented state space represents a feasible state transition.

At first, the data of many successful traces with different initial states are collected from the virtual environment. A feasible state transition region is then approximated by multiple sizes of hyper-ellipsoids using a Multi-resolution Radial basis Competitive and Cooperative Network (MRCCN). This is a bi-directional many-to-many mapping neural network capable of learning complex many-to-many relations of input and output, deriving more than one solution from the given value if necessary [Lee95].

The hidden units in MRCCN are Radial Basis Function Units (RBFUs). A RBFU is denoted by three parameters that define a hyper-ellipsoid in the augmented state space, called an accommodation boundary. Fig. 4-4 shows the accommodation and generation of RBFUs for a one-dimensional system. The initial sample data is used as a reference point for a given circular characteristic function (circle). When a subsequent teaching sample is within the circle, the reference point moves towards the sample and the shape also changes towards it (Fig. 4-4(a)). When the sample is out of the circle, a new RBFU is created at the sample point with a circular boundary (Fig. 4-4(b)) [Lee92]. This process is repeated for the updated and newly generated boundaries.

In the second stage, an optimal path from any state to the goal state in the feasible state transition region is sought based on the Bidirectional Dynamic Path Planning (BDPP) algorithm [Lee95].

Fig. 4-5(a) provides an example of the feasible state transition region of a one-dimensional dynamic system. The result of multi-resolution clustering looks like Fig.
4-5(b). The optimal transition from 0 to 12.5 is labelled in Fig. 4-5(c) by three points (0, 5), (5, 7.5), and (7.5, 12.5), which means the optimal path is $0 \rightarrow 5 \rightarrow 7.5 \rightarrow 12.5$.

Fig. 4-4 Accommodation and generation of RBFUs

Fig. 4-5 An example of a one-dimension dynamic system

However, this method requires a great deal of training data to explore the augmented state space and hence it is quite computing intensive for high dimension systems. For such reasons, this approach is not appropriate for on-line learning.

In this thesis, a simple method is employed to identify the optimal state change sequence for the insertion phase. Initially different states are classified according to the forces/torque and the angles of the hole. The classification is then used to recognize the state change sequence for each training data file, in which the outputs are actions such as rotating the hole. The optimum sequence to perform a task is either identified from the training data or generated by combining different sequences according to a criterion such as the shortest time. If Sequence A takes shorter time from the initial state to a mid-state but longer from the mid-state to the
final state than Sequence B, then the first part of Sequence A is combined with the second part of Sequence B to generate a sequence with shorter implementation time than both Sequences A and B (as shown in Fig. 4-6). The actions or outputs recorded for each state change in the optimum sequence with different initial state are defined as the basic skills. For the round peg-in-hole insertion, the data can be mapped symmetrically to reduce the amount of training data needed.

![Diagram of sequences](image)

Fig. 4-6 Combination of sequences illustrated in 2 dimensions (Assuming that the longer displacement takes longer time)

4.6 Manipulator Task Planner Module

The function of the Manipulator Task Planner Module is shown by the flowchart of Fig. 4-7. The conditions referred to in the flowchart including “Set initial angles of the hole”, “move the peg just above the hole”, “move the peg down a little”, “move the peg up a little” and “turn the hole” are considered as tasks in the peg-in-hole insertion process. Peg-in-hole insertion can itself be a task in a complex assembly process.
The block in the ellipse; “turn the hole according to the learned skills” can be further decomposed into lower level tasks which are described in details in the following pseudo codes:

If the peg is jammed
    If the inputs belong to the previous classified classes
        Apply skills of Type 1 (inputs)
If this is not possible let the peg be moved slightly upward
End
Apply skills of Type 2 (inputs)
Else
Apply skills of Type 2 (inputs)
End
End
Apply skills of Type 1 (inputs)
{
  Calculate the steps according to the class
  Turn the hole directly to the final state
}
Apply skills of Type 2 (inputs)
{
  Output the steps using the Lwpr model
  Turn the hole according to the output
}

The Task Planning Module plans the peg-in-hole insertion in the physical rig according to the jamming states. The combination of corrective angles about x and y axes, \((x_{\text{step}}, y_{\text{step}})\) is referred to as a class. Due to the resolution of the system and its accuracy, different jamming states may result in the same class.

Let’s assume a corrective rotation angle for the hole about \(X\) and \(Y\) axes varies between -4.5° and 4.5°. The number of equivalent corrective steps about each axis for a 0.18° rotation per step will be 51. This will result in 2601 classes if such variation is considered for corrective rotations about both \(X\) and \(Y\) axes. This represents a large number of classes which slows down the training process. In order to overcome this difficulty, two separate models are used to classify the jamming states according to angles of rotations about \(X\) and \(Y\) axes.

If the magnitude of the required corrective angle is more than 4.5°, the hole is rotated initially for 4.5°. Then the correction algorithm is re-applied to achieve the final realignment.
The Task Planning Module can also define the constraints governing the jamming states and the depth of a successful insertion. It also sets the velocity of the peg according to the magnitudes of forces measured and the position of the peg.

The insertion performed by a human operator is not dependent on jamming states and can be quite smooth. The task planning module, however, uses jamming states to progress and hence it is somehow different.

4. 7 Learning Module

During physical manipulation, a new stream of data is generated. This data can be either stored and processed after manipulation (batch learning) or processed online and discarded afterwards (online learning).

Batch learning is used to learn new Type 1 skills. The sum of $x_{\text{step}}$ and $y_{\text{step}}$ values observed from a jamming state to a final successful state is used by the learning module as a measure of the amount of rotation required to align the axis of the hole with the axis of the peg from a particular state.

Online learning can be used to learn new Type 2 skills. This will continuously improve the quality of the model and overcome the deficiencies arisen due to material wear and changes in the dynamics of the rig. Heuristic rules learned on line can be used to assist in the insertion process if skills of Type 2 do not resolve a jamming state. Online learning module will learn from the effective actions which improve the insertion states as the following pseudo codes show:

```
Normalize current forces, torque and position
[x_{\text{step}}, y_{\text{step}}] = system_output (normalized current forces, torque and position)
If x_{\text{step}} is not acceptable
    Input another value for x_{\text{step}}
    X_flag = 1
End
If x_{\text{step}} is not 0
    Turn the motor on the X axis by x_{\text{step}} steps
End
If y_{\text{step}} is not acceptable
```
Input another value for \textit{y\_step}
\textit{Y\_flag} = 1
End
If \textit{y\_step} is not 0
Turn the motor on the X axis by \textit{y\_step} steps
End
If \textit{X\_flag} = 1 or \textit{Y\_flag} = 1
system update (normalized current forces, torque and position, normalized \textit{x\_step} and \textit{y\_step})
End
system save (name)

\textbf{4. 8 Chapter Summary}

In this chapter an overview of the skill transfer system proposed in this work was provided. While the approach was developed based on the peg-in-hole insertion application, its generic nature was maintained as much as possible. The hierarchical learning structure developed in the work was introduced and its different components were described.
CHAPTER 5  SKILL ACQUISITION ALGORITHM

5.1 Introduction

Skill learning or skill acquisition is the major task in skill transfer. The key question is how the basic skills used by a human operator to perform a task can be transferred to a machine. The approach to some extent is similar to building knowledge-base systems from the knowledge acquired from human experts. In reality, however, there are major differences. Primarily because the human operators cannot always explicitly describe the skills deployed in a manipulation, although they can successfully perform it. This makes it difficult to describe the task of duplicating the human manipulation.

Hence, it is important to develop a modelling and learning strategy which can derive the human manipulation skills systematically. A significant amount of work is reported in the literature on modelling and learning of human skills [Sato97] [Nakawaki99]. The proposed techniques can be divided into symbolic learning (e.g. inductive, analytic, case-based, and relational), rule-based (e.g. Fuzzy Logic), connectionist (e.g. Neural Networks), Hidden Markov Models, and hybrids (e.g. Fuzzy Neural Networks and Neuro Symbolic Learning). The learning methods are then employed to optimize the parameters and to minimize the cost functions. Cost functions are mainly based on RMS (Root Mean Square), ARMA (Auto Regressive Moving Average) model, and HMM (Hidden Markov Model). The learning algorithms employed mainly includes optimization techniques such as a Gradient algorithm, a Genetic algorithm, and Simulated Annealing. In this chapter some commonly used learning algorithms will be reviewed, and then the skill acquisition method developed and applied in this work will be described.
5.2 Review of Skill Learning Methods

- Look-Up Table

The look-up table method is fast and simple for skill representation and learning, but it needs a large amount of memory. The CMAC (Cerebellar Model Arithmetic Computer) [Albus79] [Albus81] [Eldracher] [Miller87a] is a frequently used table look-up method. It is not trained to generate signals required to follow a particular trajectory. Instead, it reproduces the relationship between available sensory feedback parameters and the system command parameters over convex regions of the system state space [Miller87b]. The CMAC modelling method can provide continuous output and lead to a global optimum through efficient interpolation. The CMAC output is, however, very sensitive to sensor noise.

- Neural Networks (NN)

Neural networks are widely used for skill representation and learning [Asada91] [Li99]. Various neural networks (eg. MLP (Multi-Layer Perceptron) [Kawato87] [Ritter], time delay NNs [Kaiser94], group method of data handling NN [Chen91]) have been applied to model robot control as function approximators. Liu et al (1993) reports a special type of neural networks used for on-line classification and measurement of drill wear. After training, the neural networks can continuously monitor the drill conditions with features of thrust and torque as the input vector and wear states and drill wear measurements as outputs. The average flank wear of the drill can be also measured on-line by the neural networks [Liu93].

Another NN architecture and learning algorithm known as Hierarchically Self-Organizing Learning (HSOL) Network [Lee91], is used for self-organization of hidden units and approximating the arbitrary input-output mappings. A skill learning paradigm consisting of supervisory controller, chaotic neuron filter and associative
memory is employed to update the weights and structure of a feed-forward NN controller which is similar to the macroscopic model of brain [Eom96].

Conventional NNs are based on the misconception that no other physical entity in the brain can directly signal “changes” to the behaviour of cell. There is, however, clear evidence in neuroscience that there are “different pathways” through which “additional signals” could influence synaptic adjustments directly [Roy00]. In addition, standard techniques for function approximation like neural networks or associative memories are unable to work with rules.

- **Fuzzy Approximators**

  Fuzzy approximators [Baroglio96] [Young97] represent domain knowledge by means of rules understandable to the human operator and provide analytical output in the form of smoothed piecewise approximators. The design and development of fuzzy approximators require fewer examples, but larger training time. Fuzzy approximators can achieve on-line refinement of the skill model as NNs do.

  A good example of this methodology is the Fuzzy Drilling Direction Controller [Fuzzy-online] which was originally used in oil well drilling designed based on human skills (or human thinking).

### 5.3 Building Skills Database

The paradigm developed in this work to build and employ a skills database is illustrated in Fig. 5-1 and Fig. 5-2. As mentioned in Section 4.5.2, two types of skills have been identified in the development of the database.

The characteristics of this paradigm are discussed in the following sections.
Fig. 5-1  Skill learning paradigm: (a) Learning through SVM for Type 1 skills; (b) Learning through Lwpr for Type 2 skills
5.4 Off-Line Learning

The skill learning acquisition algorithms are described in this section in the context of the peg-in-hole insertion. This represents the micro-level implementation of the approach which varies from one application to another.

5.4.1 Acquisition of Skills Type 1

Skills of Type 1 are based on state classification. The training data can be collected from both the virtual environment and the physical rig. For the peg-in-hole insertion process, the collected training data is used to develop a model to estimate the
displacement angles required to overcome a jamming state based on the input variables.

The approaches used to acquire data from the virtual environment and physical rig are similar. The axes of the peg and the hole are aligned first. The hole is then turned at an angle and the peg is moved down until it is jammed inside the hole. In this state, the forces, torques, and the position of the peg are recorded as an input vector of the jammed state and the opposite values of the angles about $X$ and $Y$ axes are recorded as outputs (or actions). It is assumed that $X$ and $Y$ axes are on a plane perpendicular to the original aligned axes. The angles can be represented by the number of steps required to turn the hole ($0.18^\circ$ per step).

When collecting training data during an insertion process, all the possible jammed states in an insertion are considered. Each time the peg is jammed in the hole, the forces, torques, and the position along the $Z$-axis are recorded as an input vector of the jammed state. In the virtual environment, the angles or steps about $X$ and $Y$ axes are always known. In the physical rig, the angles or steps about $X$ and $Y$ axes can be easily calculated if the axis of the peg and that of the hole are aligned at the beginning or at the end of the insertion. Then the opposite values of the angles are recorded as outputs (or actions) as mentioned above.

5.4.2 Modelling of Training Data Using SVM

Incremental Support Vector Machine (SVM) Classification method [Incremental(online)] [Amund03] is used for state classification of training data.

SVM is a new pattern recognition technique based on statistical learning theory [Vapnik95]. Its decision rules reflect the regularities of the training data rather than the incapacities of the learning machine, which guarantees generalisation. SVM allows various other learning machines to be constructed under a unified framework,
which simplifies comparisons and promotes understanding [Chin98]. SVM training also achieves low capacity for a given classification task. The Neural Networks community has recently become very interested in SVM and has successfully applied it to pattern classification [Osuna97] [Schmidt96].

The SVM can construct a nonlinear kernel function to map data from the input space into a possibly high-dimensional feature space and then generalize the optimal hyper-plane with maximum margin between the two classes. Hence, it is basically used for binary (positive or negative) classification or two-class pattern classification [Peng(online)].

For a general two-class pattern classification problem, assuming \( l \) i.i.d. (Independently identically distributed) samples: \((x_1, y_1), \ldots, (x_l, y_l)\), where \( x_i \), for \( i = 1, \ldots, l \) is a feature vector of length \( n \) and \( y_i = \{+1, -1\} \) is the class label for data point \( x_i \), the task is to find a classifier with the decision function \( f(x) \) such that \( y = f(x) \), where \( y \) is the class label for \( x \). For a two-class classifier with oriented hyper-planes, the equation of the optimal hyper-plane separating the two classes can be expressed by \( w \cdot x + b = 0 \), where \( w \) denotes the normal of the hyper-plane, and \( b \) denotes the offset.

The training data samples are represented by \((x_i, y_i), \ldots, (x_l, y_l)\), where \( x_i \in \mathbb{R}^n \) denotes a vector in an \( n \)-dimensional space and \( y_i \in \{-1, +1\} \) represents the class to which \( x_i \) belongs. The decision function of the classifier is \( y = f(x) = sgn (w \cdot x + b) \), i.e.

\[
\begin{cases}
  \text{if } (w \cdot x_i) + b \geq 1 & y_i = 1 \\
  \text{if } (w \cdot x_i) + b \leq -1 & y_i = -1
\end{cases} \iff y_i [w \cdot x_i + b] \geq 1, i = 1, 2, \ldots, l \quad (5-1)
\]

The solution of this classification problem is to find a pair of \( \{w, b\} \) which satisfies Equation 5-1 while maximizing the following value:

\[
Margin(w, b) = \min_{\{y_i = 1\}} \frac{w \cdot x_i + b}{|w|} - \min_{\{y_i = -1\}} \frac{w \cdot x_i + b}{|w|} = \frac{2}{|w|} \quad (5-2)
\]
The classifier with the larger margin will give lower expected risk, i.e. better generalisation. To maximise the margin, which is just $2/|w|$ (see Fig. 5-3), one needs to minimise $|w|$ with constraints in Equation 5-1.

\[
(w \cdot x_i) + b = 0
\]

Support Vectors

\[
(w \cdot x_i) + b = 1
\]

\[
(w \cdot x_i) + b = -1
\]

Margin = $2/|w|$

Fig. 5-3 Decision margin for an oriented hyper-plane classifier

Vapnik [Vapnik95] has proved that $w$ can be expressed by a linear combination of the training samples, as illustrated in Equation 5-3.

\[
w_0 = \sum_{i=1}^l (\alpha_i y_i) x_i \quad (\alpha_i \geq 0), i = 1, \ldots, l
\]  

(5-3)

The objective of training is to obtain each sample’s $\alpha$ value and classify it according to the test sample’s distance to the hyper-plane. Equation 5-4 illustrates the decision function used to classify the test sample $x$.

\[
Label(x) = sgn(w_0x + b_0)
\]  

(5-4)

Usually, only a small portion of samples have non-zero $\alpha_i$ coefficients, whose corresponding $x_i$ (i.e., support vectors) and $y_i$ fully define the decision function. Therefore, the Support Vector set can fully describe the classification characteristics of the entire training set.

This optimisation problem is solved using the Lagrangian formulation. Assume
\[ w(\lambda) = \sum_{i=1}^{l} \lambda_i y_i x_i \]  

(5-5)

Then the solution is to optimize the following problem:

\[
\begin{align*}
\text{Max} \, W(\lambda) &= \sum_{i} \lambda_i - \frac{1}{2} w(\lambda) \cdot w(\lambda) \\
\text{subject to} \quad 0 &\leq \lambda_i \leq C, \sum_i a_i y_i = 0
\end{align*}
\]  

(5-6)

or

\[
\begin{align*}
\text{Min} \left( \frac{1}{2} |w|^2 + C \sum \xi_i \right) \\
\text{subject to} \quad y_i (w \cdot x_i + b) \geq 1 - \xi_i, \xi_i \geq 0
\end{align*}
\]  

(5-7)

where \( \xi_i \) is a slack variable and \( C \) is a given or adjustable parameter.

From the Karush-Kuhn-Tucker (KKT) conditions [SAS(online)] for this convex QP (quadratic programming) problem (convex \(|w|\) and linear constraints), the following conclusion can be drawn [Chin98]:

- if \( \lambda_i = 0 \), then \( y_i (w \cdot x_i + b) \geq 1 \)
- if \( \lambda_i > 0 \), then \( y_i (w \cdot x_i + b) = 1 \)

The training data with nonzero \( \lambda \) values will fall on the +1 or -1 plane (see Fig. 5-3), i.e. these are the Support Vectors (SV) that contribute to defining the decision boundary. If the other data with zero \( \lambda \) values are removed, the training on the Support Vectors (SV) will result in the same decision boundary. The larger the value of \( \lambda \), the more important is the SV and the stronger its influence on the decision boundary.

If the sample space is not linearly separable, a mapping \( \Phi \) can be used to non-linearly transform the input samples into a high dimensional feature space so as to make these samples linearly separable. Thus the decision function is:
\[
\text{sgn}\left(\sum_{i=1}^{l}[(\alpha_i y_i)\Phi(x_i) \cdot \Phi(x) + b]\right) = \text{sgn}\left(\sum_{i=1}^{l}[(\alpha_i y_i)K(x_i, x) + b]\right)
\]

(5-8)

where \(K(x, x_i) = \Phi(x) \cdot \Phi(x_i)\) is known as the kernel function. There are four basic kernels:

- linear: \(K(x, x_i) = x^T \cdot x_i\)
- polynomial: \(K(x, x_i) = (\gamma x^T \cdot x_i + r)^d, \gamma > 0\)
- radial basis function (RBF): \(K(x, x_i) = \exp(-\gamma \|x_i - x_j\|^2), \gamma > 0\)
- sigmoid: \(K(x, x_i) = \tanh(\gamma x^T \cdot x_i + r)\)

where \(\gamma, r\) and \(d\) are kernel parameters.

5.4.3 Incremental Training of the Support Vector Machine

The incremental SVM classifier [Fung02] [Amund03] can modify an existing linear classifier by both retiring old data and adding new data, while in a non-incremental learning algorithm, the new received training data is merged with the previous training data to re-estimate the system model. The Incremental Support Vector Machine (SVM) Classification method has the following advantages [Fung02]:

- Old data can be easily retired while new data can be just as easily incorporated into the classifier.
- It doesn’t need much storage capacity and computational time;
- Extremely large datasets can be classified incrementally;
- Data compression of a large data block of big size into a data block of small size is easily carried out for use by the incremental SVM.

The computational complexity of the training process in SVM -- a quadratic programming problem, is much higher than a linear complexity. Therefore, if the SVM is trained on the SV set instead of the whole training set, the computational
time can be reduced greatly without much loss of the classification precision. This idea can be used in the incremental learning algorithm.

An incremental training scheme similar to the chunking method described in [Boser92] is defined as follows:

1. Divide the training data into segments;
2. Select a segment and retrain the SVM using these data;
3. Discard all the data in the selected segment except the support vectors;
4. Repeat steps 2 and 3 until no segments are left;

In this scheme, once a segment is selected it will not be selected again, whereas in the chunking method, any data point dropped from the chunk can be reselected again. Hence there is a possibility that the final classifier may not classify these data correctly.

Assume $\Gamma$ is a SVM classifier, $Train$ is the training process, $IS$ is the initial training set, $INS$ is the incremental training set, $WS$ is the working set, $BS$ is the backup set, $NS$ is the new set, $DS$ is the discarded set, the subscript $sv$ means support vector and $non-sv$ means non support vector.

An incremental SVM learning algorithm [Syed99], which uses only the historical SV samples ($WS$) and the incremental training samples ($INS$) in re-training, is illustrated as follows:

1. $\Gamma=Train(IS)$, $WS=IS_{sv}$;
2. $WS=WS \cap INS$;
3. $\Gamma=Train(WS)$, $WS=WS_{sv}$.

An alternative incremental SVM learning algorithm [Xiao00] based on the boosting idea can be illustrated as follows.

1. $\Gamma=Train(IS)$, $WS=IS_{sv}$, $BS=IS_{non-sv}$
2. $NS=INS$, firsttime=TRUE
3. Classify($NS$) by $\Gamma$, $WS=NS_{err} \cup WS$, $BS=NS_{ok} \cup BS$
4. If $p > \beta$ then Stop
Step 5. If firsttime then DS=Discard,(BS), BS=BS-DS, firsttime=FALSE
Step 6. Γ=Train(WS), WS=WSsv, NS=BS, BS=WSnon-sv
Step 7. Goto Step3

This method classifies the incremental training set (NS) into correct samples (NSok) and wrong samples (NSerr) using the previously trained classifier (Γ). If the classification precision $p$ is higher than a threshold $β$, the training is stopped. The redundant samples which are far from the hyper-plane, are discarded from BS according to a discarding rate $τ$ to reduce the re-classification time and storage space. The discard process $Discard,(BS)$ is executed only once after the first time re-classification [Peng(online)].

5.4.4 Multi-Class Support Vector Machine

The $K$-class pattern classification problem is to find a classifier with the decision function $f(x)$ such that the class label for $x$ is $y = f(x)$, given $l$ i.i.d. (Independently identically distributed) samples: $(x_1, y_1), \ldots, (x_l, y_l)$, where $x_i$, for $i=1, \ldots, l$ is a feature vector of length $n$ and $y_i = \{1, \ldots, k\}$ is the class label for data point $x_i$.

SVM is basically a binary classifier. Several different schemes can be applied to the basic SVM algorithm to handle the $K$-class pattern classification problem [Hastie96] [Weston98], as described in the following subsections.

5.4.4.1 One-Against-the-Rest Classifiers

This scheme constructs $K$ classifiers, one for each class. The final classification decision on the $K$-class classification (Equation 5-9) is the combination of all the classifiers, where the $K$th classifier classifies the training data of class $k$ against all other training data.
\[ f(x) = \arg \max_k \sum_{i=1}^{J} \left( c_i^k \sum_{m=1}^{K} \lambda_i^m - \lambda_i^k \right) (x_i \cdot x) + b_k \]  

(5-9)

where \( \arg \max \) returns the index of a maximal element of a list. The decision rule is the largest of the decision functions at any \( x \).

5.4.4.2 One-Against-One or Pairwise Classifiers

The schemes require a binary classifier for each possible pair of classes but only one of \( y_1 \)-to-\( y_2 \) and \( y_2 \)-to-\( y_1 \) SVM classifiers is needed. Hence, the total number of classifiers for a \( K \)-class problem is \( K(K-1)/2 \). As shown in Fig. 5-4, the total number of classifiers for a 4-class problem is \( 4(4-1)/2=6 \). The training data for each classifier only contains the data for the two involved classes. To give the final classification results, the classifiers will be combined with some voting scheme which needs the pairwise probability, i.e. the probability of \( x \) belong to class \( i \) given that it can only belong to class \( i \) or \( j \) (Equation 5-10).

\[ P(i \mid i, j) = P(y = i \mid y = i \text{ or } y = j, x) = \frac{P(y = i)}{P(y = i) + P(y = j)} \]  

(5-10)

The pairwise probability can be computed directly using two Gaussian density functions, one for each class, for every pairwise classifier [Hastie96] (Equation 5-11).

\[ P(i \mid i, j) = \frac{g_i(x)}{g_i(x) + g_j(x)} \]  

(5-11)

where

\[ g_i(x) = \phi(m_i, \sigma_i) \]  

(5-12)

\[ g_j(x) = \phi(m_j, \sigma_j) \]  

(5-13)

g_i(x) is the p.d.f. (Probability Density Function) for class \( i \) estimated from the output values of the decision function, \( f(x) \), for all \( x \) in that class. \( \phi(m, \sigma) \) is the Gaussian
density with mean $m$ and standard deviation $\sigma$. $m_i$ and $\sigma_i$ are the mean and standard deviation of $f(x)$ respectively ($x$ is in class $i$).

The pairwise probability can be also estimated by using a pseudo p.d.f. (eg, $e^{\phi(x)}$) which satisfies the following factors:

- The classification boundary of a SVM classifier is at $f(x)=0$, and the larger the magnitude of the decision function, the higher the classification confidence.

- The pairwise probability is a sigmoid function with values from 0 to 1 and the decision boundary is the point where the value of the pairwise probability is 0.5.

---

These estimated pairwise probabilities ($r_{ij}$) of $p(i|i,j)$ can be combined in the following ways to vote for the most probable class [Hastie96]:

**Method 1**
Assign the unknown data points to the class that wins the most pairwise comparisons [Friedman96]:

$$f_{\max}(x) = \arg \max_i \sum_{j \neq i} I(r_{ij} > r_{ji})$$

(5-14)

where,

$$I(a > b) = \begin{cases} 
1 & \text{if } a > b, \\
0 & \text{otherwise}
\end{cases}$$

(5-15)

**Method 2**

The decision rule for the $K$-class problem (Equation 5-16) labels $x$ with the class with the largest posterior probability of class $i$, $\bar{P}_i$, obtained using a simple non-iterative estimation from the averages of $r_{ij}$,

$$f_{\text{avg}}(x) = \arg \max_i \bar{P}_i$$

(5-16)

$$\bar{P}_i = \frac{2}{K(K-1)} \sum_{j \neq i} r_{ij}$$

(5-17)

where the estimates (Equation 5-17) are derived as an approximation to Equation 5-18, by replacing $P_i + P_j$ with $2/K$ and $p(i|i,j)$ with $r_{ij}$.

$$P_i = \sum_{j \neq i} \left( \frac{P_i + P_j}{K-1} \right) \left( \frac{P_i}{P_i + P_j} \right)$$

(5-18)

**Method 3**

Similar to the previous method, $x$ will be labelled as the class with the highest estimated value of a set of posterior probabilities $\hat{P}_i$

$$f_{\text{opt}}(x) = \arg \max_i \hat{P}_i$$

(5-19)

where $\hat{P}_i$ are compatible with the given $r_{ij}$ (Equation 5-20)
\[
\hat{P}_i = \frac{\hat{P}_i}{\hat{P}_i + \hat{P}_j} 
\] for \( i, j = 1, \ldots, K \) and \( i \neq j \)

(5-20)

It may not be possible to find an exact solution for this problem with \( K-1 \) free variables (also called independent variables) and \( K(K-1)/2 \) equations. However, the closest solution can be estimated by using an iterative algorithm [Bradley52] [Hastie96] that minimises the weighted Kullback-Leibler distance between \( r_{ij} \) and \( p(i|i,j) \):

\[
KL = \sum_{i \neq j} n_{ij} r_{ij} \log \frac{r_{ij}}{p(i|i,j)}
\]

(5-21)

where \( n_{ij} \) is the number of training data in the \( i \)th or \( j \)th class.

Generally speaking, the Kullback-Leibler distance of an estimated model \( \hat{\theta} \) from the true model \( \theta \) is given by Equation 5-22. KL (Kullback-Leibler) distance measures the “distance” (in bits usually) between two probability distributions. The interpretation is that, if one of the distributions is the “true” distribution, the KL distance is the number of extra bits to be used if the variable is coded using the second distribution. It is non-symmetric, which will get different answers if the “true” distribution and the other one are swapped.

\[
KL(\theta, \hat{\theta}) = \sum_{i} n_i \log \frac{\theta_i}{\hat{\theta}_i} \text{ nits}
\]

(5-22)

5.4.4.3 K-Class SVM

This approach [Weston98] constructs the decision function by considering all classes at once. It is similar to the \( K \) 1-to-rest classifiers method. One decision function will be constructed for each class but the optimisation problem is generalised to consider all the decision functions at once. The decision function for the correct class can just be greater than the rest of the decision function by a margin of 2, instead of giving
zero value at the decision boundary (Equation 5-23 subject to the constraints in Equation 5-24).

\[
L(w,r) = \frac{1}{2} \sum_{m=1}^{K} (w_m \cdot w_m) + C \sum_{i=1}^{l} \sum_{m \neq y_i} \xi_i^m
\]

(5-23)

\[
(w_m \cdot x_i) + b_i \geq (w_m \cdot x_i) + b + 2 - \xi_i^m
\]

\[
\xi_i^m \geq 0, i = 1, \ldots, l \quad m \in \{1, \ldots, K\} \setminus y_i
\]

(5-24)

The class label is the class whose decision function gives the largest output value:

\[
f(x) = \arg \max_k (w_k \cdot x + b_k) \quad k = 1, \ldots, K
\]

(5-25)

Lagrangian and primal dual formulation can be used to solve the optimisation problem in the formulation above by reformulating the above problem to a QP (Quadratic Programming) problem of maximising Equation 5-26 subject to the linear constraints in Equation 5-27.

\[
W(\lambda) = 2 \sum_{i,m} \lambda_i^m + \sum_{i,j,m} \left[-\frac{1}{2} c_i^n \sum_{p=1}^{K} \lambda_i^p \sum_{q=1}^{K} \lambda_j^q + \lambda_i^n \lambda_j^n - \frac{1}{2} \lambda_i^n \lambda_j^n \right](x_i \cdot y_j)
\]

(5-26)

\[
\sum_{i=1}^{l} \lambda_i^n = \sum_{i=1}^{l} c_i^n \sum_{m=1}^{K} \lambda_i^m \quad n = 1, \ldots, K
\]

\[
0 \leq \lambda_i^n \leq C \quad \text{and} \quad \lambda_i^n = 0
\]

(5-27)

\[
i = 1, \ldots, l \quad m \in \{1, \ldots, K\} \setminus y_i
\]

where,

\[
c_i^n = \begin{cases} 
1 & \text{if } y_i = n \\
0 & \text{if } y_i \neq n
\end{cases}
\]

(5-28)

The function for labelling an unknown data point is then:

\[
f(x) = \arg \max_k \left( \sum_{i=1}^{l} c_i^n \sum_{m=1}^{K} \lambda_i^m - \lambda_i^n \right)(x_i \cdot x) + b_k
\]

(5-29)

This new formulation does not offer significant advantage over the $K$ 1-to-rest classifier combination method. The reformulation is achieved by combining $K$ QP
problems of size $l$ into a QP problem of size $K*l$. The main disadvantage of this formulation is that the QP size becomes very large.

5.4.4.4 K-Class Linear Programming SVM

This method [Weston98] constructs $K$ decision functions for each class as the previous method, though each of these decision functions is a linear combination of all the training data points belonging to that particular class (Equation 5-30). A test data point is labelled as the class with the maximum output value from its decision function (Equation 5-31).

$$f_i(x) = \sum_{j,y=1}^{l} \lambda_{ij} K(x_i \cdot x_j) + b_k$$  

(5-30)

$$f(x) = \arg \max_k f_k(x)$$  

(5-31)

The training of this SVM is to minimise Equation 5-23 subject to constraints in Equation 5-24, while $w_m$ is defined as Equation 5-32 with a simple dot product as its kernel.

$$w_m = \sum_{j,y=1}^{l} \lambda_{ij} x_i$$  

(5-32)

The QP size is $l$ and there are still $K$ bias values to solve using least mean squares. The training cost is much more tractable than that of the first approach [Chin98].

5.4.5 An Example of Acquiring Type 1 Skills

In order to illustrate the approach, an example of collected jamming states and the corresponding optimal actions are illustrated in Fig. 5-5 to Fig. 5-7. A peg is considered jammed when magnitude of one of the measured force components is larger than 100uf (5.6448N) (see Section 6.3 in Chapter 6). The measured force and torque components are shown in Fig. 5-5 and the position of the peg along the Z-axis
are shown in Fig. 5-6. All samples are jamming states. The actions (Fig. 5-7) with the opposite value of the steps turned by the stepper motors away from the aligned states can overcome the corresponding jamming state in one attempt. The sample points in Fig. 5-5 represent samples from probably different insertion processes while the sampling points in Fig. 3-19 are obtained from one insertion process.

**Fig. 5-5**  Forces/torque of jamming states

**Fig. 5-6**  Position of peg at jamming states (measured in the number of pulses)
Fig. 5-7  Actions at jamming states

A segment of the collected training data in this experiment is tabulated in Table 5-1. The number of steps ranges from -15 to 15. The inputs are normalized $F_x$, $F_y$, $F_z$, $M_x$, $M_y$ and $Pos$ labelled by 1, 2, 3, 4, 5 and 6 respectively. The normalized values are between 0 and 1. The normalization formulas are provided in Section 6.3 in Chapter 6.

Two models which classify $x\_step$ and $y\_step$ as illustrated in Fig. 5-8 (a) and (b) are obtained from the training data given in Fig. 5-5 ~ Fig. 5-7. The training data recorded a number of jamming states and the corresponding actions (Fig. 5-7) which drove the peg from the jamming states to the fully-inserted states. The original corrective steps from the training data and the steps generated by the estimated models for $X$ and $Y$ axes are compared in Fig. 5-9 and Fig. 5-10 respectively.

The corresponding cross-correlation values between the estimated and actual values of the corrective steps are shown in Fig. 5-11 and Fig. 5-12 for $x\_step$ and $y\_step$ respectively. The similarities between the estimated and actual values of the corrective steps strongly indicate the validity of the approach and the accuracy of the
models. The cross-correlation is motivated by the distance measure (squared Euclidean distance)

\[ d_{i,j}^2(u) = \sum (f(x) - t(x - u))^2 \]  

(5-33)

In the physical manipulation rig, the actions are applied directly when the estimated angles are known. Let’s assume that the axis of the hole is aligned with the Z-axis at the start of the insertion. The hole is turned \( A \) steps about \( X \)-axis and \( B \) steps about \( Y \)-axis while the peg is pushed down until it is jammed. The jamming state is overcome when the axes of the hole and the \( Z \)-axis are realigned. Naturally this occurs when the hole is turned \(-A\) steps about \( X \)-axis and \(-B\) steps about \( Y \)-axis.

Table 5-1  Training data

<table>
<thead>
<tr>
<th>( x_{step} ), normalized inputs</th>
<th>( y_{step} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 1:0.308108 2:-0.417085 3:-0.748148 4:0.588589 5:0.339652 6:0.572862</td>
<td>-15</td>
</tr>
<tr>
<td>8 1:0.362162 2:-0.396985 3:-0.422222 4:0.552553 5:0.402844 6:0.639502</td>
<td>-14</td>
</tr>
<tr>
<td>1:0.448649 2:-0.407035 3:-0.6 4:0.585586 5:0.469194 6:0.751104</td>
<td>-13</td>
</tr>
<tr>
<td>8 1:0.589189 2:-0.376884 3:-0.377778 4:0.522523 5:0.592417 6:0.831393</td>
<td>-12</td>
</tr>
<tr>
<td>1:0.772973 2:-0.366834 3:-0.348148 4:0.51952 5:0.78831 6:0.957447</td>
<td>-11</td>
</tr>
<tr>
<td>8 1:0.902703 2:-0.316583 3:0.496296 4:0.393393 5:0.911532 6:1</td>
<td>-10</td>
</tr>
<tr>
<td>1:0.891892 2:-0.306533 3:0.585185 4:0.375375 5:0.902054 6:0.995183</td>
<td>-9</td>
</tr>
<tr>
<td>8 1:0.924324 2:-0.306533 3:0.555556 4:0.36036 5:0.921011 6:0.998394</td>
<td>-8</td>
</tr>
<tr>
<td>8 1:0.902703 2:-0.296482 3:0.57037 4:0.336336 5:0.892575 6:0.961461</td>
<td>-7</td>
</tr>
<tr>
<td>8 1:0.891892 2:-0.256281 3:0.659259 4:0.285285 5:0.898894 6:0.857086</td>
<td>-6</td>
</tr>
<tr>
<td>8 1:0.891892 2:-0.21608 3:0.748148 4:0.252525 5:0.902054 6:0.789643</td>
<td>-5</td>
</tr>
<tr>
<td>8 1:0.913514 2:-0.20603 3:0.792593 4:0.213213 5:0.908373 6:0.501405</td>
<td>-4</td>
</tr>
<tr>
<td>8 1:0.902703 2:-0.175879 3:0.911111 4:0.195195 5:0.908373 6:0.380169</td>
<td>-3</td>
</tr>
<tr>
<td>8 1:0.913514 2:-0.135678 3:0.955556 4:0.138138 5:0.905213 6:0.176234</td>
<td>-2</td>
</tr>
<tr>
<td>8 1:0.924324 2:-0.085427 3:0.955556 4:0.0900901 5:0.92733 6:0.0525893</td>
<td>-1</td>
</tr>
<tr>
<td>8 1:0.891892 2:-0.0653266 3:0.94074 4:0.0870871 5:0.898894 6:-0.0325171</td>
<td>0</td>
</tr>
<tr>
<td>8 1:0.891892 2:-0.0452261 3:0.985185 4:0.0609691 5:0.895735 6:-0.124047</td>
<td>1</td>
</tr>
<tr>
<td>15 1:0.956575 2:-0.949749 3:0.555556 4:0.933934 5:0.968404 6:-0.457246</td>
<td>-15</td>
</tr>
<tr>
<td>15 1:0.924324 2:1 3:0.688889 4:1 5:0.92733 6:-0.460458</td>
<td>-14</td>
</tr>
<tr>
<td>15 1:0.589189 2:-0.236181 3:-0.585185 4:0.345345 5:0.649289 6:-0.242874</td>
<td>-13</td>
</tr>
<tr>
<td>15 1:0.654054 2:-0.256281 3:-0.866667 4:0.36036 5:0.728278 6:-0.220393</td>
<td>-12</td>
</tr>
<tr>
<td>15 1:0.708108 2:-0.246231 3:-0.748148 4:0.324324 5:0.759874 6:-0.21397</td>
<td>-11</td>
</tr>
<tr>
<td>15 1:0.794595 2:-0.20603 3:1 4:0.294294 5:0.864139 6:-0.190686</td>
<td>-10</td>
</tr>
<tr>
<td>……</td>
<td>……</td>
</tr>
</tbody>
</table>
Fig. 5-8 Two models of SVM classification

Fig. 5-9 Performance of Model x

Fig. 5-10 Performance of Model y
If the values of $A$ or $B$ are too large, the hole can be turned to some middle states before it reaches its final stage. Due to passive compliance in the mechanism and the gap between the peg and the hole, the final state can vary up to 5 steps.
5.5 On-Line Learning

5.5.1 Method for Learning Skills of Type 2

Skills of Type 2 are learned through the Locally Weighted Projection Regression (Lwpr) [Lwpr(online)] algorithm, which is an on-line incremental learning algorithm. In this approach, new skills are acquired from the new training data. The system model can be conveniently loaded, updated and saved during the learning process. In this application, a human operator does not have any understanding of the exact angles of the hole and peg during the insertion process. The feedback is provided through the senses of touch and vision. The Lwpr algorithm learns the human actions directly. In addition to the data obtained in the virtual environment, the information produced by the physical experimental rig is also used in the learning process.

5.5.2 Locally Weighted Learning

Locally Weighted learning (LWL) is a type of memory-based learning which uses locally weighted training to average, interpolate between, extrapolate from, or combine training data. It fits the training data only in a region around the location of the query [Vapnil92]. Memory-based learning stores the training data in memory, and finds relevant data in the database to answer a particular query.

Assume the model generating data for the regression problem is \( y = f(x) + \varepsilon \), where \( x \in R^n \) is an \( n \)-dimensional input vector, the noise term has zero mean, \( E\{\varepsilon\} = 0 \), and \( y \) is a one dimensional output. LWL methods approximate nonlinear functions by means of piecewise linear models [Cleveland79], similar to a first-order Taylor series expansion. Locally linear models are an excellent statistical compromise among the possible local polynomials that can be fitted to a set of data [Hastie93].
Assume \( \hat{y} \) is the prediction of the true function \( f(x) \) given the n-dimensional input vector \( x \) and \( \phi_k(x) \) is a set of vector valued nonlinear basis functions. Function approximation using linear parameterisation with nonlinear basis functions can be formalized as

\[
\hat{y} = \hat{f}(x) = \sum_{k=1}^{N} \phi_k^T(x)\hat{\theta}_k
\]  

(5-34)

The parameters \( \hat{\theta}_k \) are estimated from data which arrive as pairs of \((x_i, y_i)\) or as \((x_i, e_i)\), where \( e_i \) approximates the prediction error \( e_{p,i} = f(x_i) - \hat{f}(x_i) \).

Usually, the learning algorithm minimizes the least squares criterion

\[
J = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2
\]  

(5-35)

over all \( m \) available training data points. Thus each basis function in Equation 5-34 contributes a fraction towards reducing the approximation error, while locally weighted learning (LWL) [Atkeson97] advocates that each basis function and its parameters should be considered as an independent local model that minimizes the locally weighted error criterion [Nakanishi03]

\[
J_k = \sum_{i=1}^{m} w_{k,i} (y_i - \hat{y}_{k,i})^2 \quad \text{where} \quad \hat{y}_{k,i} = \phi_k^T(x_i)\hat{\theta}_k \quad \text{and}
\]

(5-36)

\[
w_{k,i} = \exp(-0.5(x_i - c_k)^T D_k (x_i - c_k))
\]

The region of validity of the model, called a receptive field (see Fig. 5-13), is calculated by Gaussian kernel [Atkeson97]:

\[
w_k = \exp(-0.5(x - c_k)^T D_k (x - c_k))
\]  

(5-37)

where \( c_k \) is the centre of the \( k \)th linear model, and \( D_k \) is a positive semi-definite distance metric that determines the size and shape of the region of validity of the linear model.
5.5.3 Locally Weighted Regression

Locally Weighted Regression (LWR) [Atkeson89] [Atkeson92] [Atkeson95] is a memory-based method with locally linear models that perform a regression around a point of interest using only training data that are local to that point. Model-based methods such as neural networks use the training data to build a parametric model for predictions and then discard the training data afterwards. Memory-based methods, on the other hand, are non-parametric, explicitly retain the training data and use it for prediction when required [Lwr(online)].

An LWR can be quickly trained by adding new training data to the memory or the old training data. The method was proven to be suitable for real-time control by constructing an LWR-based system learning a difficult juggling task [Schaal94].

For Locally Weighted Regression, $\phi_k(x)$ equals $[x^T \ 1]^T$ in Equation 5-33 with $w_k$ defined by Equation 5-36. The following weighted regression analysis is performed.
when a prediction is needed for a query point $x_q$ and $p$ training data sets \{$x_n$, $y_i$\} in memory:

a. compute diagonal weight matrix $W$,

$$w_{ii} = \exp\left(-\frac{1}{2}(x_i - x_q)^T D(x_i - x_q)\right)$$

b. build matrix $X$ and vector $y$ such that $y = (y_1, y_2, \ldots, y_p)^T$

$$X = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_p)^T \text{ where } \tilde{x}_i = [(x_i - x_q)^T, 1]^T$$

c. compute locally linear model $\beta = (X^T W X)^{-1} X^T W y$

d. the prediction for $x_q$ is $\hat{y} = \beta_{n+1}$

$\beta_{n+1}$ is the $(n+1)$th element of the regression vector $\beta$. Normally most of the $p$ training data points receive an approximately zero weight when they are too far away from the query point. This significantly reduces the computational complexity of LWR [Moore90]. Hence, LWR can be applied efficiently in real-time for problems that are not too high dimensional in the number of inputs $n$ and that do not accumulate too much data in one particular area of the input space.

The distance metric $D$ can be optimized by leave-one-out cross validation. To avoid too many open parameters, $D$ is usually assumed to be a global diagonal matrix $D = h \cdot \text{diag}( [n_1, n_2, \ldots, n_n] )$, where $h$ is a scale parameter, and the $n_i$ normalizes the range of the input dimensions, e.g., by the variance of each input dimension $n_i = 1/\sigma^2$.

Leave-one-out cross validation is performed as a one-dimensional search over the parameter $h$ for a set $H$ of reasonable values $h_i$:

For all $h_i$:

$sse_i = 0$

For $i = 1:p$

a. $x_q = x_i$

b. temporarily exclude \{$x_i$, $y_i$\} from training data

c. compute LWR predication $\hat{y}_q$ with reduced data
d. \( sse_x = sse_x + (y_i - \hat{y}_i)^2 \).

End
End

Then choose the optimal \( h^{\circ}_x \) such that \( h^{\circ}_x = \min \{ sse_x \} \)

At an increased computational cost, leave-one-out cross validation can be also performed treating all coefficients of the distance metric as open parameters. This is usually achieved by using gradient descent methods [Atkeson92] [Atkeson95] [Lowe95] to learn \( D_k \) in a locally weighted leave-one-out cross validation criterion that approximates the statistical expectation of the cost function (Equation 5-33) [Schaal98].

5.5.4 Locally Weighted Projection Regression

If the number of input dimensions grows large, or if there are redundant input dimensions such that the matrix inversion in LWR becomes numerically unstable, Partial Least Squares Regression (PLS) [Wold75] [Frank93] [Vijayakumar00] can be used to reduce the computational complexity of LWR and to avoid numerically unstable problems. If the input data is locally statistically independent (i.e., has a diagonal covariance matrix) and is approximately locally linear, LWPLS (Locally Weighted Partial Least Squares Regression) will find an optimal linear approximation for the data with a single projection [Vijayakumar00].

If the learning system receives a large, possibly never ending stream of input data, both memory requirements to store all the data as well as the computational cost for LWR and LWPLS become too large. Under these circumstances, a non-memory based incremental LWL algorithm [Schaal98] can be used to build local models continuously in the entire support area of the input data at selected points in input space instead of postponing the computation of a local linear model until a prediction needs to be made. Incrementally updating the parameters of the linear models can be
accomplished with recursive least squares techniques [Ljung86]. Thus, the LWR algorithm becomes the Receptive Field Weighted Regression (RFWR) algorithm [Schaal98].

However, RFWR becomes computationally too expensive in high dimensional spaces. For such cases, LWPLS can be reformulated as an incremental algorithm, called Locally Weighted Projection Regression (LWPR). For a training point \((x, y)\), the update equations are:

Update the means of inputs and output:

\[
x^{n+1}_0 = \frac{\lambda W^n x^n_0 + wx}{W^{n+1}}
\]

\[
\beta^{n+1}_0 = \frac{\lambda W^n \beta^n_0 + wy}{W^{n+1}}
\]

where \(W^{n+1} = \lambda W^n + w\)

Update the local model:

\[\text{Initialize: } x_0 = x - x^{n+1}_0, res_0 = y - \beta^{n+1}_0\]

For i= 1, r

a) \(u^{n+1}_i = \lambda u^n_i + wz_i - res_i - 1\)

b) \(s_i = z_i^T u^{n+1}_i\)

c) \(SS_i^{n+1} = \lambda SS_i^n + w s_i^2\)

d) \(SR_i^{n+1} = \lambda SR_i^n + ws_i^2 res_i - 1\)

e) \(SZ_i^{n+1} = \lambda SZ_i^n + wz_i - s\)

f) \(\beta_i^{n+1} = SR_i^{n+1} / SS_i^{n+1}\)

g) \(p_i^{n+1} = SZ_i^{n+1} / SS_i^{n+1}\)

h) \(z_i = z_i - s p_i^{n+1}\)

i) \(res_i = res_i - 1 - s \beta_i^{n+1}\)

j) \(SSE_i^{n+1} = \lambda SSE_i^n + w res_i^2\)

where SS, SR, and SZ are memory terms that enable us to achieve the univariate regression in step f) in a recursive least squares fashion, i.e., a fast Newton-like method as in RFWR.

LWPR Gradient descent update of \(D\) is shown in [Lwpr(online)]. The decomposition of \(D\) can be achieved with a Cholesky decomposition [Press89]. The computational
complexity of the update of all parameters of LWPR is linear in the number of input dimensions, even for very high dimensional input spaces.

5.5.5 An Example of Acquiring Type 2 Skills

The on-line incremental learning system can learn from the new incoming data in real time. It can produce the corrective actions in terms of rotation of the hole about $X$ and $Y$ axes according to the measured forces, torque and the peg position using the updated Lwpr algorithm model. The Lwpr algorithm is a multi-input (state) and multi-output (action) model as shown in Fig. 5-14.

The process of acquiring Type 2 skills is illustrated in this section through an example. In order to generate the required training data from the virtual environment, the axes of the hole and the peg were misaligned over the class range of (5,5) to (2,-5), resulting in 44 classes as illustrated in Table 5-2. At each class, the generated forces, torque and the peg position were recorded as inputs to the learning algorithm while the opposite values of the corrective actions were recorded as outputs. The Lwpr learning algorithm introduced in Section 5.5.4 was applied to the data to estimate the initial skill model.

In the first stage, the inputs from the training data were used in the model to produce outputs. In Fig. 5-15, the $y_{\text{steps}}$ (normalised) produced from the skill model are compared with $y_{\text{steps}}$ (normalised) of the original data. The corresponding cross-correlation values between the estimated and actual values of $y_{\text{steps}}$ are shown in Fig. 5-16. The results show close correlation between two sets of outputs.

In the second stage, a new set of training data was used to update the model. The updated model was then applied to the inputs from the old set of data. The correlation between the $y_{\text{steps}}$ (normalised) produced from the updated model and $y_{\text{steps}}$ (normalised) of the old training data is illustrated in Fig. 5-17. The
corresponding cross-correlation values are shown in Fig. 5-18. The comparison show that the information learned from the old data has not been lost in the model when it was updated with a new set of data.

![Lwpr algorithm model](image)

**Fig. 5-14  Lwpr algorithm model**

**Table 5-2  Pairs of angles (measured in step) between the hole and the peg**

<table>
<thead>
<tr>
<th>x_step</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>y_step</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
</tr>
<tr>
<td>x_step</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>y_step</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
</tr>
<tr>
<td>x_step</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>y_step</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
</tr>
<tr>
<td>x_step</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>y_step</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
</tr>
</tbody>
</table>

![First learning results (normalised)](image)

**Fig. 5-15  First learning results (normalised)**
Fig. 5-16  Cross correlation between original steps and output steps of the first learning results

Fig. 5-17  Second learning results with the model updated by the new data (normalised)
5.6 Chapter Summary

The focus of this chapter was on the acquisition of the manipulation skills. An overview of classical learning methods was provided. Learning methods used in this work to develop skill models including Incremental Multi-class Support Vector Machine (off-line), Locally Weighted Projection Regression (on-line) were introduced. The effectiveness of the approach was illustrated through some examples.
CHAPTER 6   VALIDATION

6. 1 Introduction
The methodologies developed in this work are validated in this chapter. Initially the experimental rig and experimental settings will be described. Then the experimental procedure designed to illustrate the performance of the system will be explained. Finally the results obtained will be presented and a critical analysis of them will be carried out.

6. 2 Experimental Rig
The experiment rig is designed to decouple the control and operation of the hole and the peg. The hole has two degrees of freedom (the pitch and yaw angles) and is controlled by two stepper motors. The peg has one degree of freedom of translation along the axis of the peg and is controlled by a DC motor (see Fig. 6-1). These 3 DOF (Degrees of Freedom) are sufficient for study of the insertion phase of the process. The radius of the peg is 10mm and that of the hole is 10.05mm. This provides a clearance of 0.05mm between the peg and the hole. The peg is attached to a 6 DOF force sensor as introduced in Section A.2.7.

The block diagram of the control system driving and controlling the experimental rig is shown in Fig. 6-2. The DAQ-802 board is an effective high speed data acquisition board for IBM® compatible ISA (Industry Standard Architecture) bus applications. It is used to send signals to control servo motor and stepper motors, and read signals from the decoder. Two stepper motor drive boards are used to control the stepper motors. Force sensor communicates the computer through COM1 port.

The signal generated by the decoder and the signal controlling the servo motor are read into the computer by DAQ802 I/O interface card. The axis of the peg and that of
the hole are not exactly aligned. Hence, there are still outstanding forces and torques when the peg is fully inside the hole.

(a) Photo

(b) Diagram

Fig. 6-1 Real rig

Fig. 6-2 Block diagram of the experimental rig
To avoid backlash of the gearbox, high torque stepper motors and micro-step drives are used to provide high torque and small steps without the help of gearboxes. Each step represents 0.18°, which is the minimum resolution the equipment can provide. The whole system is programmed using Visual C++. Windows API (Application Programming Interface) functions are used for serial port communication between the computer and the force sensor, which can work under Windows NT/2000 and Windows XP. Standard I/O functions such as `Inp` and `Outp` are used to directly access I/O ports for communication between the computer and the DAQ board. The standard I/O functions can only work under Windows 9x & ME, since Windows NT/2000 and Windows XP have strict control over I/O ports for security reasons. The system has been set up accordingly.

6.3 Experimental Set Up

The physical manipulation rig provides linear insertion motion for the peg and tilting motion about X and Y axes for the hole. The peg is driven fast when the magnitudes of the forces measured during insertion is small. The position of the peg is measured by the number of pulses generated by the encoder attached to the actuator driving the peg. A magnitude of 50 pluses of the encoder is equivalent to a displacement of 0.1 mm. The position of the peg “Pos” is represented by a value between 0 and 50000. The larger the value of Pos, the deeper is the peg inside the hole. The tilting motion is measured by the number of steps of the stepper motors driving the hole. As mentioned before, the angular displacement per step is 0.18°. The maximum absolute value of steps for setting the initial angles is chosen to be 25 which is equivalent to an initial angle of 4.5° (25 × 0.18° = 4.5°).
The force unit of the force sensor is “uf”. One uf is equivalent to 0.056448N. The maximum force allowed is 8.5N in Phantom 1.0 and 6.4N in Phantom 1.5. Hence, when a force signal exceeds 100uf (5.6448N) during experimental work, the peg is stopped and then moved slightly back (about 0.4 mm) waiting for the system’s reaction to align the axis of the hole with the peg’s. The maximum force during physical experiments will not exceed 150uf (8.4672N), which is just under the limit of Phantom 1.0.

The general equation for normalising a variable \( X \) into normalised \( X_n \) in the range \([0, 1]\) is:

\[
X_n = \frac{(X - X_{\text{min}})}{(X_{\text{max}} - X_{\text{min}})}
\]

where \( X_{\text{min}} \) can be smaller than the minimum value of all \( X \) in the training data and \( X_{\text{max}} \) can be bigger than the maximum value of all \( X \) in the training data to ensure that all \( X_n \) falls within \([0, 1]\). \( X_{\text{max}} \) is set to be 150 uf which is well above the jamming state threshold of 100uf. A value of 500 uf·cm is chosen as the maximum magnitude for torques. Initially, the peg is driven down until the value of \( Pos \) is 26000 at which the peg is placed just above the hole. If the \( Pos \) value is greater than 47000, the insertion is considered complete. Hence, the minimum number of pulses received from the encoder is chosen to be 26000 and the maximum is 50000 which is slightly larger than 47000.

Accordingly, in on-line learning, the following normalisation equations are used:

\[
\begin{align*}
F_X &= (F_X + 150.0)/(300) \\
F_Y &= (F_Y + 150.0)/(300) \\
F_Z &= (F_Z + 150.0)/(300) \\
M_X &= (M_X + 500.0)/(1000) \\
M_Y &= (M_Y + 500.0)/(1000) \\
Pos &= (Pos - 26000.0)/(50000 - 26000) \\
x_{\text{step}} &= (x_{\text{step}} + 30)/60 \\
y_{\text{step}} &= (y_{\text{step}} + 30)/60
\end{align*}
\]
Relationships defined by Equation 6-2 are used for both SVM and LWPRS algorithms. Since $x_{\text{step}}$ and $y_{\text{step}}$ are directly employed in SVM algorithm as classification labels without normalization or transformation, Equation 6-3 is exclusively used for LWPRS algorithm.

Assuming that the viewer is facing the rig as shown in Fig. 6-4, turning the hole around the $X$-axis clockwise means that the value of $x_{\text{step}}$ is negative and the hole is turned forward, while turning the hole around the $Y$-axis anti-clockwise means that the value of $y_{\text{step}}$ is negative and the hole is turned to the left hand side.

The axis of the force/torque transducer and that of the hole are displaced by $45^\circ$ (see Fig. 6-4). Hence, the forces/torque data obtained from the virtual environment is transformed by $45^\circ$ about the axis of the hole to match the forces/torque obtained from the transducer in the physical rig. The anti-clockwise rotation of the motor mounted on the $X$-axis of the hole is considered positive whereas for the motor mounted on the $Y$-axis, the clockwise rotation is positive.

The angle of the axis of each motor is initialized on powering up the stepper motor drives. On the power up, the axis of the peg may not be fully aligned with that of the hole. The alignment can be achieved by performing a successful insertion first.
The first channel analogue output of the DAQ-802 board (Pin 9, DA0 Out, see Fig. A-9) should be initially set to zero to prevent the peg from moving when the power of the servo motor drive is just turned on.

In order to derive force/torque information from the force sensor, the Bias Sensor command “BS” (see Section A.2.7) is issued to initialize the force sensor. The OR command (see Section A.2.7) which selects the Output Record mode is issued to begin the transmission of one record of F/T data over the serial port. The force sensor can stop operation due to the sequencing error “out of sequence”. This can be overcome by initializing the force sensor.

6.4 Experimental Results

Type 1 skills use the forces and torques measured at the jamming states as inputs while the inputs of Type 2 skills use the forces and torques at states where the position of the peg is up to 0.4mm above the jamming state along Z-axis. Consequently the magnitudes of the forces in these states are usually smaller than the jamming states. Table 6-1 shows typical magnitudes of forces and torques utilised in acquiring skills of types 1 and 2.
Experimental work shows that Type 1 skills have higher success rate than Type 2, provided that sufficient learning from the virtual manipulation environment has taken place. The data from a successful insertion process using Type 1 skills is provided in Table 6-2. In this case, the corrective action class (10, -16) generated from Type 1 skills is opposite to the actual rotation angles and hence result in successful insertion.

Table 6-1 Typical forces and torques used in acquiring Type 1 & Type 2 skills

<table>
<thead>
<tr>
<th>Type of Skills</th>
<th>Jamming states</th>
<th>Peg is moved up to 0.4mm above jamming state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>$F_x = -38$, $F_y = 100$, $F_z = -42$, $M_x = -342$, $M_y = -145$</td>
<td>$F_x = -34$, $F_y = 70$, $F_z = -25$, $M_x = -241$, $M_y = -117$</td>
</tr>
<tr>
<td>Case 2</td>
<td>$F_x = -30$, $F_y = 105$, $F_z = -57$, $M_x = -356$, $M_y = -124$</td>
<td>$F_x = -29$, $F_y = 87$, $F_z = -35$, $M_x = -299$, $M_y = -110$</td>
</tr>
</tbody>
</table>

Table 6-2 Data from a successful insertion with Type 1 skills only

<table>
<thead>
<tr>
<th>States before jamming</th>
<th>$F_x$</th>
<th>$F_y$</th>
<th>$F_z$</th>
<th>$M_x$</th>
<th>$M_y$</th>
<th>Pos</th>
<th>$x_{\text{step}}$</th>
<th>$y_{\text{step}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>-88</td>
<td>-16</td>
<td>290</td>
<td>269</td>
<td>29001</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>81</td>
<td>-92</td>
<td>-20</td>
<td>304</td>
<td>276</td>
<td>29048</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>90</td>
<td>-90</td>
<td>-26</td>
<td>305</td>
<td>306</td>
<td>29093</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>96</td>
<td>-94</td>
<td>-33</td>
<td>317</td>
<td>327</td>
<td>29135</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>93</td>
<td>-98</td>
<td>-99</td>
<td>350</td>
<td>317</td>
<td>29177</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Jamming state and actions</th>
<th>$F_x$</th>
<th>$F_y$</th>
<th>$F_z$</th>
<th>$M_x$</th>
<th>$M_y$</th>
<th>Pos</th>
<th>$x_{\text{step}}$</th>
<th>$y_{\text{step}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
<td>-99</td>
<td>-159</td>
<td>382</td>
<td>314</td>
<td>29029</td>
<td>10</td>
<td>-16</td>
<td></td>
</tr>
</tbody>
</table>

| Force/torque            | $F_x = 91$, $F_y = -99$, $F_z = -159$, $M_x = 382$, $M_y = 314$ |
| Normalized inputs for Type 1 skills | $1:0.803333$, $2:0.170000$, $3:-0.030000$, $4:0.882000$, $5:0.814000$, $6:0.167080$ |

<table>
<thead>
<tr>
<th>Actual angles</th>
<th>$x_{\text{step}} = 10$, $y_{\text{step}} = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_x$</td>
<td>$F_y$</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>-1</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
</tr>
</tbody>
</table>

In the remainder of this section the results of two experimental works conducted under two defined jamming conditions will be reported.
6.4.1 Jamming Condition 1

In this situation, the peg is considered jammed when $F_z$ is less than -200uf. Two examples of jamming under this condition are illustrated in Fig. 6-5 and Fig. 6-6. The variation of 8 series $F_x, F_y, F_z, M_x, M_y, Z$ (scaled), $x\_step$ (scaled) and $y\_step$ (scaled) are illustrated in these diagrams, where

$F_x, F_y, F_z$ are forces,

$M_x, M_y$ are torques about $X$ and $Y$ axes,

$Z$ is the displacement of the peg along Z-axis,

$X\_step$ and $y\_step$ are corrective actions generated by the developed skill models. They are steps for the stepper motors to turn the hole about $X$-axis and $Y$-axis, respectively.

On the horizontal axis in Fig. 6-5 and Fig. 6-6, ‘1’ indicates the first sampled state; ‘2’ indicates the second sampled state and so on. A sudden increase in the amplitudes of the forces and torques represent the jamming of the peg in the hole. Each time the peg is jammed or some force/torque thresholds are reached, the peg is stopped immediately and the hole is turned about $X$ and $Y$ axes according to the corrective angles estimated by the skill models. On the other hand, if the peg is not jammed, it is pushed down consistently.

There are two jamming incidents for the first case (Fig. 6-5) and one for the second case (Fig. 6-6). Each jamming condition is corrected by one set of actions. These actions and their corresponding states at the jamming points are listed in Table 6-3.

$F_x, F_y, F_z, M_x, M_y$ are recorded from the force sensor. $Pos$ is read from the decoder. $Pos\_n$ is the normalized value of $Pos$. $Step\_x$ and $Step\_y$ are scaled values of $x\_step$ and $y\_step$ respectively for illustration in Fig. 6-5 and Fig. 6-6. During insertion performed by the real rig, a loop in the program (as shown in Fig. 4-7) will
keep running to control the insertion process. Each time the loop will move the peg slightly down or take an action and check whether the peg is jammed or whether the insertion is completed. The number of iterations is the number of times the loop runs. The iteration time depends on many factors such as the computer speed, the peg moving speed, and the stepper motor turning speed. The insertion speed of the real rig can be improved by moving the peg faster but it’s better not to increase the moving speed of the peg to avoid accidental damages of the rig.
Table 6-3  Corrective actions

<table>
<thead>
<tr>
<th>Fig.</th>
<th>6-6</th>
<th>6-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>79</td>
<td>217</td>
</tr>
<tr>
<td>$Fx$</td>
<td>366</td>
<td>171</td>
</tr>
<tr>
<td>$Fy$</td>
<td>-421</td>
<td>-162</td>
</tr>
<tr>
<td>$Fz$</td>
<td>-220</td>
<td>-207</td>
</tr>
<tr>
<td>$Mx$</td>
<td>1453</td>
<td>337</td>
</tr>
<tr>
<td>$My$</td>
<td>1315</td>
<td>288</td>
</tr>
<tr>
<td>$Pos$</td>
<td>29174</td>
<td>33568</td>
</tr>
<tr>
<td>$Pos-n$</td>
<td>18.97273</td>
<td>38.94545</td>
</tr>
<tr>
<td>$x_{step}$</td>
<td>-1</td>
<td>-4</td>
</tr>
<tr>
<td>$y_{step}$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$Step_x$</td>
<td>-200</td>
<td>-800</td>
</tr>
<tr>
<td>$Step_y$</td>
<td>400</td>
<td>400</td>
</tr>
</tbody>
</table>

At Point 79 in Fig. 6-5, the peg is jammed. The actions are taken to rotate the hole about the $X$-axis one step clockwise and about the $Y$-axis two steps counterclockwise. The peg is then inserted further down until the $Pos$ value has reached a magnitude of 33568 at Point 217 where the peg is jammed again. Actions are taken to rotate the hole about the $X$-axis four steps clockwise and about the $Y$-axis two steps counterclockwise. The insertion is successfully completed afterwards.

It can be seen from Table 6-3 that in the first experiment (Fig. 6-5) the forces and torques at Point 217 are smaller than those at Point 79, although the peg is jammed at both occasions.

In the second experiment (Fig. 6-6), the peg is jammed at Point 73. Actions are taken to rotate the hole by two steps clockwise about the $X$-axis and 35 steps counterclockwise about the $Y$-axis. Then the peg is inserted down successfully.

These two experimental results illustrate the efficiency of the system and its effectiveness in dealing with jamming problems.
6.4.2 Jamming Condition 2

The threshold 200uf in jamming condition 1 is equivalent to about 11.3N (2*5.6448N) which exceeds the force limit in the Phantom. Thus the jamming state of the peg in the physical rig has no equivalence in the virtual environment. Jamming condition 2 is defined tighter to ensure the forces in the real rig do not exceed the force constraints of the haptic device. In jamming condition 2, the peg is considered jammed when $F_z$ is less than -100uf (-5.6448N) and the magnitude of $F_x$ or $F_y$ is more than 100uf (5.6448N). This condition is implemented for two examples of on-line training. In the first example, the insertion algorithm is trained on-line based on 100 successful insertions. In the second example, the training is carried out for more than 400 successful insertions.

The historical data for insertion examples one and two are respectively illustrated in Fig. 6-7 and Fig. 6-8. The number of corrective actions in the second example is only 2, significantly smaller than the first example. The sequences of events and actions around the two jamming states for the second example are illustrated in Table 6-4 and Table 6-5.
Fig. 6-7  Example 1 under Condition 2, (a) Forces/torque; (b) Actions; (c) Position.
Fig. 6-8  Example 2 under Condition 2. (a) Forces/torque; (b) Actions; (c) Position
Table 6-4  First series of corrective actions (Example 2 of Situation 2)

<table>
<thead>
<tr>
<th>Fx</th>
<th>Fy</th>
<th>Fz</th>
<th>Mx</th>
<th>My</th>
<th>Pos</th>
<th>x_step</th>
<th>y_step</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>-48</td>
<td>-23</td>
<td>156</td>
<td>234</td>
<td>28541</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>72</td>
<td>-53</td>
<td>-21</td>
<td>173</td>
<td>238</td>
<td>28579</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>72</td>
<td>-56</td>
<td>-22</td>
<td>183</td>
<td>249</td>
<td>28617</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>-55</td>
<td>-29</td>
<td>181</td>
<td>273</td>
<td>28655</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>83</td>
<td>-55</td>
<td>-28</td>
<td>180</td>
<td>284</td>
<td>28691</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>-55</td>
<td>-31</td>
<td>182</td>
<td>311</td>
<td>28729</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>95</td>
<td>-58</td>
<td>-28</td>
<td>196</td>
<td>323</td>
<td>28768</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>-57</td>
<td>-30</td>
<td>191</td>
<td>345</td>
<td>28717</td>
<td>10</td>
<td>-10</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>-6</td>
<td>-64</td>
<td>32</td>
<td>28662</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>-7</td>
<td>-70</td>
<td>42</td>
<td>28702</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>23</td>
<td>-6</td>
<td>-78</td>
<td>52</td>
<td>28741</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>18</td>
<td>-5</td>
<td>-64</td>
<td>46</td>
<td>28781</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>21</td>
<td>-3</td>
<td>-72</td>
<td>47</td>
<td>28823</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>21</td>
<td>-4</td>
<td>-69</td>
<td>51</td>
<td>28861</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6-5  Second series of corrective actions (Example 2 of Situation 2)

<table>
<thead>
<tr>
<th>Fx</th>
<th>Fy</th>
<th>Fz</th>
<th>Mx</th>
<th>My</th>
<th>Pos</th>
<th>x_step</th>
<th>y_step</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>46</td>
<td>-31</td>
<td>-151</td>
<td>196</td>
<td>45525</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>98</td>
<td>46</td>
<td>-29</td>
<td>-152</td>
<td>195</td>
<td>45549</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>98</td>
<td>47</td>
<td>-27</td>
<td>-151</td>
<td>197</td>
<td>45572</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>47</td>
<td>-27</td>
<td>-151</td>
<td>199</td>
<td>45414</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>90</td>
<td>43</td>
<td>-34</td>
<td>-127</td>
<td>176</td>
<td>45442</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>90</td>
<td>43</td>
<td>-38</td>
<td>-130</td>
<td>179</td>
<td>45471</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>91</td>
<td>41</td>
<td>-34</td>
<td>-126</td>
<td>182</td>
<td>45499</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>42</td>
<td>-35</td>
<td>-124</td>
<td>185</td>
<td>45525</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>86</td>
<td>38</td>
<td>-37</td>
<td>-82</td>
<td>175</td>
<td>45548</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

6.4.3 Overall Performance

The approach developed in this study was applied to 50 insertions with initial angles between -4.5° to 4.5°. Three cases required human interventions and the rest were successfully completed by applying the two types of skills. Table 6-6 shows the related statistic analysis.
Table 6-6  Statistical analysis

<table>
<thead>
<tr>
<th>50 trials</th>
<th>Average* (per trial)</th>
<th>Effected trials</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion time (s)</td>
<td>60.2</td>
<td></td>
<td>Insertion time can be reduced if peg moving-down speed along Z-axis is increase.</td>
</tr>
<tr>
<td>Number of jamming states</td>
<td>6.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of corrective actions</td>
<td>(4.8, 5.3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actions applied by Type 1 skills</td>
<td>(4.6, 5.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actions applied by Type 2 skills</td>
<td>(0.18, 0.16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of human interventions</td>
<td>0.1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Initial angles</td>
<td></td>
<td></td>
<td>1.8° &lt;</td>
</tr>
</tbody>
</table>

*For data of type (x,y), x means actions along x-axis while x means actions along x-axis.

6.4.4 Comparison with Other Methods

In order to illustrate the significance of the algorithm developed in this work, the following analysis and comparisons have been carried out:

1. The impact of different planning settings.

2. Applying Lwprs algorithm to acquire Type 1 skills

3. Acquiring data on jamming states from the physical rig rather than the virtual environment

4. Comparison with other insertion algorithms.

The experimental results provided in Sections 6.4.1 and 6.4.2 illustrate Case 1 listed above. As shown, different planning settings can lead to different behaviours. For example, the final state under the jamming state 2 is better than that under the jamming state 1. The insertion, however, may take longer to complete. In the following sections cases 2-4 are discussed.
6.4.4.1 Case 2

Figures 6-9 to 6-12 show four successful insertion examples with different initial angles under jamming condition 2. The transients in the force/torque data are the result of initialization of the force sensor due to “out of sequence” error as mentioned in Section 6.3. These cases require more actions when compared with the results shown in Fig. 6-7 and Fig. 6-8. Hence, the Lwpr algorithm does not perform as well as the SVM algorithm in acquiring Type 1 skills which are classification skills.

Fig. 6-9 Example 1 in Case 2 (initial angles: X-axis Misalignment = 4.5º and Y-axis Misalignment = -4.5º), (a) Forces/torque; (b) Actions.
Figure 6-10: Example 2 in Case 2 (initial angles: X-axis Misalignment = -4.5° and Y-axis Misalignment = -4.5°), (a) Forces/torque; (b) Actions.
Fig. 6-11  Example 3 in Case 2 (initial angles: X-axis Misalignment = -4.5° and Y-axis Misalignment = 4.5°), (a) Forces/torque; (b) Actions.
6.4.4.2 Case 3

In this case, the data on the jamming states are collected from the rig. Forces/torque, positions and angles of the jamming states are respectively shown in figures 6-13 to 6-15. The angles of the states are between the two irregular circles shown in Fig. 6-16. The states inside the small irregular circle are the final states. The peg is inserted successfully when it is at these states. The jamming condition is defined as Section 6.4.2, i.e., when the magnitude of any force is greater than 100 uf, the peg is stopped.

In this case, the manipulation skills are learned from this data acquired from the physical rig. The results obtained in Case 3 are shown in figures 6-17 to 6-19 for
Example 1 and in figures 6-20 to 6-22 for Example 2. The results clearly indicate that the system performs better when the training data is obtained from the physical environment. This was expected as the virtual environment does not fully reflect the dynamics of the physical rig.

![Force/Torque Graph](image1)

**Fig. 6-13** Forces/torque of jamming states

![Position Graph](image2)

**Fig. 6-14** Positions of the jamming states
Fig. 6-15  Angles of the jamming states

Fig. 6-16  Jamming states
Fig. 6-17 Forces/torque (Example 1 of Case 3)

Fig. 6-18 Position (Example 1 of Case 3)
Fig. 6-19  Actions (Example 1 of Case 3)

Fig. 6-20  Forces/torque (Example 2 of Case 3)
6.4.4.3 Case 4

In this section the approach developed in this work will be compared with two other methods:
1. A Heuristic method using the Adaptive Spline Modelling of Observed Data (ASMOD) [Lukasiak(thesis)]

A multi-dimensional B-spline model is the tensor product of one dimensional models while a one dimensional B-spline model is a linear combination of B-spline basis functions which are simply piecewise polynomial mappings similar to fuzzy membership functions. However, if \( k \) univariate basis functions are defined for each axis and the input dimension equal to \( n \), then \( k^*n \) parameters are required which makes B-splines unsuitable for modelling high dimensional problems [Kavli95]. The ASMOD algorithm can model the data in high dimensional data modelling problems as a sum of low dimensional sub models. For a model with 8 basis functions and 10 input dimensions, the number of parameters could be reduced from 810 to 320 \((8^2 + 8^2 + 8^2 + 8^2 + 8^2)\) if data is modelled as a sum of two dimensional sub models. ASMOD also determines strong correlation between input variables which can remove variables that contribute little to the model’s performance or accuracy.

2. Fuzzy Controller [Nguyen(thesis)]

In this method, two fuzzy controllers are used for insertion motion (ie, moving down the peg) control and corrective motion (ie, tilting the hole) control respectively. The rules of insertion motion control have a typical form of:

\[
\text{IF } F_z \text{ is } S \text{ (small), } M_x \text{ is } S \text{ (small), } M_y \text{ is } S \text{ (small) and } Z \text{ is } B \text{ (big) THEN } d_z \text{ (displacement of the peg) is } B \text{ (big)}. 
\]

The rules of corrective motion control have a typical form of:

\[
\text{IF } M \text{ (torque) is } Z \text{ (zero) and } CM \text{ (change of } M) \text{ is any THEN } \Theta \text{ (corrective angle for tilting the hole) is } Z \text{ (zero)}. 
\]
The performances of these two algorithms are provided in tables 6-7 and 6-8 as produced in previous studies on this rig. Table 6-7 shows the results for a Heuristic method using the Adaptive Spline Modeling of Observed Data (ASMOD) [Lukasiak(thesis)]. Table 6-7 shows the results of Fuzzy Controller [Nguyen(thesis)]. The jamming conditions of these two methods are the same, i.e., the peg is considered jammed when $F_z$ is less than -200uf.

Table 6-9 shows the results for the method used in this thesis. The jamming conditions are more strict, i.e., the peg is considered jammed when $|F_x| > 100uf$, $|F_y| > 100uf$, $F_z > -100uf$. A comparison of the results provided in tables 6-7 to 6-9, reveal that the ASMOD method needs too many actions and the fuzzy controller fails too many times, while the method used in this thesis allows the peg to be inserted more gently and effectively without excessive forces/torque.

Table 6-7  Results for ASMOD method (jamming condition: $F_z < -200uf$)

<table>
<thead>
<tr>
<th>Trial Number</th>
<th>X axis Misalignment in degrees</th>
<th>Y axis Misalignment in degrees</th>
<th>Insertion time in seconds</th>
<th>Program Iterations</th>
<th>X model actions</th>
<th>Y model actions</th>
<th>X model program corrections</th>
<th>Y model program corrections</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-4.5</td>
<td>-4.5</td>
<td>180</td>
<td>720</td>
<td>58</td>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-4.5</td>
<td>4.5</td>
<td>200</td>
<td>708</td>
<td>57</td>
<td>21</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>4.5</td>
<td>4.5</td>
<td>240</td>
<td>680</td>
<td>45</td>
<td>18</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>-4.5</td>
<td>220</td>
<td>764</td>
<td>79</td>
<td>8</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4.5</td>
<td>4.5</td>
<td>190</td>
<td>651</td>
<td>36</td>
<td>18</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>-4.5</td>
<td>-4.5</td>
<td>166</td>
<td>639</td>
<td>33</td>
<td>1</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>-4.5</td>
<td>4.5</td>
<td>158</td>
<td>640</td>
<td>33</td>
<td>7</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>4.5</td>
<td>4.5</td>
<td>172</td>
<td>578</td>
<td>42</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Average values</td>
<td>190.8</td>
<td>639.0</td>
<td>47.9</td>
<td>12.5</td>
<td>10.1</td>
<td>4.6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 6-8  Results of the fuzzy controller (jamming condition: $F_z < -200uf$)

<table>
<thead>
<tr>
<th>Trial Number</th>
<th>X axis Misalignment in degrees</th>
<th>Y axis Misalignment in degrees</th>
<th>Insertion time in seconds</th>
<th>Program Iterations</th>
<th>X model actions</th>
<th>Y model actions</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.5</td>
<td>4.5</td>
<td>N/A</td>
<td>32</td>
<td>3</td>
<td>3</td>
<td>Failed. Excessive force</td>
</tr>
<tr>
<td>2</td>
<td>4.5</td>
<td>4.5</td>
<td>N/A</td>
<td>80</td>
<td>7</td>
<td>7</td>
<td>Failed. Excessive force</td>
</tr>
<tr>
<td>3</td>
<td>4.5</td>
<td>4.5</td>
<td>350</td>
<td>1014</td>
<td>7</td>
<td>7</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>-4.5</td>
<td>-4.5</td>
<td>N/A</td>
<td>262</td>
<td>14</td>
<td>14</td>
<td>Failed. Excessive force</td>
</tr>
<tr>
<td>5</td>
<td>-4.5</td>
<td>4.5</td>
<td>N/A</td>
<td>301</td>
<td>53</td>
<td>53</td>
<td>Failed. Excessive force</td>
</tr>
<tr>
<td>6</td>
<td>4.5</td>
<td>-4.5</td>
<td>N/A</td>
<td>248</td>
<td>44</td>
<td>44</td>
<td>Failed. Excessive force</td>
</tr>
<tr>
<td>7</td>
<td>4.5</td>
<td>4.5</td>
<td>250</td>
<td>781</td>
<td>9</td>
<td>9</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>-4.5</td>
<td>-4.5</td>
<td>N/A</td>
<td>527</td>
<td>99</td>
<td>99</td>
<td>Failed. Excessive force</td>
</tr>
<tr>
<td>9</td>
<td>-4.5</td>
<td>4.1</td>
<td>N/A</td>
<td>306</td>
<td>54</td>
<td>54</td>
<td>Failed. Excessive force</td>
</tr>
<tr>
<td>10</td>
<td>2.8</td>
<td>2.8</td>
<td>130</td>
<td>494</td>
<td>6</td>
<td>6</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>4</td>
<td>170</td>
<td>493</td>
<td>12</td>
<td>12</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>2.75</td>
<td>1</td>
<td>150</td>
<td>525</td>
<td>2</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>-2.2</td>
<td>2</td>
<td>86</td>
<td>301</td>
<td>5</td>
<td>5</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 6-9  Results for the method used in this thesis (jamming condition: $|F_x| >100uf, |F_y| >100uf, F_z < -100uf$)

<table>
<thead>
<tr>
<th>Trial Number</th>
<th>X axis Misalignment in degrees</th>
<th>Y axis Misalignment in degrees</th>
<th>actions (X model)</th>
<th>actions (Y model)</th>
<th>Max force encountered (uf)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.5</td>
<td>4.5</td>
<td>2</td>
<td>2</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>-4.5</td>
<td>-4.5</td>
<td>7</td>
<td>7</td>
<td>108</td>
</tr>
<tr>
<td>3</td>
<td>-4.5</td>
<td>4.5</td>
<td>5</td>
<td>8</td>
<td>105</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>-4.5</td>
<td>6</td>
<td>9</td>
<td>111</td>
</tr>
<tr>
<td>5</td>
<td>1.8</td>
<td>1.8</td>
<td>1</td>
<td>2</td>
<td>100</td>
</tr>
</tbody>
</table>

6.5 Chapter Summary

The structure of the experimental rig and the results of the experiments conducted to validate the approach have been presented in this chapter. The success of the method,
similar to other heuristic approaches, depends very much on the quality of the training data obtained from the virtual environment. The performance of the algorithm is improved when on-line data obtained from the physical rig is used for further enhancement of the insertion algorithm. The results reflect efficient and effective performance of the approach while highlight that a haptic rendered virtual environment cannot provide an ultimate system for training unless it accurately manifests the characteristics of the physical rig.
CHAPTER 7 CONCLUSIONS AND FURTHER RESEARCH

7.1 Overview
The primary aim of this project was to explore the feasibility of developing a skill-based robotics assembly system for constrained motion manipulation. The training data for building the skill primitives was derived from a haptic rendered virtual model of the manipulation. While the study was conducted in the context of the peg-in-hole insertion, the developed approach was maintained to be as generic as possible.

The approach was designed and developed based on the model identified for human psychomotor learning, imitating some possible stages in this process. The overall system has been extensively validated. The developed strategy has performed effectively. The results are encouraging and indicate that the hypothesis pursued in the project is feasible, though challenging.

7.2 Evolution in Robot Programming
A comprehensive study of evolution of robot programming methodologies was conducted in Chapter 2 of the thesis. It was clearly shown that programming of a robotic manipulator has been an important area of research and development since the inception of modern computer-based robotic systems. In this evolution, new programming methodologies have been developed as the complexity of the robotics manipulators and their applications have increased. Different stages of this development were identified as text programming, offline simulation-based programming, inductive learning, teaching by guiding and teaching by showing. Technologically, significant research and development works in each category were reviewed and their major strengths and shortcomings were analysed.
The outcomes of the review highlighted that the work pursued in this thesis was a new paradigm for programming of robotics manipulators engaged in complex constrained motion assembly and manipulation tasks. The approach though has similarities to the work reported in the literature on learning from examples, goes beyond knowledge acquisition and building a knowledge-base. The focus is on acquiring human manipulation skills performed in psychomotor domain.

7.3 Haptic Rendered Virtual Manipulation

The training data for acquiring basic manipulation skills has been obtained from a haptic-rendered virtual model of the assembly task. Experience gained in the work indicates that such approach has tangible advantages while suffers from constraints imposed in computer modelling of a system.

A haptic rendered model of the manipulation process offers a user-friendly, secure, safe and ‘soft’ environment which can be easily modified if the system requirements change. It also provides a direct and simple method to extract and record the training data used in building the manipulation skills.

On the other hand, constructing an accurate haptic rendered model of the assembly process is quite challenging and sometimes impossible due to existing software and hardware constraints. One limiting factor is the maximum force feedback that a haptic device can generate in response to the constraints encountered in the model. This can be significantly less than the actual force generated in a physical assembly rig.

Collision detection is a critical process in understanding the interaction taking place in the virtual environment and calculating the generated forces and torques. If the geometric and haptic models are too complex, the force computational rate will
decrease which in turn results in increased latency. As a result, hard surfaces in the virtual environment may feel soft and the system may become unstable.

In order to overcome latency, the geometric and haptic model of the manipulation environment are kept simple. This has the diverse effect of not reflecting the full characteristics of the physical manipulation system.

In this work, the shortcomings of the approach are compensated by further generalization of the skill learning methods and enhancing of the skills database by employing complimentary on-line methods such as on-line incremental learning.

The virtual model of the peg-in-hole assembly process was constructed through the development of graphic and geometric models. Haptic update rate must be around 1 KHz to maintain stable force interactions while graphic update rate must be between 20-30Hz to meet the real-time requirements. These constraints primarily determined the approaches adopted in building the two models.

The graphic model was constructed by triangle polygon mesh provided in the Ghost modelling package. The approach significantly reduces the computation time during haptic rendering as only the polygons representing the top and inner surfaces of the hole are updated.

Three different approaches have been applied for haptic rendering modelling, including two PointShell methods and a Cylindrical method. The Cylindrical method has proved more accurate as it does not use polygons to approximate the circle.

7.4 Skill Acquisition and Transfer

The approach adopted for transfer of manipulation skills from the operator’s manipulation in the virtual environment to the physical robot was inspired by the human skill acquisition taxonomy. Imitation is the product of interaction between
perception, memory, and motor control. In addition to self-discovery and observation, learning of skills in humans generally takes place through training by an instructor in the psychomotor domain. In this study, the focus is on the manipulative movements as they represent the type of movements emulated by robotics manipulators in automatic assembly.

The approach developed in this work identifies a set of basic skills from the training data obtained from the virtual environment. Basic skills are defined generically based on the contact states and state transitions taking place in the virtual manipulation. While independent from the physical manipulator, the basic skills are closely associated with the application and the method used for the manipulation.

The training data used for skill acquisition is primarily generated from the haptic rendered virtual environment. Some on-line learning also takes place from the physical rig to make up for the inaccuracies of the virtual model. Two different types of skills have been identified. The first type of skills is learned during task sequence planning or trajectory optimization which finds the best state change sequence. In order to apply this skill, the next desired state or the method of choosing the next state should be known. State changes with the same current state but different next states might result in different output actions.

The second type of skills is acquired during performing a task with no obvious or fixed state change sequence. It is only based on the current state to simplify the skill learning process. It does not require an optimum state change sequence or to follow a pre-defined state change sequence. The second type of skills is derived from the virtual environment and further enhanced online during physical manipulation.

This approach expands the input space of the training scheme and increases the generality of the approach.
7.5 Overall Performance of the System

The validation of the approach during different stages of its development and the final experimental work indicate that the hypothesis pursued in this work is feasible and there is potential for its further development.

The force, torques and position data produced from the haptic rendered virtual manipulation explicitly reflects the characteristics of the manipulation taking place in the virtual environment. This was illustrated by the sample data sets provided in Chapter 3. In particular, the force and torque data sets directly represent the tactile experience of the operator when manipulating the virtual system.

The proposed manipulation skill transfer scheme is a multi-layer approach which can be easily applied to other manipulation systems. While, the learning mechanism is modelled according to the dynamics of the peg-in-hole insertion process at the low level, it provides a generic methodology at macroscopic level for use in other applications. The scheme augments the off-line skill acquisition process with further on-line skill learning in order to compensate for the limitations introduced by the haptic rendered virtual model.

The operation of skill acquisition algorithm for both Type 1 and Type 2 skills are illustrated through two experimental studies in provided Chapter 5. The results obtained strongly indicate a high correlation between the estimated and actual correction in terms of the magnitudes of the corrective angles about X and Y axes of the hole.

The overall performance of the approach was further validated in Chapter 6 through a series of experiments. The results indicate that similar to other heuristic approaches, the performance quality of the system depends very much on the quality of the training data obtained from the virtual environment. It is also shown that better
results are obtained when the skill acquisition algorithm is further modified by the on-line data obtained from the physical rig.

Overall, the results obtained from the experimental work show that the proposed approach works efficiently and effectively. They also highlight that a haptic rendered virtual environment cannot provide an ultimate system for training unless it accurately manifests the characteristics of the physical rig.

7.6 Further Research

The work conducted in this thesis can be further extended in a number of directions as described in the following sections.

7.6.1 Higher Degree of Freedom Haptic Device

The haptic rendered model and the training data generated from it are designed and manipulated by a 3 d.o.f haptic device. In this model, the force interaction between the peg and the hole takes place along $X$, $Y$ and $Z$ axes. The haptic device, however, cannot provide torque feedback or manipulation about these axes.

In this application, the graphics model of the assembly is constructed using OpenGL, whereas its physical model and the force/torque vectors generated in the virtual manipulation environment are modelled based on two different approaches of PointShell and TriPolyMesh. The developed system has proved quite stable when the peg-in-hole insertion is performed using 3 DOF Phantom.

In applications where simulation of arbitrary object to object interaction is required, a six degree-of-freedom (6 DOF) haptic device can be a more effective tool. It provides torque feedback in addition to force display within a large translation and rotational range of motion, and provides the user with the much needed dexterity to feel, explore and manoeuvre around other objects in the virtual environment.
A study should be conducted to explore whether a six d.o.f haptic device can successfully drive the models developed for the three d.o.f system.

7.6.2 Skill Acquisition Algorithm

A more systematic approach should be developed for the classification of the contact states from the haptic rendered virtual manipulation model and estimation of the assembly states during physical manipulation. This will lead to a more generic skill acquisition algorithm at both micro and macro levels. It will also simplify and speed up the implementation of the approach for a new application. Hidden Markov Model or some variation of it can be used as the first step in this study.

7.6.3 Transportability of the Approach

This study has not examined how transportable is the approach across different applications. The future work can identify a different constrained motion manipulation such as a gear assembly with tight fit and explore how effectively the algorithms and methodologies developed in this work can be deployed and applied in the new application. This will highlight the strengths and weaknesses of the approach across different applications and will also reveal the major changes which need to be applied to the proposed methodology to make it more generic and effective.
REFERENCES


[Ahmad(online)] Image processing & machine vision Dr. Shaun Lawson, URL: http://www.surrey.ac.uk/eng/InfoPoint/online/emc/robotics%202.ppt


[AMROSE(online)] AMROSE Robotics, URL: http://www.amrose.dk/side.asp?ID=8


[Fuzzy(online)] Fuzzy Drilling Direction Controller. ULR: http://www.makinhole.com/thdt-1.htm


[Inpout32(online)] Power tools for interfacing, URL: http://www.logix4u.cjb.net/

[Interaction(online)] Interaction Lab. Extracting Motion Primitives from Movement Data, URL: http://robotics.usc.edu/~agents/projects/ajo-project.html


[labview(online)] Labview, URL: http://www.ni.com/labview/

[Lawson(online)] Roadmap of Learning (PDF), URL: http://www.eng.usm.my/~badli/rizal/rb_lt12.pdf


[Lwpr(online)] Locally Weighted Projection Regression (LWPR), URL: http://www-clmc.usc.edu/sethu/software/LWPR/

[Lwr(online)] Locally Weighted Regression, URL: http://www-2.cs.cmu.edu/afs/cs/project/jair/pub/volume4/cohn96a-html/node7.html


[Mechatronics(online)] Mechatronics, URL: http://claymore.engineer.gvsu.edu/~jackh/eod/mechtron/mechtron-1.html


<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Millersville(online)]</td>
<td>Millersville University Penn State College of Medicine Research Project in Surgical Simulation, URL: <a href="http://cs.millersv.edu/haptics/index.html">http://cs.millersv.edu/haptics/index.html</a>.</td>
</tr>
<tr>
<td>[MIT(online)]</td>
<td>MIT Touch Lab, URL: <a href="http://touchlab.mit.edu/oldresearch/areas/softwaretools/">http://touchlab.mit.edu/oldresearch/areas/softwaretools/</a></td>
</tr>
<tr>
<td>[MOBARC(online)]</td>
<td>MORO LORE/RTMPC - Realtime Motionplanning And Control, URL: <a href="http://www.mip.sdu.dk/robbox/Mobarc/">http://www.mip.sdu.dk/robbox/Mobarc/</a></td>
</tr>
</tbody>
</table>


[NTPort(online)] NTPort Library, URL: http://www.zealsoftstudio.com/ntport/


<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>


[SensAble(online)] SensAble Technologies - Haptics Research, URL: http://www.sensable.com/haptics/

[Sense8(online)] Sense8, URL: http://www.sense8.com/index.html


[ToonTalk (online)] ToonTalk - Making programming child's play, URL: http://www.toontalk.com/


[Visual(online)] Lecture notes, URL: http://www.csse.monash.edu.au/~berndm/SFT5030/Lectures/Lecture%2010.ppt


Minneapolis, 1996, 2616-2621.


APPENDIX A  EXPERIMENTAL RIG AND RIG CONTROL PROGRAMS

A. 1  Introduction
This appendix introduces and describes the hardware, software and programs associated with the experimental rig. The connection and relationship between different parts of the rig are also explained.

A. 2  Hardware Components
A diagram illustrating the hardware structure of the rig is illustrated in Fig. 6-2. Step Motors, Microstep Drives, Servo Motor with Encoder, DC Servo Controller, DAQ-802 Board, Decoder and F/T Sensor used in this project are introduced below.

A.2.1  Sanyo Denki 2 Phase Stepper Motor
In the previous rig or its first generation, each axis of the hole was driven by stepper motor driving the shaft through a gearbox to provide high torque. The gearbox, however, caused backlash. Hence in the second generation the motors were replaced with high torque Sanyo Denki 2 phase stepper motors (103H7822-0440) (Fig. A-1)

Fig. A-1  Sanyo Denki 2 phase stepper motor [Farnell(online)]

This motor has the following features [Farnell(online)]:

- [Feature 1 description]
- [Feature 2 description]
- [Feature 3 description]
- Hybrid, High Torque 1.8° Step Angle
- Size 24 Body: H=82, W=60 D=53.8 (85.8, 635-236)
- 2 phase hybrid rare earth magnet technology offering 15% to 20% more torque than standard hybrid types
- Improved high speed operation, low noise and low vibration
- Very high positional accuracy, designed for micro-stepping
- Can be unipolar or bipolar driven (bipolar only, 720-495, 635-248, 635-250)
- Size 24 motors are available with integrated connector for ease of assembly, crimps and matching socket supplied
- Holding Torque (mNm): 1170; Voltage(v): 4; Current(A/phase): 2; Resistance(Ω/phase): 2; Inductance(mH/phase): 3.6; Weight(g): 770;

Fig. A-2 illustrates the linking of the stepper motors to the drive circuit.

![Fig. A-2 Connections for stepper motors](image)

A.2.2 Gecko G201 7A, 80V 10 Microstep Drive

The stepper motor drives used in the first generation could provide a minimum of 0.9 degree resolution for turning the hole. The Gecko microstep drives used to derive the motors in the new generation can improve the resolution by up to 10 times and to enhance the holding torque through a special circuit. The Gecko microstep drive
doesn’t need external circuits to generate and enlarge the clock and direction signals as the old step drive does. The Gecko microstep drive is illustrated in Fig. A-3.

![Gecko microstep drive](Gecko(online))

The drive has the following characteristics [Gecko(online)]:

- 0.3A to 7A phase current
- 24V to 80V power supply
- 10 microsteps per step
- Size 17 to 42 motors
- Mid-band resonance compensation
- Low-speed resonance nulling trimpot
- Auto current reduction
- 200kHz max step rate
- Opto-isolated Step and Direction
- 20W dissipation at 7A per phase
- Silent: 20kHz master oscillator PWM
- 0°C to 70°C operating temperature
- Rugged: All n-channel TO-220 MOSFETs
- LED power indicator
- Small size: 2.5" by 2.5" by 0.82" (63mm by 63mm by 21mm)
- Light: 3.6 oz (100gm)
- Anodized aluminium package
Fig. A-4 shows connections for the stepper motor drive.

A.2.3 DC Servo Motor W/Encoder

The DC servo motor used is from the Baldor family. It can provide self-contained integral encoder for rugged industrial applications as shown in Fig. A-5.

The features of the motor include [Baldor(online)]:

- Range of output torques to 56 lb-in (A.5 N-m)
- High continuous duty 155°C rotor temperature
- Premium moisture resistant, multi-coated copper wire
- Small compact design
- TENV construction for protection
- Stock and custom designs available
- Optional integral tach and encoder
- Superior performance down to zero speed
- Matched performance with Baldor controls

Fig. A-5  MTE-2250-AMACN DC servo motor W/Encoder [Baldor(online)]

Table A-1 shows how to connect the DC Servo Motor with other parts:

<table>
<thead>
<tr>
<th>PIN</th>
<th>FUNCTION (OPTIONAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ENCODER SHIELD (OR BRAKE)</td>
</tr>
<tr>
<td>B</td>
<td>TACH C.C.W. (+)</td>
</tr>
<tr>
<td>C</td>
<td>MOTOR C.C.W. (-)</td>
</tr>
<tr>
<td>D</td>
<td>CHANNEL Z</td>
</tr>
<tr>
<td>E</td>
<td>CHANNEL B</td>
</tr>
<tr>
<td>F</td>
<td>COMMON</td>
</tr>
<tr>
<td>G</td>
<td>CHANNEL A</td>
</tr>
<tr>
<td>H</td>
<td>MOTOR C.C.W. (+)</td>
</tr>
<tr>
<td>I</td>
<td>TACH SHIELD (OR BRAKE)</td>
</tr>
<tr>
<td>J</td>
<td>CHANNEL B</td>
</tr>
<tr>
<td>K</td>
<td>TACH C.C.W. (-)</td>
</tr>
<tr>
<td>L</td>
<td>CHANNEL Z</td>
</tr>
<tr>
<td>M</td>
<td>+5VDC</td>
</tr>
<tr>
<td>N</td>
<td>CHANNEL A</td>
</tr>
</tbody>
</table>
A.2.4 DC Servo Controller

The Baldor “TSD” Series “Twin Servo Driver” (as shown in Fig. A-6) is a one or two axis pulse width modulated (PWM), high performance servo control for DC brush type permanent magnet servo motors. This transistorized servo control provides not only converted DC power to the motor, but also precisely controlling current and velocity at the motor.

![TSD DC Servo Controller](image1)

![TSD Front Panel](image2)

(a) TSD  
(b) TSD Front Panel

Fig. A-6  DC Servo Controller (TSD-050-05-1-I) [Baldor (online)]

The DC Servo Controller has the following features [Baldor (online)]:

- 500 Watts per axis
- Velocity/current configuration
- Standard ±10VDC command input signal
- Front Panel on/off switch
- Front panel adjustments
- Status indicators
- Fully protected unit

Fig. A-7 shows connections for this component.
A.2.5 DAQ-802 Board and DaqEZ Professional Software

The DAQ-802 board is a cost effective high speed data acquisition board for IBM® compatible ISA bus applications (Fig. A-8). The pins of the DAQ-802 connector are shown in Fig. A-9 [DAQ(online)]. Correct analogue input and output field wiring is necessary for applications. The digital and analogue inputs can be read through a program or directly viewed and saved using the DaqEZ Professional software with the interface shown in Fig. A-10. In this example, the digital meter is chosen from the display manual to read the voltage of CH00-00 A/D channel, i.e., the voltage
from Pin 37 of the main I/O connector (Fig. A-9) with Pin 19 and 18 connected to the ground of the external signal source.
A.2.6 Decoder

The HCTL-2016 decoder [HCTL(online)] is connected to the encoder and the DAQ-802 board for obtaining position information (Fig. A-11). CHA and CHB read signals from the encoder mounted on the servo motor. The 12/1A-bit position latch is read through an 8-bit output port (D0-D7) in 2 sequential bytes. At first, both OE and SEL pins are set low via DAQ-802, and read in high byte of position. Then, SEL pins are set high via DAQ-802 and read in low byte of position. Finally, high and low bytes are combined to produce a 16 bit word. The features of the decoder are:

- 1A-bit counter; 14 MHz clock operation;
- 20 pin PLCC surface mount package.
A.2.7 F/T Sensor

The force/torque sensor used is Lord Force/Torque (F/T) sensing system which consists of a transducer unit and a system controller unit.

In this project, the F/T system is connected to the computer through RS-232 serial port which is configured for 9600 baud rate and 8N1 (8 bits, noparity, 1 stop bit) mode. The commands issued through the serial port are:

FT – Select resolved force/torque data

In response to this command, forces and torques acting on the transducer unit are resolved into three Cartesian force components fx, fy, fz and three Cartesian torque
components tx, ty, tz. The force/torque components are represented as a six-element vector, \( F = (f_x, f_y, f_z, t_x, t_y, t_z) \).

**OA** – Output ASCII on the serial port.

In response to this command, F/T will output continuous ASCII data. If FT type was selected, F/T records are transmitted as strings of 37 characters. The first character is ‘0’ or ‘1’, character ‘1’ representing the strain gauge saturation flag. Following are the six Cartesian force/torque components, each is expressed as a decimal number, right-hand justified in a six character field.

**OR** – Output one data record in ASCII mode.

In response to this command, output record is in the same format as with OA command, but only one data record is issued (one force/torque components vector).

**BS** – Normally, force/torque data output is biased by gravitational loading due to the weight of the end effector, work piece and any attached cables or hoses. For task where these effects are constant (the orientation of the end effector remains fixed with respect to the gravity vector), BS command can be used to remove this bias. This command establishes the current Transducer Unit load as bias to be subtracted from all subsequent force/torque output.

### A. 3 Rig Control Programs

C programs were first made and Djgpp were used as the C compiler (see Section A.3.1). To use VC++ as the compiler, the rig control programs are modified as shown in Section A.3.2. As mentioned in Section 6.2, Inp and Outp can work in Window 2000 & XP, if I/O port device drivers [NTPort(online)] are used.
A.3.1 Program for Get the Force/Torque Information

A.3.1.1 Using Djgpp as A Compiler

Djgpp is a C compiler which supports writing directly to the hardware [Djgpp(online)]. The programs for controlling the motors and getting the force/torque information are listed and explained below.

```c
#include<stdio.h>
#include<bios.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<dos.h>

//Setting the COM port
#define COM1          1
#define COM2          0
#define COM3          2
#define DATA_READY    0x100
#define TRUE          1
#define FALSE         0
#define SETTINGS      (_COM_9600 | _COM_CHR8 | _COM_STOP1 |
                    _COM_NOPARITY)

//****** Functions declaration ******************
int getFT(void) // Get forces/torque information (fx, fy, fz, Mx, My)
{
    unsigned data, Cstatus;
    unsigned buffer1[50], buffer2[50]; /* buffer for received data */
    int i, j, k, m;
    static int stepnumber;

    int conv(unsigned *force, int t);

    fx=0; fy=0; fz=0; Mx=0; My=0;

    flag: i=0; /* initialise */
    /**********Giving OR command to FT sensor***********/
    _bios_serialcom(_COM_SEND, COM1, 'F');
    _bios_serialcom(_COM_SEND, COM1, 'T');
    _bios_serialcom(_COM_SEND, COM1, '\r');
    /*
    _bios_serialcom(_COM_SEND, COM1, 'O');
    _bios_serialcom(_COM_SEND, COM1, 'R');
    _bios_serialcom(_COM_SEND, COM1, '\r');
    
    /***********************************************************************************/

    while(1) {
```

```c
```
Cstatus = _bios_serialcom(_COM_STATUS,COM1,SETTINGS);

if (Cstatus & DATA_READY) { //check port's status
    data = _bios_serialcom(_COM_RECEIVE,COM1,0);
    //buffering received characters
    if((data<=57 & data >= 48) || (data == 32) || (data==45)||(data==10)||(data==13))
    /*The number represents a character. Thus a string of numbers
    represents the command from the F/T sensing system*/
    {
        buffer1[i]=data;
        i++;  
        if(buffer1[i-1]==10 && i>5 && i<40)
        {
            goto flag;
        }
        if(buffer1[i-1]==10 && i>=40){
            ****** Transfering information to buffer2 **********/
            j=0;
            for(k=0;k<i;k++)
            {
                if((buffer1[k]<=57 && buffer1[k]>=48)||(buffer1[k]==32)||(buffer1[k]==45))
                {
                    buffer2[j]=buffer1[k];
                    j++;
                }
            }
            **************************************************************************/
            fx=conv(buffer2,6);
            fy=conv(buffer2,12);
            fz=conv(buffer2,18);
            Mx=conv(buffer2,24);
            My=conv(buffer2,30);
            stepnumber++;
            break;
        }
    }
    return fx,fy,fz,Mx,My;
}

void init_FT(void)
/*This routine compensates for the gravitational effects of
the end effector and workpiece. It should be run once before
any F/T readings are taken*/
{
    ******Giving BS command to FT sensor**************
    _bios_serialcom(_COM_SEND,COM1,'B');
    _bios_serialcom(_COM_SEND,COM1,'S');
    _bios_serialcom(_COM_SEND,COM1,'');
    **************************************************************************/
int conv(unsigned *force, int t)
{
    int k = 0;
    int i, realft;
    double ft = 0;
    for (i = t; force[i] != ' ' && force[i] != '-'; --i)
    {
        ft = (force[i] - 48) * pow(10, k) + ft;
        k++;
    }
    if (force[i] == 45) ft = -ft;
    realft = ft;
    return realft;
}

A.3.1.2 Using VC++ as A Compiler

DCB dcb;          /* device control block */
BOOL fSuccess;
char cKb;
COMMTIMEOUTS timeouts;
int portid;
char *ComPort[] = {"COM1", "COM2", "COM3", "COM4", "COM5", "COM6"};

/* DCB and COMMTIMEOUTS are system-defined structures, */
/* HANDLE, BOOL, DWORD are predefined simple data types */
/* The used constants are also defined in the windows-header files */

printf("Port-Nummer [1..6]: ");
scanf("%i", &portid);
portid--;

hCom = CreateFile(ComPort[portid],
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,
    OPEN_EXISTING,
    0, /* no overlapped I/O */
    NULL); /* must be NULL for comm devices */
PERR(hCom != INVALID_HANDLE_VALUE, "CreateFile");

fSuccess = GetCommState(hCom, &dcb);
PERR(fSuccess, "GetCommState");

/* configure the port */
dcb.BaudRate = 9600;
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;
dcb.fDtrControl = DTR_CONTROL_DISABLE;
dcb.fInX = FALSE;

fSuccess = SetCommState(hCom, &dcb);
PERR(fSuccess, "SetCommState");

fSuccess = GetCommTimeouts(hCom, &timeouts);
PERR(fSuccess, "GetCommTimeouts");

/* Only to show the content of the COMMTIMEOUTS structure */
printf("Timeout-values:\n" "ReadIntervalTimeout = %u\n" "ReadTotalTimeoutMultiplier = %u\n" "ReadTotalTimeoutConstant = %u\n" "WriteTotalTimeoutMultiplier = %u\n" "WriteTotalTimeoutConstant = %u\n",
    timeouts.ReadIntervalTimeout,
    timeouts.ReadTotalTimeoutMultiplier,
    timeouts.ReadTotalTimeoutConstant,
    timeouts.WriteTotalTimeoutMultiplier,
    timeouts.WriteTotalTimeoutConstant);

    /* Set timeout to 0 to force that:
    If a character is in the buffer, the character is read,
    If no character is in the buffer, the function does not wait
    and returns immediately
    */
    timeouts.ReadIntervalTimeout = MAXDWORD;
    timeouts.ReadTotalTimeoutMultiplier = 0;
    timeouts.ReadTotalTimeoutConstant = 0;

    fSuccess = SetCommTimeouts (hCom, &timeouts);
    PERR(fSuccess, "SetCommTimeouts");

    WriteFile( hCom, "B", 1, &BytesWrite, NULL);
    WriteFile( hCom, "S", 1, &BytesWrite, NULL);
    WriteFile( hCom, "\r", 1, &BytesWrite, NULL);


 int getFT(void)
 { int i; char cKb;

     WriteFile( hCom, "O", 1, &BytesWrite, NULL);
     WriteFile( hCom, "R", 1, &BytesWrite, NULL);
     WriteFile( hCom, "\r", 1, &BytesWrite, NULL);

     i = 0;
     bLineEnd = FALSE;
     do{
        /* look for a character in the input buffer */
        ReadFile(hCom, &szLine[i], 1, &BytesRead, NULL);
        if (BytesRead > 0) {
            /* a character was read, show the character and the ASCII -
            Code */
            if (szLine[i] == EOL)  /* check end of line */
                bLineEnd = TRUE;
            i++;
        }
        if (bLineEnd || i > 70) {
            szLine[--i] = '\0';
            printf("\n%-s  (%3i characters\n)", szLine, i);
            if(i>30&&szLine[1]=='?'){
                fx=conv(szLine,6);
                fy=conv(szLine,12);
                fz=conv(szLine,18);
                Mx=conv(szLine,24);
                My=conv(szLine,30);
                printf("\nx=%d, y=%d, z=%d, Mx=%d, My=%d\n",fx,fy,fz,Mx,My);
                fprintf(fp_in,"%d	%d	%d	%d	%d",
                        fx,fy,fz,Mx,My);
                break;}
                WriteFile( hCom, "B", 1, &BytesWrite, NULL);
                WriteFile( hCom, "S", 1, &BytesWrite, NULL);
WriteFile( hCom, "\r", 1, &BytesWrite, NULL);
WriteFile( hCom, "O", 1, &BytesWrite, NULL);
WriteFile( hCom, "R", 1, &BytesWrite, NULL);
WriteFile( hCom, "\r", 1, &BytesWrite, NULL);
}
i = 0;
bLineEnd = FALSE;
}
if (kbhit())
cKb = getch();
}while (cKb != ESC);
return 1;
}

A.3.2 Head File for Position Getting, Hole Turning and Peg Moving

This following programs using VC++ as compiler:

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<dos.h>
#include <time.h>
#include <windows.h>
#include "try1.h"
#include "ntport.h"

/*header file rto include all motor functions*/
int fx,fy,fz,My,Mx;
FILE *fp_in;
unsigned int oldpos=0,pos=0;
float degx=0,degy=0;
define badd 0x300
#define porta badd+12
#define portb badd+13
#define portc badd+14
#define cont_pt badd+15
#define ao0h badd+8
#define ao0l badd+9

void step(char axis,char direct,int stepnum)
{
    int n,port,rsteps;
    int time_delay = 5;
    static char x_predir,y_predir;

    switch(direct){
    case 'c':
        if(axis=='x') {
            if(direct == x_predir) rsteps=stepnum;
            else rsteps=0+stepnum;
            ...
for(n=1;n<=rsteps;n++)
{
    Sleep(time_delay);
    Outp(portc,0x01);
    Sleep(time_delay);
    Outp(portc,0x00);
}

if(axis=='y') {
    if(direct == y_predir) rsteps=stepnum;
    else                  rsteps=0+stepnum;
    for(n=1;n<=rsteps;n++)
    {
        Sleep(time_delay);
        Outp(portc,0x04);
        Sleep(time_delay);
        Outp(portc,0x00);
    }
}

    break;

    case 'a':
    if(axis=='x') {
        if(direct == x_predir) rsteps=stepnum;
        else                  rsteps=0+stepnum;
        for(n=1;n<=rsteps;n++)
        {
            Sleep(time_delay);
            Outp(portc,0x03);
            Sleep(time_delay);
            Outp(portc,0x02);
        }
    }
    else
    {
        if(axis=='y') {
            if(direct == y_predir) rsteps=stepnum;
            else                  rsteps=0+stepnum;
            for(n=1;n<=rsteps;n++)
            {
                Sleep(time_delay);
                Outp(portc,0x0c);
                Sleep(time_delay);
                Outp(portc,0x08);
            }
        }
    }

    break;

    default :
    printf("ERROR,UNKNOWN DIRECTION");
    break;
}

if(axis=='x') x_predir=direct;
else          y_predir=direct;

int get_ai(unsigned int chan) //get ai ch 0
{
    int data=0;
    Outp(badd+2,0x02);
    Outp(badd+3,0x20);//reset a/d
    Outp(badd+2,0x00); //select index reg 0
Outp(badd+3,0x0f); //setup analogue input
Outp(badd+7,chan); //select channel 0-chan
Outp(badd+0,0x01); //set gain to 2
Outp(badd+4,0x01); //enable a/d
Outp(badd+2,0x02); //select index reg 2
Outp(badd+3,0x80); //set s/ware trigger
Outp(badd+2,0x02);
Outp(badd+3,0x00); //reset s/ware trigger
while(!(Inp(badd+4)&0x80));
data=Inp(badd);
return(data);

void reset(void)
{
    unsigned int i;
    Outp(porta,0x03);
    for(i=0;i<1000;i++);
    Outp(porta,0x07);
}

unsigned int get_pos(void)
{
    //modified for 1000ppr encoder
    long int i;
    unsigned int data;
    Outp(porta,0x05);
    for(i=0;i<1000;i++);
    Outp(porta,0x04);
    for(i=0;i<1000;i++);
    data=((Inp(portb)& 0x00ff) <<8);
    Outp(porta,0x05);
    for(i=0;i<1000;i++);
    data=data+(Inp(portb) & 0x00ff);
    Outp(porta,0x07);
    for(i=0;i<1000;i++);
    return(data);
}

void move_pos(float target)
{
#define ao0 badd+8
unsigned vout;
int pos,flag,old_pos,pos0;
float kp=10,out=0,kd=5,ki=.30,error=1.,old=0,sum=0,target1;
pos=get_pos();pos0=pos;
printf("move_pos pos%d target=%f vout=%d sum=%f
",pos,target,vout,sum);
while (target!=pos)
{
    error=(target-pos)/1.0000;
    out=error*kp+(error-old)*kd+sum*ki;
    out=out+2048; //covert to 12 bit;
    if (out>4095) out=4095;
    if(out<0) out= 0;
    vout=out;  //cast to integer
    /*send 12 bits of vout to port 8 lsb and 9 msb*/
    Outpw(badd+8,vout&0xfff);
old=error;
sum=sum+error;
if(sum>1000) sum=1000;
if(sum<-1000) sum=-1000; //limit the integral windup
pos=get_pos();

if((pos-target)*(target-pos0)>=0)break;

if(pos==old_pos)flag++; else flag=0;
if(flag>20&&abs(error)<20)break;
old_pos=pos;
}

vout=2048;
Outpw(ao0,vout);
}

void inc_pos(float target)
{

unsigned vout;
int pos,flag,old_pos,pos0;
float kp=10,out=0,kd=5,ki=.10,error=1,old=0,sum=0,target1,target2;
pos=get_pos();pos0=pos;
target2=target;
target=target+pos; //set target to current pos plus increment sent

while (target!=pos)
{
    error=(target-pos)/1.0000;
    out=error*kp+(error-old)*kd+sum*ki;
    out=out+2048;//covert to 12 bit;
    if (out>4095) out=4095;
    if(out<0) out= 0;
    vout=out;    //cast to integer
    //send 12 bits of vout to port 8 lsb and 9 msb
    Outpw(badd+8,vout&0xfff);
    old=error;
    sum=sum+error;
    if(sum>1000) sum=1000;
    if(sum<-1000) sum=-1000; //limit the integral windup
    pos=get_pos();
    if((pos-pos0-target)*target>0)break;
    if(pos==old_pos)flag++; else flag=0;
    if(flag>20&&abs(error)<20)break;
    old_pos=pos;
}

vout=2048;
Outpw(ao0,vout);
APPENDIX B RELATED PUBLICATIONS

Journal paper


Conference paper


