Coping with Demonstration Suboptimality in Robot Programming by Demonstration

Jason Robert Chen

B.E. (Hons), University of Sydney

July 2001

A thesis submitted for the degree of Doctor of Philosophy of The Australian National University

Department of Engineering
Faculty of Engineering and Information Technology
The Australian National University

Declaration

The work contained in this thesis, except where explicitly stated, is original research, the major portion of which has been conducted by the author. He worked under the supervision of the members of the advisory panel, namely Professor Alex Zelinsky, Dr. Brenan McCarragher Dr. Matt James, and Dr. Henry Gardner. This work has not been submitted for a degree at any other university or institution.

Much of the research contained in this thesis has been published in book chapters and conferences, as listed below.

Book Chapters

[B1] J. R. Chen and B. J. McCarragher, "Configuration Space Generation for Assembly Tasks from Demonstration", In J.Billingsley, editor, Mechatronics and Machine Vision, pages 349-364, Research Studies Press Ltd., 2000

Conference Papers

[C5] J. Chen and A. Zelinsky, "Programming by Demonstration: Removing Suboptimal Actions in a Partially Known Configuration Space", Proceedings of the 2001 IEEE Conference on Robotics and Automation, Seoul, 21-26 May 2000.

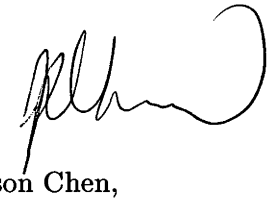
[C4] J. Chen and A. Zelinsky, "Generating a Configuration Space Representation for Assembly Tasks from Demonstration", Proceedings of the 2001 IEEE Conference on

Robotics and Automation, Seoul, 21-26 May 2000.

[C3] J. Chen and B.J. McCarragher, "Programming by Demonstration - Constructing Task Level Plans in a Hybrid Dynamic Framework", Proceedings of the 2000 IEEE Conference on Robotics and Automation, pp. 1402-1407, San Francisco, 24-28 April 2000.

[C2] J. Chen and B.J. McCarragher, "Programming by Demonstration using Hybrid Dynamic System Modeling", Proceedings of the Australian Conference on Robotics and Automation, pp. 138-143, Brisbane, 30 March - 1 April 1999.

[C1] J. Chen and B.J. McCarragher, "Programming by Demonstration - Building Event Paths in a Hybrid Dynamic Framework", Proceedings of the 1998 IEEE Conference on Robotics and Automation, pp. 518-523, Leuven, 16-20 May 1998.



Jason Chen,

Department of Engineering,

Faculty of Engineering and Information Technology,

The Australian National University,

Canberra, ACT 0200, Australia.

July 2001.

Acknowledgements

First and foremost, thank you to my supervisors Prof.Alex Zelinsky, Dr.Brenan McCarragher, and Dr.Matt James, who all provided valuable feedback and support at different stages of the PhD. Thank you also to others at the ANU who made life easier and/or more enjoyable: to Peter Aigner and David Austin for showing me in the early days how the system worked, to Tomasz Celinski for the many interesting discussions we had on topics I cannot even begin to recall, to James Macnicol for your assistance in the vagaries of the Sun operating system, to Bruce Mascord for prompt production of experimental apparatus, and to Sue Cameron and Josephine Farmer for your help with the wealth of administrative matters that arose during the course of my degree. In my personal life, thank you to my wife Ingrid. It was you who provided support when the pressures of the degree become great. Thank you also to my Family, who sustained infrequent visits because of my commitment to this degree.

J.R.C.

July 2001

Abstract

Finding a simple but powerful robot programming method for realistic tasks has been one of the main aims of robotics researchers for over two decades. A promising approach is robot Programming by Demonstration (PbD). Here, a demonstration of the task is interpreted by a PbD interface so that a set of control commands to achieve the task are produced for the robot. PbD is a promising approach to robot programming, however a well known weakness of the method is that human demonstrations can be suboptimal. Research has identified that demonstrations can contain inconsistencies, noise, or even incorrect or unintended actions. In this thesis our focus is on identifying and removing sub-optimality from the demonstration. Our aim is to ensure that control commands formed for the robot from demonstrated actions encode efficient and reliable execution of the task. The work we propose is divided into three distinct areas. They are: (i) determining task-specific, geometric properties of a task from demonstration, (ii) deriving efficient, low-level, robot control-commands from demonstration, and (iii) determining optimal task-level strategies from demonstration.

Research area (i) is important since knowledge of task geometry can help identify the presence of suboptimal actions in the demonstration. Our solution uses the concept of Configuration Space (C-space) as a means to represent task geometry. We apply statistical regression analysis to build-up a knowledge of task geometry in regions of the task that were visited in the demonstration. Experimental results showed the validity of the approach. The two key results were, first, that only a partial representation of C-space could be

determined. That is, a representation of C-space was determined only for regions of the task visited in the demonstration. Second, the method could determine quite accurately the true geometric properties of the task, so long as it had sufficient information to do so. That is, C-space was derived accurately in regions visited often in the demonstration, and less accurately in less-visited regions.

Research area (ii) involves deriving low-level control commands for the robot from demonstration, and has been the main focus of research into removing demonstration sub-optimality in PbD to date. In this thesis we adopt the well known control regime of hybrid force-position control, and so the problem divides into two sub-areas: position control-command synthesis, and force control-command synthesis. For position control-command synthesis we present a novel method for path planning in a partially known C-space. The method has the advantages compared to other methods in the literature that, it can derive paths containing undemonstrated points, it is applicable to a task with any degree of freedom, and that it does not assume a set form of demonstration topology. A drawback of the approach is that, theoretically, in some circumstances a valid control command will not be derived. However, we show with experimental results for a realistic task how with appropriate tuning, the method will produce a valid set of position control commands. For force control-command synthesis, a well known characteristic of force commands recorded from a demonstration are that: they include friction, and that force sensors introduce into these forces a high frequence noise component. We use our knowledge of C-space to determine an appropriate direction for force control. We base the magnitude of force control commands on the forces commanded by the human in the demonstration, however we remove friction and noise using filtering and spline fitting techniques.

Research area (iii) has to our knowledge not been the subject of work in the PbD field to date. In this thesis, we identify that a set of demonstrations will contain many different task-level strategies. Some of these will result in more optimal robot performance of the task than others. We present a method for selecting a task-level strategy for the robot.

It is based on the concept of modeling skill in a task as a Hybrid Dynamic System. The Hybrid Dynamic System representation allows a demonstration to be represented at a task level as a sequence of discrete events. Our method is to form an automaton from the event sequences encoded by each demonstration. We use a set of metrics to determine a cost for each event in the automaton. Finally, an optimal task level strategy is produced by conducting a least cost path search in the automaton. We show with experiments how the task-level strategies selected by our method do in fact result in more optimal robot performance of the task.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms and Abbreviations

CC	=	Continuous Controller
C-space	=	Configuration Space
C-surface	=	Configuration Surface
DEC	=	Discrete Event Controller
EPP	=	Event Path Planner
HDS	=	Hybrid Dynamic System
PM	=	Process Monitor
RMSE	=	Root Mean Squared Error
SSE	=	Sum Squared Error

Symbols

$\mathcal{A}$	=	automaton of discrete states in the task
$\mathcal{A}_D$	=	automaton of states in the task visited in the demonstration
$\mathcal{A}_D^*$	=	automaton of contact defining states visited in the demonstration
$\{b, c, d, e, f\}$	=	set of true parameter values for scalar equations ϕ_1 and ϕ_2
$\{\hat{b}, \hat{c}, \hat{d}, \hat{e}, \hat{f}\}$	=	set of estimate parameter values for scalar equations ϕ_1 and ϕ_2
$c_i^{\mathcal{A}}$	=	C-surface for the i^{th} distinct state in the task
$c_{ij}^{\mathcal{A}}$	=	primitive C-surface defined by constraint $\rho_{ij}^{\mathcal{A}}$
c_m^*	=	surface in C-space defined by ρ_m^*
c_w^{λ}	=	C-surface for the w^{th}state in desired state path λ
ct	=	PM information primitive describing constraint type
C_{con}	=	contact defining part of C-space
C_{free}	=	obstacle-free part of C-space
C_{obs}	=	non-obstacle-free part of C-space
Ct_k	=	cost of event $\tau_k^{\mathcal{A}}$ in the time performance area
Cr_k	=	cost of event $\tau_k^{\mathcal{A}}$ in the reliability performance area
Ce_k	=	cost of event $\tau_k^{\mathcal{A}}$ in the control effort performance area
Cn_k	=	cost of event $\tau_k^{\mathcal{A}}$ in the number-of-demonstrations performance area

C_k	=	overall cost assigned in $\mathcal{A}_D$ for event $\tau_k^{\mathcal{A}}$
D	=	the demonstration set
D_p	=	the p^{th} demonstration in the demonstration set
dst	=	distance from boundary segment at which point Q will be generated
eps	=	minimum distance from boundary segment at which a point Q can be generated
$\mathcal{D}$	=	graph describing connectivity between points on a C-surface in interior segments
$\mathbf{f}_p$	=	force control vector in $\mathbf{F}(t)$
$\breve{\mathbf{f}}_p$	=	demonstrated force vector at point $\mathbf{p}$
${}^{prj}\breve{\mathbf{f}}_p$	=	projection of $\breve{\mathbf{f}}_p$ onto the force control space $\mathcal{F}_p$
$\mathbf{f}_0^\eta$	=	force command derived for point $\mathbf{p}_0^\eta$
$\mathbf{f}_{n_l+1}^\eta$	=	force command derived for point $\mathbf{p}_l^\eta$
f_0^η	=	magnitude of force vector $\mathbf{f}_0^\eta$
$f_{n_l+1}^\eta$	=	magnitude of force vector $\mathbf{f}_{n_l+1}^\eta$
$\mathcal{F}_p$	=	force control subspace at point $\mathbf{p}$ on a C-surface
$\mathbf{F}(t)$	=	force control component of Hybrid Controller Command
gl	=	PM information primitive describing constraint gain or loss
g_s	=	equation describing the s^{th} constraint in the task
$\mathcal{K}$	=	graph describing connectivity between all generated points on a C-surface
$\mathcal{L}$	=	graph describing connectivity between points on a C-surface in likely-free regions
mcd_d	=	maximum connected distance between two points in a distinct interior segments
mcd_l	=	maximum connected distance between two points in likely-free-regions
md	=	maximum distance from boundary segment at which a point Q can be generated
n	=	dimension of C-space
$\mathbf{p}$	=	position control vector in $\mathbf{P}(t)$
$\dot{\mathbf{p}}_p$	=	time derivative of $\mathbf{P}(t)$ at $\mathbf{p}$
$\mathbf{p}_{w_s}^\lambda$	=	start point in the $\mathbf{P}(t)$ derived for state γ_w^λ
$\mathbf{p}_{w_e}^\lambda$	=	end point in the $\mathbf{P}(t)$ derived for state γ_w^λ
$\mathbf{p}_l^\eta$	=	generic undemonstrated point in a segment η of $\mathbf{P}(t)$
$\mathbf{p}_0^\eta$	=	demonstrated point in $\mathbf{P}(t)$ prior to segment η
$\mathbf{p}_{n_l+1}^\eta$	=	demonstrated point in $\mathbf{P}(t)$ following segment η
$\mathbf{P}(t)$	=	position control component of Hybrid Controller Command
Q	=	generic point in likely free region
R	=	generic point in boundary segment
S	=	basis of force control subspace $\mathcal{F}_p$
$\acute{S}$	=	orthonormal basis of force control subspace $\mathcal{F}_p$
$\mathbf{u}(t))$	=	control input vector in the continuous-time system
W_t	=	weighting value in the time performance area
W_r	=	weighting value in the reliability performance area
W_e	=	weighting value in the control-effort performance area
W_n	=	weighting value in the number-of-demonstration performance area

$\mathbf{x}(t)$	=	state vector in the continuous-time system
$[y, z, \theta, \delta]^T$	=	configuration vector in spindle-assembly task
ϕ_1,ϕ_2	=	generic scalar equations for the c_m^* in the spindle assembly task
$\phi_{\rho_{ij}^A}$	=	equation for the j^{th} primitive C-surface defining c_i^A
γ_i^A	=	i^{th} possible discrete state in the task
γ_r	=	r^{th} discrete state visited in an assembly sequence
$\gamma_r^{D_p}$	=	the r^{th} discrete state in the demonstrated sequence D_p
γ_w^λ	=	w^{th} distinct discrete state in state sequence λ
η	=	generic sequence of points on a C-surface
λ	=	desired state path
ρ_{ij}^A	=	the j^{th} constraint existing in the true constraint set Ω_i^A
$\hat{\rho}_{ij}^A$	=	the j^{th} constraint existing in the constraint set estimate $\hat{\Omega}_i^A$
$\hat{\rho}_{rs}^{D_p}$	=	the s^{th} constraint existing in the constraint set estimate $\hat{\Omega}_r^{D_p}$
ρ_m^*	=	the m^{th} constraint in Ω^*
τ_k^A	=	k^{th} possible event in the task
τ_w^σ	=	w^{th} distinct event in event sequence σ
Ω_i^A	=	the set of motion constraints existing in i^{th} distinct state in the task
$\hat{\Omega}_i^A$	=	estimate of the set of motion constraints existing in i^{th} distinct state in the task
$\hat{\Omega}_r^{D_p}$	=	estimate of the constraint set for the r^{th} state in the D_p
Ω^*	=	the set of unique constraints existing in D
Ω_w^λ	=	set of constraints defining state γ_w^λ

Chapter 1

Introduction

1.1 Introduction

Early visions of robots were of household helpers: robots to clean the house, help with the shopping, or to wash the car. In reality, today almost all robots exist in factory environments [53]. Recently there has been a resurgence of interest in expanding robotic application out of the factory and into domestic type environments; so called service robotics [8]. Recent publications show robotic research in areas such as health care [83], recreation/entertainment [71], domestic transport [18], education [70], construction [100] and the home [50]. Robotics research in these areas is certainly a worthwhile exercise. Schraft [94], for example, predicts enormous economic, scientific, and social payoffs if the transition of robots out of the factory and into service environments can be successfully made.

A major difference between service environments and the more traditional, industrial domain of robotics, is that robots must be able to interact with non-technical users. Such users will in general be unfamiliar with computer technology, and will almost certainly have no experience in robotics. Then an important question is one of how should users communicate with robots in these domains? More specifically, how should these users specify to the robot what tasks they want done? This is an issue of robot programming.

Current methods of programming used in industry are not suitable for service environments. The two main methods of robot programming in the factory today are (i) writing code by hand, and (ii) teach pendant methods; where the robot is lead through a series of points that are recorded and played back directly at run-time. These programming methods are clearly not suitable for use in service environments. First, writing code is a skill not possessed by many end users in typical service environments. Second, it is well known that teach pendant methods, while natural for the programmer, are limited to programming non-contact type tasks [2]. Clearly, a new approach to robot programming for service environments is needed. We see three required characteristics for robot programming systems operating in service environments. They are:

1. The system requires a non-technical user interface, preferably one that is natural and intuitive for humans. By *non-technical*, we mean that it is usable by end-users not familiar with either robotics or computer-science techniques and terminologies.

2. It can program the robot to complete typical everyday service-robotic-type tasks, ie. it copes with tasks involving contact and constrained motions between task objects.

3. It has *a-priori* information requirements about the task to be programmed that are realistic for a service-robotic type environment. For example, unlike in the factory, a full geometric model of the task (eg. a CAD model) is unlikely to be available.

A promising approach to robot programming that fulfills these three requirements is *Programming by Demonstration* (PbD). Here, robot programming is based on a demonstration of the task provided by the end user. The end user provides a demonstration of the task, and a PbD interface lying between the robot and demonstrator interprets the demonstration and determines the control details required by the robot . Two phases to PbD exist. The first is the demonstration phase, where the information about how to execute the task is extracted from the demonstration. The second phase is the execution phase, where the PbD interface controls the robot to achieve the task. We show in Figure 1.1 a schematic representation of the PbD approach.

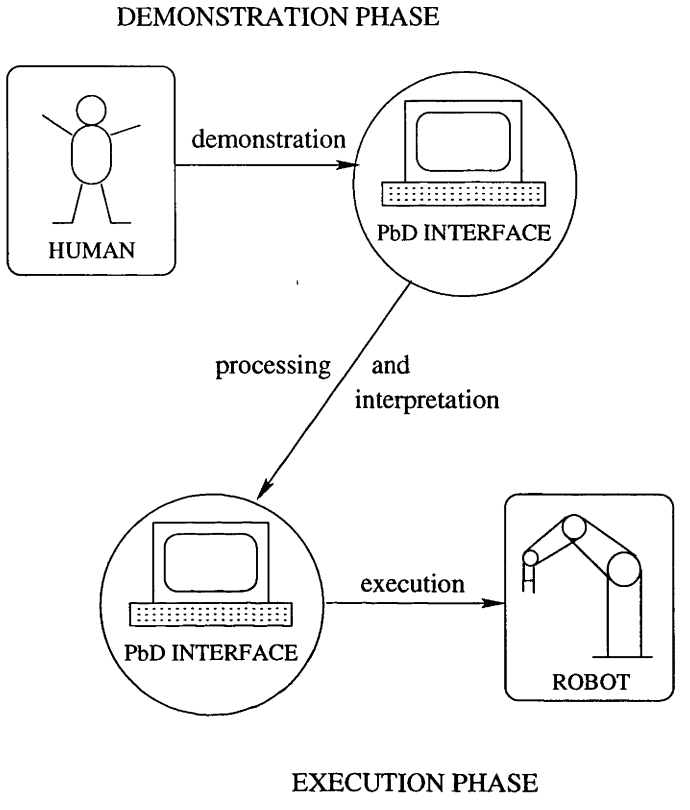


Figure 1.1: Schematic representation of the PbD robot programming method

PbD is a method that promises to fulfill our three requirements for the following reasons. First, it provides a very natural and intuitive programming method for humans. For example, work by Patrick [58] showed that in human-to-human interaction, instruction by demonstration and practice is the preferred mode of skill transfer. Second, PbD can be used to program a wide range tasks. Its only limitation is that the human can complete the task himself. The strength of PbD in this respect is reflected by the diverse range of tasks presented for PbD in the literature, eg. door-opening [63], metal grinding [117], peg-in-hole assembly [63, 90], balancing an inverted pendulum [7], sandwich making [123], driving an automobile [20], etc. Third and finally, limited a-priori information about the task is required because there is much task-specific information available in the demonstration itself. We shall see later in this thesis how task-specific, geometric information can be

obtained from the demonstration phase in PbD.

While PbD holds much promise, there is still work to be done. A well known weakness with the approach is that a demonstration can often contain sub-optimality. For example, work by De Schutter et al [90, 89], Delson and West [79], and Dillmann et. al. [63, 74] has identified that a demonstration can contain suboptimal actions by the demonstrator, eg. actions that are non-essential, erroneous, or even unintended. Sub-optimal actions of this type obscure the skillful set of actions required to complete the task, and can be viewed as *noise*. Noise of this type is such a problem in PbD because PbD is an implicit learning approach [45]. That is, instruction on how to complete the task is based on examples. If the examples are corrupted by noise, then learning skillful execution of the task becomes very difficult.

Then an important area for focus in PbD is on methods that make it robust to noise in the demonstration. Our work in this thesis has exactly this focus. We present methods that allow noise to be removed from a demonstration so that a noise-free, efficient set of actions that achieve the task can be programmed into the robot. We will present details of our work in following chapters. However, prior to that presentation, for the remainder of this chapter we first survey work in the literature that is related to, and important for, the work we will present.

The remainder of the chapter is set-out as follows. In Section 1.2 we review methods of robot programming that are not PbD. We identify five distinct approaches to robot programming, and highlight the advantages and limitations of each. In Section 1.3 we formally introduce PbD as a robot programming method. In Section 1.3.1 we compare PbD to the other programming approaches, and discuss why we feel it is more suitable for robot programming in general, and for programming in service robotic environments in particular. In Section 1.3.2 we review work in the PbD research field, discussing the virtues and drawbacks of different approaches. In Section 1.3.3 we make some observations about where new work is required if PbD is to become more robust to noise in the demonstration. Finally, in Section 1.4 we present the specific contributions of this thesis to the literature,

followed in Section 1.5 by an outline of the chapters in this thesis.

1.2 General Approaches to Robot Programming

Automatic robot programming is not a new concept. Work as far back as the mid 1970's recognised the need to create a simpler and more powerful robot programming interface [95, 67, 126]. A wealth of work has been published in the area since that time. In this section our aim is to provide a review of this work. We divide the work in our review into five sections:

1. Robot programming languages
2. Modular code libraries and tool-sets
3. Simulation and graphical user interfaces
4. Natural language
5. Virtual reality

1.2.1 Robot Programming Languages

Much of the early work focussed on the solution of creating a programming language specifically for robotics [130, 3]. The idea was that a programming language specifically tailored to robotics could simplify the programming process. Similar to developments in computer science at the time, the language could be given a command set that was abstracted away from the low-level details of underlying hardware. Programmers could be relieved of specifying arduous, low-level details, and could work at a higher level. Quite a number of robotic specific languages were proposed [107]. These languages can be classified into four distinct levels, according to the degree of abstraction achieved [2]. The four levels are:

- *Joint level*: motions are specified in terms of the required position of the robots' joints. While textual input is possible, in many cases, languages at this level use a teach pendant. The robot is moved through a sequence of desired points, with the joint positions recorded at each point. The sequence of joint positions is then played back at run time.

- *Cartesian level*: the positions of the manipulator and its movements are specified directly in the Cartesian Space, usually in textual form. Most languages at this level extend more traditional languages such as BASIC and PASCAL by including robotic specific commands [15]. This level represents the current state of the art in commercially available industrial robots, eg. VAL-II [13] (which has been used by the robotics companies Unimation and Adept). Other languages proposed at this level include AML [96], AL [119], and LM [31].

- *Object level*: a representation of objects in the workspace exists, simplifying the description of operations on objects. The programmer specifies the spatial relationship required between objects, and the system determines the required positions in the workspace for objects so that the spatial relationships are satisfied. Note however that, at this level, the system does not automatically determine how the assembly should proceed. That is, obstacle avoidance and fine motion capabilities still need to be specified manually by the programmer. RAPT [97, 6] and LEO-GM[30] are two examples of languages proposed at this level.

- *Task level*: An assembly is specified only at a very high level. The programmer might expect to specify the assembly to the robot in as much detail as if specifying it to another human. Commands such as *insert* object A into object B, or *align* object A with object B, are examples of commands in these languages. No details of how the robot should achieve the insertion or alignment are required of the programmer. The programming system should generate these details automatically.

As a natural robot programming interface in a domestic environment, languages at the joint and manipulator level are not suitable. These languages require significant programming expertise and a detailed knowledge of many aspects of robotics. Languages at the object level simplify programming compared to those at the joint and manipulator levels, however the programmer must still specify all details of motion of objects in the work space. Most promising as a natural programming interface are languages at the task level. Here, the programmer is isolated from the tedious, low-level details of program generation, providing a comfortable and relatively [1] natural programming interface. Two languages at this level were proposed some years ago, LAMA [126] and AUTOPASS [67], however neither were completed. The problem to be solved for creating a task level language is of how to generate automatically the low-level control details for the robot from the high-level commands specified by the human. This is an extremely difficult problem that for general, real-world tasks , still remains open today.

1.2.2 Modular Code Libraries and Tool-sets

Another approach to robot programming in the literature is via modular code libraries and tool-sets. The idea here is that robots usually exist to carry out a variety of tasks, eg. painting, deburring, welding, etc. Many of these tasks share common functionality. For example, surface following is a robot functionality necessary for both welding and grinding tasks. The purpose in this work is to create a library of reusable code *modules*, where each module corresponds to a certain robot functionality. Programming is simplified because the programmer need only call modules in the library in such a way as to achieve the desired task.

A good example of work of this type is proposed by Borelly et. al. [25, 52]. They present a programming architecture called ORCCAD, where the programmer creates core robot functionalities called RTs (Real-time tasks). Each RT has an input, an output, and 3 levels of possible interrupt (weak, strong and fatal) by a central, coordinating process. The

[1] The programmer is still required to write code, albeit at a very high level of abstraction.

RTs can be nested in a recursive structure called RPs (Robotic Procedures) to represent meaningful high level robot behaviors. The idea behind this structure is that the same core set of RT's can be used to encode many different RP's. An example of an underwater vehicle fitted with an arm for manipulation is provided [24]. RP's for both the arm and vehicle body are built up from RT's representing the core functionalities of each device. In turn, a high level RP is constructed from the RP's of the arm and body to create a unified control framework for the entire vehicle. Work with a similar focus has also been proposed by [92, 103].

One of the key questions with this approach is of what functionalities a module in the library should possess. Troxell and Davis [134] propose four core functionalities for an IBM 7565 robot and a Soma cube assembly task. They were PICKUP, REGRASP, PUTDOWN, and PATTING. However, it is not made clear in this work how these functionalities were chosen. Uechi, et. al [76] propose a method for determining core robot competencies for industrial-type tasks. They identify common elements of tasks such as welding, sanding, polishing, etc. by focusing on the tool used in each. Three properties of the tool were used, workpiece condition, geometry condition and inherent condition. By using these classifications, they show how tasks such as welding and sealant ejection, or spray painting and air blasting require the same base robot functionality. They suggest a framework that allows such functionalities to form code modules in a reusable programming library. Another approach of this type is presented by Wenrui and Kamkper [26, 27]. They identify core functionalities for welding robots. They propose a number of often repeated actions that can form reusable "macros" in a central library. These include certain sequences of welding torch motion, TCP measurement, sensor-based calibration and search operations, and program initialisation. Marchand [29] also investigates modular programming methods in robotics, however he focuses on programming visually servoed robots . He identifies the core competency for this robot type as an elementary positioning task with respect to some "control feature" in the environment. Possible control features were identified as points, lines, circles, spheres, and cylinders. Experiments were shown where

the robot was programmed using the approach to track the edge of a long curved pipe (ie. positioning w.r.t. a line - the pipe edge tangent) and a ping pong ball (ie. positioning w.r.t. a sphere).

Programming using modules and tool-sets is closely related to the robot programming languages approach. Each code module in the library can be viewed as a different high-level command in a language. However, this approach attacks the problem from the opposite direction. Instead of needing to generate control details from a high level command, the control details are known (ie. they exist in the code modules). However, the problem here is in finding a small enough set of modules that encode control details applicable in a sufficiently general number of tasks. Work in the field has presented code modules that are reusable across some tasks. However, a set of modules that (i) are small enough in number to be workable, and (ii) are powerful enough to program a wide range of general tasks, is still yet to be found.

1.2.3 Simulation and Graphical User Interfaces

With the evolution of ever more powerful computer graphics, a natural approach to robot programming is by using simulation and graphical user interface (GUI) techniques. Early work in this field used these graphics capabilities mainly as a means for checking the validity of programs. One of the earliest systems of this type was proposed by Arai [121]. A simple 2-D graphic simulator called EARLS-2 was constructed to check programmed motions for a SCARA robot. Motion commands to achieve a simple palletising task were checked by the programmer via a simulation shown on a screen. Imam and Davis [1] proposed a similar approach, however their system supported simulation in 3-D. Program verification was achieved by showing a tool-tip trace in a 3-D rendering of the workspace. In addition, their system supported automatic collision checking for the robot's tool-tip. Systems with similar functionality were also proposed in [17, 5, 82]. More recently Zeghloul et. al. take advantage of advances in computer graphics and processor speed [120]. They show a full 3-D simulation of a welding robot and its workspace. Collision checking for the

entire robot arm, rather than just the tool-tip, is carried out by the system automatically. They simulate a complex welding task in a work-cell containing 38 degrees of freedom and over 250 distinct geometric objects. While this system facilitate program validation, it still requires the user to program the task by writing code.

In other work, graphical methods have been proposed for task specification as well as validation. For example, Arai and Yago [122] propose a graphical user interface for specifying welding tasks. A 3-D perspective view of the parts to be welded are shown on the screen. Alongside, a form-type, GUI interface allows the user to enter a step by step welding sequence, along with the details of the weld to be completed at each step (eg. length, direction, etc). Once completed, a simulation of the robot executing the program is shown to ensure correct operation. The interface was used to successfully program two welding task, one involving a collar on a shaft, and the other, a piece of corrugated sheet material. Lees and Leifer [22] also propose robot programming via graphical methods. They create a system for use by severely disabled persons on a standard PC in the home. Programs are assembled using a mouse to drag a sequence of icons onto a storyboard. Each icon represents a certain robot action, and are divided into position and force types. Position type icons move the robot to particular poses, with fine tuning possible by the user with the mouse. Force type icons specify more autonomous motions, eg. plane finding, guarded moves, contour following, etc. Validation of the system was presented in [51] where its intuitivity was proven on a microwave door opening task. Other, similar work that propose GUI's for robot program specification are presented in [56] and [142]. While GUI's facilitate programming compared with programming language approaches, we see them as suffering similar problems. Draggable icons, menus, toggle buttons, etc. all specify at a task level an action for the robot to execute. In the end, the programming system is required to solve the same problem as for robot programming languages: to determine automatically from task-level commands the low-level control details for the robot.

1.2.4 Natural Language

Another approach proposed in the literature for robot programming is by using commands expressed in natural language. Humans communicate easily and effectively between themselves in high-level languages like English. Then having a human specify tasks to a robot using natural language promises an easy and natural method for humans to program robots.

Liang et. al. [68] propose a robotic system that can be instructed by a non-expert user in an office environment using natural language. They investigate the problem of transforming ordinary English utterances into unambiguous task structures that can be performed by a robot. Two requirements on a natural language programming interface are identified. First it must support commands specified at a "robot-level" so that communication is efficient. Second, it must cope with the way humans refer to actions, and to temporal and logical relationships, when communicating in a natural language. They propose a system based on concurrent, perception-feedback-driven control to program a simple book handling task in an office-like environment.

Michalowski et.al. [118] also investigate natural language as a robot programming interface. They are attracted to the natural language approach because it allows real-time commanding of the robot by the user. They report initial results for a nine degree-of-freedom robot consisting of an arm mounted on an omnidirectional mobile base. A control architecture is implemented so that the user can direct the motions of the mobile base in real-time by means of commands expressed as colloquial English sentences.

Crangle [16] investigates a programming interface supporting conversation between the robot and user. In this work, communication can also flow from the robot to the user, eg. to clarify a command or request information. Crangle argues that conversation in natural language is possible so long as it occurs in a fairly restricted domain. A hospital is proposed as such a domain. Here, the set of tasks to be completed, and the set of objects present are small enough to allows conversation to be natural for users without being

too complex for a robot system to interpret. Two examples of functional conversational interfaces are presented, one which allows users to access online medical information, and the other to teach medical students anatomy.

There has also been work presented in the AI field on robot programming using natural language [104, 28]. This work focuses more on learning issues. For example, Huffman and Laird [104, 105] refer to programming through natural language as *interactive tutorial instruction.* They identify that, to be useful in a general environment, an instructable agent must be able to learn different types of knowledge from different instructional interactions with the user. An approach is presented that uses the constraints present in different instructional contexts to guide the learning process. The theory is implemented on an agent called Instructo-Soar, that learns new tasks and other domain knowledge from natural language instructions. Koenig [28] proposes parallel processing for interactive man-robot systems when the interaction is via natural language. Commands received by a robot from a master in a natural language are treated as incompletely stated arguments. An attempt is made by the robot to seek out missing premises or conclusions that will produce valid arguments on the basis of inference rules. Parallel processing is presented as a means to cope with the substantial computational requirements of the approach.

Natural language as a secondary or tertiary mode of communication in robot programming have also been presented [54, 141, 59]. CURL is a robot programming language presented in [54, 55, 56] which allows for voice control of an iconic interface. MUSIIC [141, 142] is a robot programming interface that includes, among other things, a natural language communication mode based on simple dialogue models. Merlet uses a natural English-like language in [59] for programming a hybrid, position-force-controlled robot to perform fine motion tasks.

Despite providing a very natural programming interface for humans, work in this field suffers the same difficulties as for the robot programming language approach. That is, despite being communicated verbally, high-level commands proposed in this work still need to be converted into low-level control details for the robot.

1.2.5 Virtual Reality

The previous four sections have focussed on methods of *explicit* robot programming. That is, programming is achieved by explicitly specifying how the task is to be done. An alternate approach to programming is by *implicit* methods. We have already seen that PbD is a method of this type. Another implicit programming approach is by using virtual reality techniques. Here the user *shows* how the task is to be done in a virtual space. It is different to GUI methods because the task is shown rather than specified via icons, menus, etc. It is different to PbD because the task is shown in a *virtual* space rather than a real-world workspace. As we shall see, this fact makes the two methods crucially different.

A key characteristic of work in this field regards the level of immersion into the virtual world that it allows. Lloyd et. al. [49, 48] propose an approach that supports little immersion. Here the human interacts with the virtual world using only a PC monitor and mouse. These authors concentrate on programming contact tasks. A grey-scale vision system is used to construct a model of a simple blocks-world environment. By using the mouse, the operator can select and move different blocks around the workspace. Contact interactions are supported, allowing the operator to bump, slide, and align the manipulated block with others. Once the task is completed, a program for the real robot is generated. During this step, potential-field path-planning techniques are used to remove extraneous motion segments, and to maintain adequate distance from objects with which contact is not desired. Impedance control is used to achieve desired contacts during task execution in the real-world.

A system where the programmer is more immersed in the virtual environment is proposed by Strommer et. al. [133]. Programming is achieved on a virtual robot in a virtual environment. The programmer senses in the virtual environment via visual (HMD: head mounted display), audio (earphone) and tactile (sensing glove) modes. The programmer can act on the virtual environment by simply moving his head or hand (which are tracked

by the system), or via special interface devices such as a 6D track-ball. Trajectories for the robot are specified simply by moving the hand. Relevant computer code is generated automatically for the real robot by using a special purpose toolkit call VR4 [46]. Special features of the system are, first, that it can generate graphics at close to real time by reducing the level of detail shown for distant objects. Second, collision detection between the virtual robot end effector and virtual objects is achieved quickly by initially using a fast bounding box check, followed by a more computationally intensive exact check only if necessary.

Yanagihara et. al. [139] also propose robot programming using virtual reality. They use virtual reality to program a seam welding task for a complicated car chassis. Their system consisted of (i) a 7 dof robot with laser range finder attached to the wrist, (ii) a human with teach pendant, HMD, and a head set (ie. microphone and earphones), and (iii) a video tracking system using two CCD cameras to track human actions. The operator teaches the system desired welding trajectories in the virtual space, with taught trajectories recorded via the video tracking system. The system even supports voice commands from the operator. Other, similar work where virtual reality is used for robot programming is presented in [37, 100, 125, 129] and [111].

A key advantage of the virtual reality approach over explicit programming methods is that it facilitates users in expressing their intentions to the robotic system. The user can show how a task is achieved, rather than specify how it is achieved. However, a disadvantage with the approach is that dynamics in the virtual world must exactly represent those in the real world if the programs that it generates are to be useful. The dynamics to be modeled include those of sensors, the robot manipulator, and contact between objects in the environment. Such real world dynamics are extremely complex and difficult to model. From what we have seen, modeling at this level of complexity is still beyond the current state of the art.

1.3 Robot Programming by Demonstration

1.3.1 Comparison to Other Approaches

A common difficulty we identified with the explicit programming methods of Sections 1.2.1, 1.2.2, 1.2.3, and 1.2.4 was that low-level control details needed to be determined from high-level task specification. These methods require that task specification be at a high level to make it natural and easy for the human. However, a consequence of this requirement is that low-level control details to complete the task need to be generated automatically by the programming system. We saw that an alternate approach to programming is via implicit methods. Here the mode of task specification is by showing. This allows an easy and natural specification method for the human. However, with this approach, the low-level control details to complete the task are also available, ie. in the demonstration.

Two methods of implicit programming have been presented, virtual reality and PbD. We saw that a key problem with virtual reality regards how accurately the real-world environment can be modeled. The dynamics of any real-world robot workspace are extremely complex. If modeling in the virtual world is not appropriate to the real-world situation, then programs generated may not be successful. PbD does not suffer this problem because the demonstration is provided in the real-world workspace. That is, the dynamics experienced by the user in the demonstration are the same as those that will exist when the robot executes the task.

We have presented the advantages of PbD as a method for robot programming. However, we note that the method is not without its difficulties. As an implicit programming approach, the idea in PbD is to construct a perception to action mapping by using examples provided in a demonstration. A mapping representing the skill in completing the task can then be generalised from the set of examples in the demonstration. One disadvantage of PbD is that this mapping can never be complete because only a finite set of examples are ever given. It may then be possible that a situation encountered by the robot during its execution of the task was not encountered by the human in the demonstration. How-

ever, while a complete mapping is desirable in an ideal world, in general a non-complete perception to action mapping is sufficient. It is not often that we need to explore every possible configuration of task objects in order to complete the task. In some ways this can also be viewed as an advantage of the PbD approach. That is, we can exploit the expertise of the human to only learn that part of the perception-to-action mapping that corresponds to skilled execution of the task.

A second limitation of PbD is that the demonstration must indeed encode skillful task execution. In general the level of intelligence and dexterity of the human well exceeds that of the robot system, so task performance by the human will be beneficial as an example for the robot. However we have already noted that human demonstrators can inadvertently introduce noise into the demonstration. Noise is something that can badly affect the performance of the PbD approach and methods are required to make it more robust in this respect. We have noted already that the focus of this thesis is on exploring such methods. We will explore the details of this question further in Section 1.3.3. However, prior to that discussion, we first survey in the next section the substantial body of work that exists in the PbD research field.

1.3.2 Literature Review

Robot Programming by Demonstration is an active research area where much work has been presented. A popular approach has been to use neural networks to "approximate" a skill mapping for a task. In this approach demonstration data is used to train the net so that it approximates the skillful perception to action mapping required to complete the task. Asada was one of the first to use this approach [39] . He proposed a multi-layered neural network to learn non-linear compliance strategies used by a human in a chamferless peg-in-hole task. Pomerleau also proposed a neural net approach. He transfers human road following skills to an automated driving system using a neural network which maps coarsely sampled video images to steering outputs [19, 20]. Using the technique, the vehicle was able to drive on a variety of roads at speeds up to 20 miles per hour. Zhang

and Xu [140] use a neural net to capture human skill for controlling a light source. A mobile platform is moved randomly to produce a shadow from a human held light in a pre-specified target area. Human strategies for moving the light source to remove the shadow are used to train the net. Koeppe and Hirzinger [98] use a neural net to map task geometry to a control command for a compliant motion controller. They identify the lack of richness of geometric information in force and velocity signals used in other work. They propose instead, as input for their neural net, signals from visual and kinesthetic sensing modes. The approach was verified on a low-tolerance, peg-in-hole task. Kaiser and Dillman [63] also propose neural nets for PbD, however this work includes a phase of reinforced learning in the the PbD process. These authors initially use PbD to program the robot, but recognise that robot performance can be improved if it is allowed to learn and adapt according to its own experience. Experiments are presented for peg-in-hole and door-opening tasks. They show how initially poor robot performance at the first execution attempt can be improved by repeated bouts of execution and learning. Other approaches that use neural nets in PbD can also be found in [115, 88, 137, 135, 99]. While the neural net approach has met with some success, there have been problems with training neural nets directly from demonstration because of inconsistent motion typically generated by human demonstrators. Essentially the issue is one of the neural net approximating a skill function that includes noise in the demonstration.

An alternate function approximation method to neural nets is proposed by Myers [23] and Dillmann et. al. [93]. They use fuzzy logic to produce a discretised description of the skill function required to complete a task. For example, Myers programs a simple 3D pallet insertion task where the range of demonstrated force/torques and velocities in the demonstration were each discretised into three fuzzy groups: zero, negative, and positive. Then, by identifying clusters of points in force-velocity space, these fuzzy groups were used to produce conditional statements in textural programs. For example, a fuzzy group may correspond to negative force in the y-direction and positive velocity in the x-direction. A conditional statement of "if Fy is negative then Vx is positive" is then included as part

of the textural program. A similar approach using only velocity signals is presented by Dillmann, Kaiser and Ude [93]. This type of work is more robust to noise than the neural net approach because irregularities in the demonstration can be "removed", ie. outliers to clusters can be ignored. However, it assumes the data can always be organised into easily distinguishable clusters. It is also difficult to know with this approach what sensory variables should form the space in which clustering is conducted.

A different method of skill function approximation is presented by Takahashi [128]. He parameterises sensory and control variables recorded in multiple demonstrations as functions of time. A time normalisation technique is used to make the same important events in each function match on a common time-scale. Wavelet multi-resolution analysis is then used to approximate each function as set of wavelets. The work focuses on using the wavelet description to identify the essential and distinctive elements of the demonstration. An assumption of the approach is that the same important events will exist in each demonstration, ie. that a human will always demonstrate a task in exactly the same way.

The methods reviewed so far are based purely on data. That is, their approach is to simply use function approximator methods on the demonstration data to derive a skill mapping for the task. A different approach is to assume, a-priori, some knowledge of the physics of the task. That is, a "model" of the skill required for a class of tasks is assumed a-priori. The details of the skill required for a particular task in the class is then obtained via PbD. This sort of approach can be advantageous for making PbD more robust to noise in the demonstration, since the model provides a framework into which demonstration data can be interpreted. Noise in the demonstration can then be more easily identified and removed.

Atkeson and Schaal propose an approach of this type to program a pendulum-swing-up-and-balance task [7]. A model of the task is derived based on laws of physics, resulting in an expression relating the pendulum's angular velocity to the acceleration of the hand at its pivot. Task demonstration by a human is used to seed a search for appropriate values for the robot of unknown parameters in the model. Liu and Asada also propose PbD using

a physics-based model [116, 117]. They derive a control law for a surface grinding task that relates the tool feed-rate and holding stiffness. They note that a human expert at the task will vary these parameters depending on such things as burr size and hardness. They use PbD to find values for these parameters appropriate for the robot. These works propose two task classes that can be programmed by PbD. However we note that these classes are quite specific, and are not likely to be found in service-robotic type environments.

A more general class to which many tasks in service robotic environments do belong is assembly, ie. contact and constrained motion tasks. A substantial amount of work has been proposed in this area. One of the first was presented by Asada and Izumi in [40]. Hybrid position/force control (as proposed by Raibert and Craig [73]) was assumed as the skill model. A demonstration of a simple place-block-in-corner task is used to determine the task-specific facets of the hybrid-control skill model, including such things as deciding on force and velocity controlled directions, finding compliance values for the force controller, and determining trajectories for the position controller.

Kang and Ikeutchi introduce a general skill model for grasping in assembly [106]. They identify three phases of grasping, (i) a pregrasp phase, involving hand transportation and hand preshape stages, (ii) a grasp phase, from where the hand makes contact with the object to when a stable hold is achieved, and (iii) the manipulation phase, when purposeful movement of the object relative to the environment is made. The work focuses on using the demonstration to automatically identify when these phases occur. Their approach is to use the volume sweep rate of the fingertip polygon area to identify when the breakpoints between these phases occur.

A common skill model for assembly is as a sequence of *discrete states.* For example, Hannaford and Lee [9], and Xu and Yang [138] propose Hidden Markov Models (HMM) as a skill model for assembly. That is, these authors view assembly as a sequence of states in a Markov process that are not directly observable. For example, Xu and Yang [138] propose two HMM model types for assembly skill. The first type models assembly where sensory feedback does not exist. It consists of a left-to-right HMM, where the Markov

states correspond to human mental states, and the observable output symbols correspond to measurable signals such as position and force [62]. The second HMM model type is proposed for assembly where sensory feedback does exist. It consists of a set of left-to-right HMM's, where each HMM encodes the mapping between a sensory data stream and control action. This work proposes states in the hidden Markov process as the mental states of the human operator. However it is not made clear what these mental states might be.

Hirai et. al. propose a discrete-state skill model for assembly involving deformable objects. The work looks at an assembly involving the insertion of a deformable tube onto a solid plug. Three states in the assembly were proposed: the approach state, the contact state and the insertion state. The work concentrates on detecting the transitions between these states based on position and force data obtained from a demonstration of the task. A skill model relevant to a more general class of deformable material tasks was not investigated.

Wang and De Schutter [90, 91] propose a discrete-state skill model for assembly formed by applying filtering and thresholding techniques to demonstration data. Demonstration data is first filtered to remove noise. Qualitative and quantitative thresholding rules are then applied to break the demonstration down into a sequence of states (subtask-segments in their terminology). The approach is used to identify subtask segments for a challenging peg-in-hole assembly. A disadvantage with this approach is that task states are variant. That is, a single demonstration of the assembly can result in a different sequence of states, depending on the values of thresholds used.

An invariant state definition for assembly skill based on geometry is proposed by Ogato and Takahashi [42, 41]. Their discrete-state skill model is formed by decomposing the configuration space (C-space [127]) of a task into a set of regions. These regions are formed from the constraint equations of obstacles in C-space. Each distinct region is defined as a task state. The skill of an assembly is then described as a sequence of discrete states. While this approach is workable for simple 2-D tasks, more complex tasks of higher

dimension can result in a very large number of possible states.

Ikeuchi, et al [64] [61] propose a different discrete-state skill model for assembly that is also based on geometry. They identify as states in their model, ten contact configurations types that can exist between two bodies in contact. The ten configurations are identified as distinct because each allows a different set of translational and rotational motion between the bodies. This definition has the advantage that each state implies a distinct control strategy to be adopted. However, it has the disadvantage that a state does not encode the "state-of-completion" of the task. That is, two contact configurations allowing the same translation and rotation between task parts will be identified as the same state, no matter if one occurs near task commencement and the other near task completion. Having states that encode the state-of-completion of the task can be important, since it is then possible to plan an assembly sequence for task completion as a unique sequence of states in the task.

An alternate discrete-state, assembly-skill model is proposed by McCarragher [11]. He introduces the notion of modeling assembly skill in PbD as a Hybrid Dynamic System (HDS). The approach defines as a state in the model, each unique contact configuration that can be formed between task objects. This definition has a number of advantages over others. First, it provides an invariant set of discrete states for a task. Second, a state definition of this type means the state-of-completion of the task is explicitly encoded into each state. Third, the number of states resulting from this definition is much less than for the approach in [42]. This is the case mainly because the approach in [42] breaks down the situation where objects in the task are not in contact into many states. In contrast, the HDS approach recognises the no-contact situation as a single state, ie. it provides a greater resolution of states in that part of the task where objects are in contact. Such an approach is appropriate, since the difficult nature of assembly comes about because of the presence of contact.

Skubic and Volz also propose a HDS modeling approach within PbD [114, 14, 75, 113]. This work looks mainly at using PbD to train a "state classifier" that can identify when

a new state in the task is reached. They train state classifiers based on both fuzzy logic, and on neural networks, to automatically recognise states in a place-block-in-corner type task. Sikka and McCarragher [85] and Hovland and McCarragher [36] also present work for deriving a state classifier by demonstration. Work in [85] proposes linear discriminant functions and clustering techniques as a means for automatically identifying from demonstration the states in the task. Work in [35] uses a multilayer perceptron neural net to achieve the same end. In this thesis we will adopt the Hybrid dynamic system approach to modeling assembly task skill. Note that we will formally introduce Hybrid Dynamic Systems, and how they model assembly skill, in Chapter 2.

1.3.3 Making PbD Robust to Noise

Work in the PbD research field has long recognised that a suboptimal demonstration can detrimentally affect the approach. For example, Kaiser, Friedrich and Dillmann [74] recognise five sources of sub-optimality that can exist in a demonstration: where the human demonstrates unnecessary, incorrect, or unmotivated actions, where there is choice of scenario regarding when to apply an action, and where the actions are demonstrated with the wrong intention (ie. the user does not know enough about the task). Delson and West [79] identify that, in a pick and place task through a field of obstacles, a human will naturally introduce noise into the demonstration by using different paths to traverse regions were the gap between obstacles is large. De Schutter et al [90, 89] found that á demonstration of a peg-in-hole task could contain actions by the demonstrator that were suboptimal, erroneous, or even unintended. Nechyba and Xu [136] identify that skill models obtained from human-produced training sets can produce trajectories not characteristic of the source process, or even worse, that are potentially unstable.

A number of approaches have been proposed to remove noise from a demonstration. Tso and Liu present a method to select the most consistent demonstration from among a number of demonstrations of the same task [112]. They identify that a human demonstrating the same task multiple times will execute varying motion, and propose that all

demonstrations will have in common a "core motion" required to complete the task . Their idea is to use Hidden Markov model to select the most consistent demonstration in the group, ie. the one with a trajectory closest to the core motion. While this approach can select the most noise-free trajectory that was demonstrated, it is limited to selecting one of the demonstrated trajectories, ie. it cannot modify trajectories.

In contrast, work by Kaiser and Dillmann, and De Schutter et. al. [63, 90] is able to remove noise from within a trajectory. This work uses thresholding, smoothing, and loop removal techniques to filter out noise. For example, Kaiser and Dillmann [63] propose two types of noise removal for a peg-in-hole task. First, they remove ineffective actions, ie. actions that changed the configuration of the peg by less than a predefined threshold. Second, they remove actions that were later corrected, ie. those that were partly or fully negated at following time steps. Removing actions that are negated at following time-steps is essentially the process of removing loops from the demonstrated path. Work by these authors has been successful in removing obvious flaws in the demonstration, however it is limited to deriving paths that contain only demonstrated points. As such, less obvious flaws cannot be removed, eg. corrected actions that do not form explicit loops in the path.

Delson and West [79, 77, 78] propose a different approach. They address the case where all demonstrated paths start and end at the same point. They then identify an obstacle-free envelope formed by the outer-most lying paths. A noise-free path is constructed to lie completely within the demonstration envelope. This approach can derive paths that contain points that were not demonstrated. As such, it is capable of removing sub-optimalities from demonstrated paths that work in [63, 90] cannot. However, the approach is restricted to finding paths that lie within the envelope. This can be a limitation, for example, where only one demonstration is provided. In addition, it has been presented for C-space of dimension 2 or 3. The method is not easily extendable to higher dimensions.

While the methods just reviewed have gone some way towards making PbD more robust to demonstration sub-optimality, more work of this type is still required. We identify three areas of further research that will help allow the problem of demonstration sub-optimality

in PbD to be solved. They are:

Determining geometric properties of the task:

We have not seen any work in the literature where the demonstration is used to determine geometric properties of the task to be programmed. Such an approach can be beneficial with regards to noise robustness and removal because knowledge of task object geometry provides a basis for better interpreting the demonstration, ie. for recognising and eliminating noise in the demonstration data.

Optimising low-level control commands:

This has been the main focus of noise removal techniques proposed in the literature to date, ie. the methods reviewed above [112, 63, 90, 79], however useful contributions in this area can still be made.

Selecting an optimal task-level strategy:

Until now, noise removal in PbD has focussed on optimising low-level control commands. This is a necessary and beneficial step. However, an important, additional part of PbD is to choose for use by the robot an optimal task-level strategy. To illustrate what we mean by a *task-level strategy*, we briefly give an example using a classic peg-in-hole task. We note two strategies that can be used to achieve such an assembly. First, one may move the peg down to be in contact with the surface surrounding the hole, and then use this surface to guide the peg to the hole entrance. Conversely, one may try to move directly towards the hole entrance from the initial starting position. Each sequence represents a different task-level strategy to achieve the assembly. A human may demonstrate any number of different strategies for a task. Some of these may be better than others. A good idea then is to select the strategies that will see the robot best perform the task.

It is in these three areas that this thesis will make a contribution to the literature.

1.4 Contributions

The specific contributions of this thesis can be summarised as follows:

i. A method for deriving a configuration space representation of an assembly task from demonstration. This is essentially the step of creating a geometric model of the task from demonstration. Configuration space can be viewed as a type of "geometric" model that focuses on the constraints on motion caused by one object in the task on another, rather than on the explicit geometric properties of task objects themselves. Although we apply the method in the context of PbD in this thesis, the concept is general in that it can be adopted to construct a configuration space representation wherever execution of a task by a skilled operator occurs; in teleoperation for example.

ii. An approach to deriving noise-free, low-level, robot control commands from demonstration. A contribution is made in two areas. First, a method for deriving noise-free position control commands is presented. Our method uses the partial knowledge of configuration space determined from (i), and knowledge of what regions in configuration space were visited in the demonstration. A position control command in the form of a path traversing in configuration space is derived. Our approach can derive paths that contain undemonstrated points and, as such, it can remove a greater range of sub-optimalities than [63, 90]. Compared to [79] our approach has the advantages that (a) it does not assume that all demonstrations pass between the same start and end points, (b) it can derive paths that lie outside the demonstration envelope, and (c) it can be applied to find paths in C-space of any dimension. However, we note that a limitation of our approach compared to others is that caution needs to be applied when tuning the parameters of the method to ensure that a valid path is produced. The method is more widely applicable than the problem of noise removal in PbD. It can be used to solve path planning problems in situations where (a) configuration space is only partially known, and (b) where some known obstacle-free points

in the space (eg. demonstrated points) exist. The second contribution of the thesis regarding low-level command synthesis is in the area of force control. A method is presented for deriving noise-free force control commands from demonstration. The method uses a partial knowledge of configuration space to determine a noise-free direction for force control. Smoothing and spline-fitting techniques are applied to determine a noise-free magnitude for the force control command.

iii. A method for choosing the demonstrated, task-level strategy most appropriate for the robot in its execution of the task. The human will generally demonstrate a number of distinct task-level strategies. Our method selects the task level strategy predicted to result in the most optimal robot performance of the task, ie. with respect to a set of chosen metrics.

1.5 Outline of the Thesis

Chapter 2 prepares a foundation for following chapters in the thesis. We present the spindle-assembly task as the task chosen in this thesis for experiments. Approaches to modeling both the task, and the skill in the task, are presented. In addition, we introduce a set of human-provided demonstrations of the spindle-assembly task that are used for experimentally validating work in following chapters.

Chapter 3 presents a method for constructing a partial representation of configuration space from demonstration. First, a method for determining a set of motion constraints in a task is presented. Next, regression analysis is applied to find an equation for the surface in C-space defined by each motion constraint. We apply the method to construct a partial configuration space representation for the spindle-assembly task.

Chapter 4 focuses on deriving a set of noise-free, low-level control commands for the robot. Two methods are presented: one for deriving a position control command,

and the other a force control command. Both methods use as their basis the partial knowledge of C-space derived in Chapter 3.

Chapter 5 addresses the problem of selecting a task-level strategy from demonstration. A set of metrics are presented for predicting robot performance of a task. Strategies predicted to see the robot best perform the task are selected. Experiments are presented to show the validity of the approach.

Chapter 6 brings the conclusion of the thesis. We state our conclusions on the work presented, and discuss open problems and areas requiring further research.

Chapter 2

Programming by Demonstration for a Spindle-Assembly Task

2.1 Introduction

In this chapter we prepare a foundation for the work that follows in this thesis. Our aims here are fourfold. First, we introduce the spindle-assembly task that will be used in experiments. This task provides a basis for the experimental testing and validation of our methods in following chapters. Second, we formally introduce Hybrid Dynamic Systems as a means for modeling assembly-task skill. We briefly introduced the concept of Hybrid Dynamic System modeling in the literature review of the previous chapter. In this chapter, we expand on this introduction to provide a detailed explanation on how a Hybrid Dynamic System is an appropriate skill model for assembly. Third, configuration space is a concept that will be used extensively in Chapters 3 and 4 of this thesis. We provide for the reader in this chapter a brief outline of the properties of configuration space that are central to our work in these chapters. We also explore the relationship existing between Configuration Space and our Hybrid Dynamic System modeling approach of assembly skill. Fourth and finally, we present a set of human-provided demonstrations for the spindle-assembly task. These demonstrations contain sub-optimalities, and provide

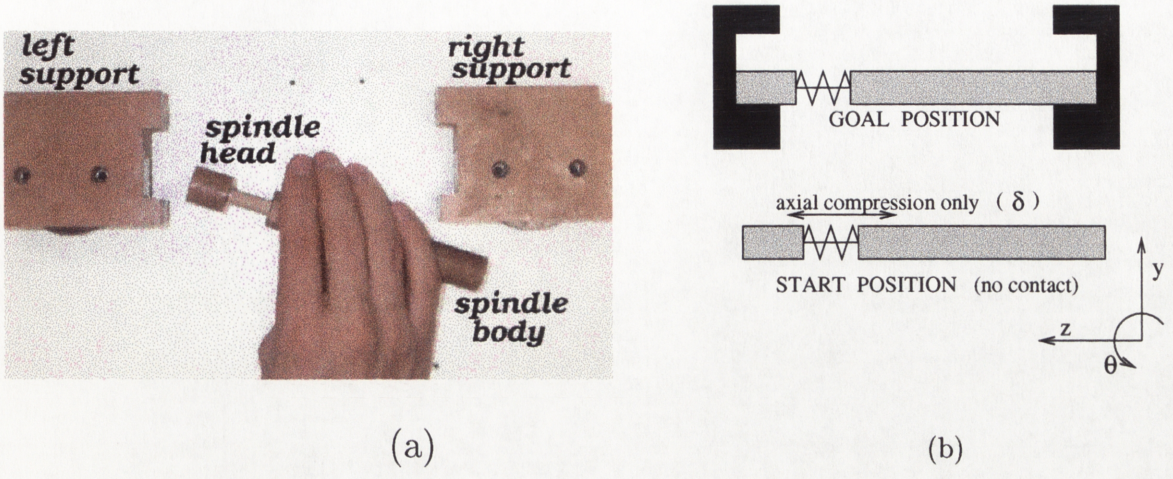


Figure 2.1: The spindle-assembly task chosen for PbD

the basis for our work in following chapters. That is, methods in following chapters will be applied to this set of demonstrations to provide, by the end of the thesis, a set of efficient and noise-free control commands for the robot to use in its execution of the task.

The chapter is set out as follows. In Section 2.2 we introduce the spindle-assembly task used for experiments. In Section 2.3 we present Hybrid Dynamic Systems as a means for modeling the skill required for such a task. Section 2.4 introduces the concept of Configuration Space, while in Section 2.5 we present the set of human-provided demonstrations of the spindle-assembly task. Finally, we conclude this chapter with Section 2.6.

2.2 The Spindle-Assembly Task

The task chosen for PbD is shown in Figure 2.1(a). It involves inserting an axially compressible spindle between two fixed supports. Figure 2.1(b) shows the start and goal spindle configurations of the task. The start position sees the spindle removed from the supports so that no contact exists. The goal position sees the spindle lying between a rebate in each support. Note that we define the task to be completed only when both ends of the spindle (i) lie in these rebates, and (ii) are touching the inside front edges of

these rebates, ie. as shown in Figure 2.1(b).

The spindle-assembly task is in essence a planar task. That is, only motion in the horizontal plane is important for its completion. Four degrees of motion freedom exist in the task, the position and orientation of the spindle body relative to the supports (y, z and θ), and the compression of the spindle head relative to the spindle body (δ). Figure 2.1(b) confirms with labels y, z, θ and δ the physical significance of each degree of freedom in the task.

An important property of the task is that at least partial compression of the spindle must occur before insertion can take place. Such a property is desirable so that the ability to cope with spindle compression is a requirement for completing the task. However we note for the reader that this property means assembly cannot take place by inserting the spindle body into the right support first. That is, insertion must follow a sequence of spindle head insertion into the left support, followed by compression, and then by spindle body insertion into the right support.

In selecting this task for experiments, we had in mind that it should be a good example of typical assembly tasks found in service-type environments. The spindle-assembly task was deemed to be a good example for the following reasons:

- The task is based on the domestic chore of changing rolls on a paper roll holder, ie. it is a real task that can be found in many service robotic environments, eg. hospitals, hotels, the home, etc.

- The task is reasonably complex. It contains a reasonable number of degrees of freedom (ie. 4), including rotational degrees of freedom.

- Assembly is about dealing with constrained motion. The spindle-assembly task contains a good range of constrained motion scenarios, eg. from where spindle motion is highly constrained (ie. where the spindle is, or is close to, fully inserted), to less constrained motion (ie. where the spindle is away from the supports).

- Finally, successful completion of the spindle-assembly task requires deformation of the spindle. Success in many common service-environment tasks require the manipulated object to deformed in some way.

The task to be used in experiments has been introduced. We now describe in the next section how the skill for achieving this task can be modeled as a hybrid dynamic system.

2.3 Hybrid Dynamic System Modeling of Assembly Skill

We introduced in Chapter 1 the idea of using a Hybrid Dynamic System to model skill for assembly tasks. Brockett first introduced the notion of a Hybrid Dynamic System (HDS) as a continuous-time system interacting with a discrete-event system [102]. Since then, HDS's have been used as models in a number of applications [4, 86], including for modeling assembly skill in robotics [10, 72, 12]. As applied in modeling assembly skill, the continuous-time and discrete-event systems each provide a different description of assembly dynamics. The continuous-time system describes the dynamics of the manipulated object (in our case, the spindle) relative to an inertially fixed environment (the supports) as (i) a vector equation in state-space form of:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{2.1}$$

and (ii) a set of constraint equations, where each equation in the set is of the general form:

$$g_j(\mathbf{x}(t)) = \mathbf{0} \tag{2.2}$$

Equation (2.1) describes the free-space dynamics of the spindle, where $\mathbf{x}(t)$ is the continuous-time system state vector $[y, z, \theta, \delta]^T$, and $\mathbf{u}(t)$ is the control input vector. As the spindle makes contact with the supports, its motion becomes constrained. The constraint on motion is described by a set of constraint equations, where each constraint equation in the set will have the general form (2.2). Each constraint equation in the set represents the

loss of a single degree of motion freedom for the spindle, and will in general correspond to a single point contact between the spindle and supports. For example, Figure 2.2(a) shows a single point contact that has been labelled b-8. [1]. This contact causes the loss of one degree of motion freedom for the spindle, and will contribute one equation of the form (2.2) to the constraint equation set. If b-8 is the only contact existing between the spindle and supports, then the constraint equation set will consist of only a single equation. However, in general, more than a single point of contact will exist between the spindle and supports during the assembly process, and so more than a single constraint equation will exist in the constraint equation set. We reflect this fact by appending a subscript j to g in Equation (2.2). That is, (2.2) is an equation describing the j^{th} constraint existing on the spindle motion at some particular point in the assembly process.

In contrast, the discrete-event system describes assembly dynamics as a sequence of asynchronous *discrete events* that occur through time. We saw for the continuous-time system how a set of constraints on spindle motion will exist during the assembly process. Then a discrete event in the discrete-event system is defined to occur when a change in this set of constraints occurs. For example, Figure 2.2(a) showed a situation in the assembly where a single constraint b-8 on spindle motion existed. Then a discrete event will occur either when (i) constraint b-8 is lost (ie. shown in Figure 2.2(b)), or (ii) if another constraint (eg. h-3) is gained (shown in Figure 2.2(c)). Discrete events trigger the discrete-event system to move between different *discrete states.* Each discrete state corresponds to a distinct set of constraints on spindle motion that can occur in the task. We saw how each constraint corresponds to a single point contact between the spindle and supports. Then, each discrete state will correspond to a unique set of contacts (ie. a unique *contact formation*) between the spindle and supports. For example, the spindle-support contact formation shown in Figure 2.2(a) will correspond to one discrete state, while the contact formations in Figure 2.2 (b) and (c) will correspond to two other, distinct discrete

[1] Note how a single point contact between the spindle and supports can be referenced as a letter-number pair when vertices are labelled with letters, and edges with numbers.

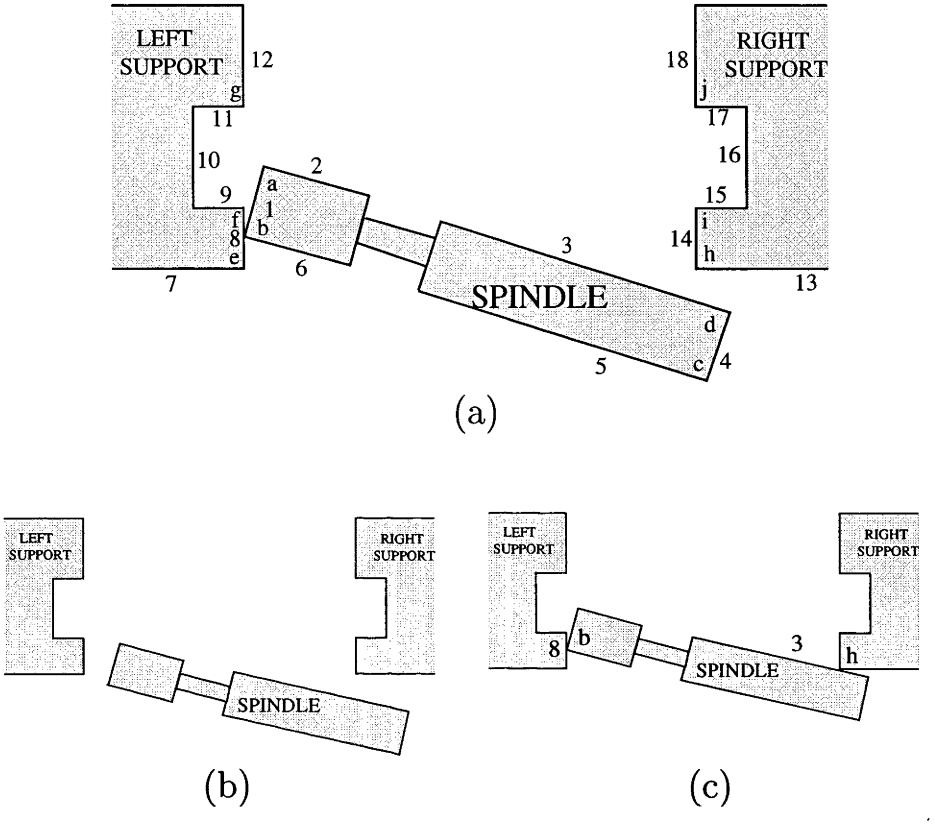


Figure 2.2: Discrete states defined on the basis of motion constraints applied by the supports on the spindle

states. These definitions of discrete events and states, allow the discrete-event system to provide a "high-level" description of assembly dynamics. That is, the discrete-event system describes assembly as a sequence of events that see the assembly pass from an initial unassembled state (eg. as shown in Figure 2.2(b)), through a sequence of partially assembled states (eg. states shown in Figures 2.2(a) or 2.2(c), to a final fully-assembled state.

We have seen how the discrete-event and continuous-time systems each provide a different description of the assembly dynamics. Then the HDS models the skill of assembly as two control processes operating in tandem, one in each of these systems. Control in the discrete system provides an abstract, task-level control regime. It has the purpose of

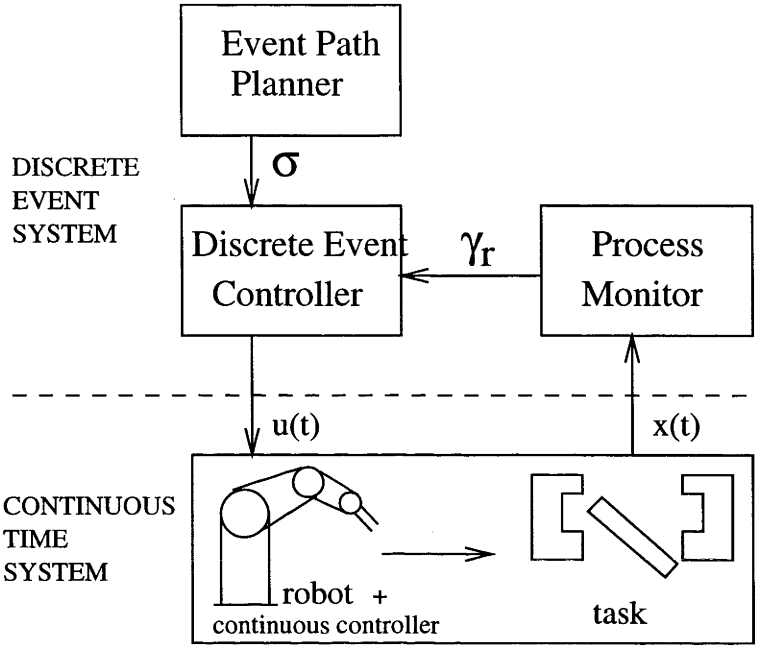


Figure 2.3: The Hybrid Dynamic System skill model for assembly tasks

pushing the assembly through the required sequence of discrete states so that the final, fully-assembled state is reached. Control in the continuous system is more focussed on controlling the assembly process within each state. It has the purpose of modifying the configuration of the spindle (ie. $\mathbf{x}(t)$) within each state so that the next desired state in the sequence is reached. Research in manipulation suggests that humans use a hierarchical structure for movement production when they execute assembly tasks. A generic plan is formulated at an abstract, task level, and implemented at a more concrete, continuous level [80]. This research suggests that a HDS is an appropriate modeling approach for assembly skill. The discrete-event system represents the abstract, task-level part of the human thought structure, while the continuous-time system forms the lower, more concrete, continuous level.

A general description of how a HDS models assembly skill has been provided. We will now describe the details of this skill model. Figure 2.3 shows schematically the essential elements of a HDS that models assembly skill. The figure divides the HDS with a dashed

line into two parts: the discrete-event system and the continuous-time system. In the discrete-event system we have a Discrete Event Controller (DEC), Process Monitor (PM) and Event Path Planner (EPP). The role of the PM is to determine the state of the discrete-event system from the continuous-time system state vector $\mathbf{x}(t)$. We denote in this thesis the state of the discrete-event system generally as γ. We add a subscript r to γ in Figure 2.3 to indicate that the current state of the discrete-event system is the r^{th} state visited in the assembly process so far. The DEC receives the current state γ_r from the PM and must decide on a command $\mathbf{u}(t)$ for output to the continuous controller. Command $\mathbf{u}(t)$ has the purpose of moving the assembly process closer to completion. That is, it has the purpose of causing an event to occur that will move the discrete-event system one state closer to the final, fully-assembled state. A sequence of events that will reach the fully-assembled state is determined by the Event Path Planner. We call this sequence the *desired event path*, and denote it as σ. In the continuous-time system, the Continuous Controller (CC) receives the command $\mathbf{u}(t)$ from the DEC. It applies this command via the robot to modify the configuration $\mathbf{x}(t)$ of the task. The PM recognises from $\mathbf{x}(t)$ when a change in the current state γ_r occurs. The new current state (ie. γ_{r+1}) is output to the DEC, and the loop continues until the discrete-event system reaches the fully-assembled state.

Note from our discussion of HDS operation how each component in the HDS encodes a sub-skill required for assembly. The PM encodes the skill of recognising a new state (ie. a new contact formation) in the task. The EPP encodes the skill of deciding on an appropriate sequence of contact formations to traverse in order to reach the fully-assembled contact formation. The DEC encodes the skill of selecting a control command that will see the next contact formation in the desired sequence achieved. Finally, the CC/manipulator encodes the skill of manipulating the workpiece within each contact formation. PbD using HDS means capturing the skill of each of these modules from the demonstration. Our focus in this thesis is on removing noise from the demonstration so that efficient, noise-free robot commands can be determined. That is, we focus in this thesis on encoding from

demonstration, the skills in the DEC and EPP modules in the HDS, looking specifically at how commands produced by these modules can be made noise-free. However, a desire of this thesis must also be to show that a PbD system using HDS skill modeling is a viable approach to programming in service robotic environments. That is, two other modules in the HDS exist: the PM and the CC. How should the functionality in these components be realised?

Monitoring of contact formations has received significant attention in the literature to date. Most work assumes the presence of detailed geometric knowledge of the task, and so is not suitable for PbD in a service robotics environment [109, 110]. Recall from our literature review of the previous chapter work by Skubic and Volz [114], and McCarragher et al. [36, 85]. Work by these authors focussed on deriving a functional PM from demonstration. Skubic and Volz used demonstration and fuzzy logic to train-up a functional PM. McCarragher et. al. presented similar approaches using a multilayer perceptron neural network, and a method combining linear discriminant functions and clustering techniques. Since the methods used by these authors are demonstration based, each would be suitable for deriving a functional PM in a service robotic environment. That is, none of these methods require a-priori geometric knowledge of the task to be programmed. For the remainder of this thesis we assume that a functional PM has been derived using one of these methods.

We have made the assumption that a functional PM exists. Another assumption we make for work that follows is the presence of a functional CC and manipulator. An obvious and valid assumption for PbD is the presence of a robot manipulator. The CC is also a component of the HDS whose functionality must also be realised. However, unlike the PM, the functionality of the CC need not be re-realised for each new task to be programmed. The PM is required to recognise the different discrete states in a task, and different tasks will have a different set of discrete states. As such, it must be retrained for each new task to be programmed. In contrast, the CC for assembly tasks encodes a skill common across different tasks: manipulating a workpiece within a specific contact formation. Raibert and Craig introduced a continuous controller regime well suited to for-filling this functionality.

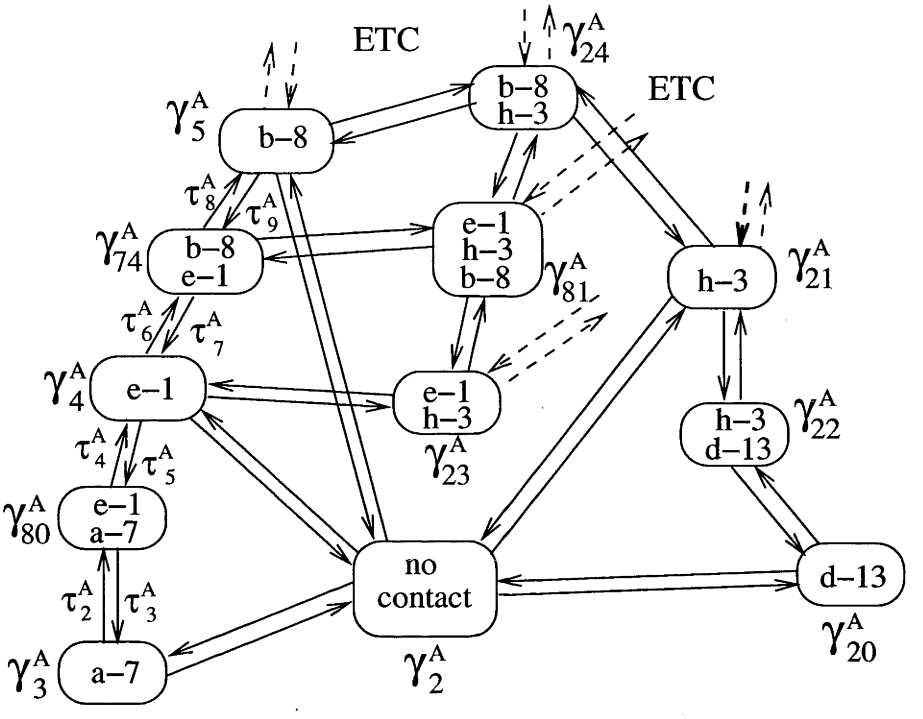


Figure 2.4: Part of the automaton of discrete states for the spindle-assembly task

They introduced the Hybrid force-position control regime [73], where in this case the term *hybrid* refers to a combination of force and position control [2]. We shall talk more later about the details of hybrid force-position control when we present work in Chapter 4. For now, we wish only to state two things: that work in this thesis will assume (i) the existence of a functional continuous controller component of the HDS prior to any demonstration of the task, and (ii) that the continuous controller existing is a hybrid force-position controller.

We have seen how a hybrid dynamic system models skill in assembly and how our aim in this thesis is to derive functional DEC and EPP components of the HDS using PbD. Before moving on, we finalise this section by introducing a representation of the discrete-event system used extensively in this thesis. The discrete-event system can be

[2]This is in contrast to our previous use of the term hybrid in *hybrid dynamic system*, which refers to a combination of discrete-event and continuous-time systems.

represented as an automaton, where each node in the automaton corresponds to a state in the discrete-event system, and each arc represents an event in the system. Denote as $\mathcal{A}$ the automaton for the spindle-assembly task. We show in Figure 2.4 a small part of the automaton $\mathcal{A}$. Each node in Figure 2.4 represents a distinct state of the discrete-event system for the spindle assembly task. We use the notation $\gamma_i^{\mathcal{A}}$ to refer to the i^{th} distinct state in the task. For example, we have labelled the node in the bottom left hand corner of the figure as $\gamma_3^{\mathcal{A}}$, ie. this node represents the 3^{rd} distinct state in the task [3]. Figure 2.4 also shows the nodes in $\mathcal{A}$ that represent states $\gamma_{80}^{\mathcal{A}}$, $\gamma_4^{\mathcal{A}}$, $\gamma_{74}^{\mathcal{A}}$, etc. Note the difference between our notation $\gamma_i^{\mathcal{A}}$ here, and the notation γ_r introduced earlier in this section. That is, $\gamma_i^{\mathcal{A}}$ denotes the i^{th} state in the task, while γ_r denoted the r^{th} state visited in a particular assembly sequence of the task. For the remainder of this thesis, when we refer to a state in the task simply as *state X*, we will be referring to state $\gamma_X^{\mathcal{A}}$.

To provide the reader with some idea of what states in the task are represented by the part of $\mathcal{A}$ shown in Figure 2.4, we include for each state shown a constraint set. For example, the figure shows the constraint set for state $\gamma_3^{\mathcal{A}}$ as $\{a-7\}$, and for state $\gamma_{80}^{\mathcal{A}}$ as $\{a-7, e-1\}$, etc. Recall how constraints in these constraint sets can be decoded into single point contacts between the spindle and supports using Figure 2.2(a)). Then, it becomes clear that the part of $\mathcal{A}$ shown in Figure 2.4 corresponds to contact formations around where the spindle head and body are in contact with the front edges of each support.

Our discussion so far has focussed on the nodes in $\mathcal{A}$. Figure 2.4 also shows a subset of the arcs in $\mathcal{A}$. Each arc represents an event that must occur for the discrete-event system to move between two states in the task. Denote as $\tau_k^{\mathcal{A}}$ the k^{th} possible event in the task. Note then how we have labelled arcs in Figure 2.4 with the events they represent [4]. For example, two arcs exist in the figure between the nodes representing states $\gamma_3^{\mathcal{A}}$ and $\gamma_{80}^{\mathcal{A}}$. One represents the event causing the discrete-event system to move from state $\gamma_3^{\mathcal{A}}$ into state $\gamma_{80}^{\mathcal{A}}$, and has been labelled $\tau_2^{\mathcal{A}}$. The other represents the event causing the system

[3] Note that the numbering of states in the task was completed in an arbitrary fashion, ie. the index i assigned to each state in the task was chosen with no particular order.

[4] For reasons of clarity we only label arcs on the left-hand side of the figure.

to move from state $\gamma_{80}^{\mathcal{A}}$ into $\gamma_{3}^{\mathcal{A}}$, and has been labelled $\tau_{3}^{\mathcal{A}}$. Note how no relationship in $\mathcal{A}$ exists between the value of the index k assigned to a particular event, and the values of the index i assigned to the states connected by that event. Often we will need to refer to a particular event according to the states between which it passes. For this purpose we introduce the additional notation for an event as an ordered pair. The first element in the pair indicates the value of index i for the state existing prior to the event occurring. The second element in the pair indicates the value of index i for the state existing after the event occurs. For example, with this notation we can refer to event $\tau_{2}^{\mathcal{A}}$ as the pair $(3,80)^{\mathcal{A}}$.

2.4 The Configuration Space Representation for Assembly

The previous section focussed on modeling the skill required in an assembly task. In this section we present *Configuration Space* (C-space) as a means for modeling the task itself. That is, we use configuration space as a method for describing in the spindle assembly task the spatial configuration of the spindle relative to the supports. The idea of C-space is applicable to describing the configuration of any object with freedom of motion in space. It was first introduced by Lozano-Perez [127] in the early 1980's. In this thesis, we use the idea of C-space extensively in Chapters 3 and 4. Our aim in this section is to introduce some of the basic concepts and properties of C-space that are central to our work in these chapters [5]. In addition, we will also highlight in this section how our HDS model of assembly skill can be interpreted in the C-space representation of a task.

The idea introduced by Lozano-Perez was to represent the configuration of an object with n degrees of freedom in a physical workspace, as a single point in an n-dimensional *configuration space.* In this thesis we are interested in modeling an assembly task. Assembly tasks generally involve a manipulated object (the spindle), and one or more surrounding objects that are inertially fixed in the environment (the supports). The idea of C-space

[5] For an authoritative presentation of Configuration Space, see [65].

in assembly is to represent the configuration of the manipulated object as a point in a configuration space. For example, the spindle's configuration can be represented as a point in a 4-D configuration space (4-D because the spindle has 4 dof). Then each fixed object in the task forms a region in C-space that defines an obstacle (called a C-obstacle). These regions define obstacles because they correspond to configurations where the manipulated object violates (ie. passes into) one or more of the fixed objects. Together, all C-obstacles form a region in C-space. Denote this region as C_{obs}. Two other regions besides C_{obs} exist in C-space. The first is C_{free}. C_{free} contains all points in C-space that lie outside C_{obs}. In the physical workspace, C_{free} corresponds to situations where the manipulated object is not in contact with any fixed objects. The other region in C-space besides C_{obs} and C_{free} is C_{con}. C_{con} contains all points in C-space lying exactly on the boundary between C_{free} and C_{obs}. It corresponds in the physical workspace to the manipulated object being exactly in contact with one or more fixed objects. C_{con} defines a surface (or hyper-surface) in C-space consisting of interconnected patches of curved surface (or hyper-surface). We denote each patch of surface making up C_{con} as a *C-surface*. We saw in Section 2.3 how a single point contact b-8 between the spindle and supports was described by single constraint equation of form (2.2). Then we note that the constraint equation for b-8 will describe one of the C-surfaces making-up C_{con}. C-surfaces have dimension ranging from zero to one less than the dimension of C-space. Contact b-8 defines the loss of one spindle dof, and so its equation defines a C-surface of dimension 3 in the 4 dimensional spindle-assembly-task C-space. C-surfaces of lesser dimension correspond to spindle-support contacts containing more than a single point of contact. For example, the two-point contact formation shown in Figure 2.2(b) corresponds to a C-surface in C_{con} of dimension 2.

We have seen something of the topology of C-space. Then we note that the concept of an assembly in C-space is straight-forward. The start and goal configurations of the manipulated object in the task correspond to two, non-coincident points in C-space. Since the assembly process usually begins with task objects not in contact, the start point will lie somewhere in C_{free}. In contrast, task objects *will* be in contact when fully assembled,

ie. the end point will lie on one of the C-surfaces that makes up C_{con}. The sequence of configurations used in the assembly process form a path through C-space between these start and end points. This path will generally pass across a number of different C-surfaces before finally reaching the C-surface that contains the end point of the assembly.

It is interesting to note what the HDS assembly skill model means in C-space. We saw that the no-contact state in the HDS was defined for the spindle-assembly task as state 2 (see Figure 2.4). Then state 2 corresponds to C_{free}. That is, any spindle configuration in state 2 will correspond to a point in C_{free}. All other states in the HDS define contact between the spindle and supports, and so together correspond to C_{con}. Individually, each contact-defining state in the HDS corresponds to one of the *C-surfaces* in C-space. This correspondence between C-surfaces in C-space and states in the discrete-event system means the operation of our HDS can be interpreted nicely in C-space. That is, the EPP decides on a sequence of C-surfaces to be traversed in order to complete the assembly. The PM identifies when we reach each new C-surface. The DEC determines the control action to be taken on the current C-surface in order to reach the next C-surface in the desired sequence. Finally, the CC controls our position on the current C-surface so that the control action determined by the DEC is traversed.

2.5 Demonstrating the Spindle-Assembly Task

Previous sections have focussed on the general issue of modeling. In this section we take a first step toward specific work in later chapters by presenting a set of human provided demonstrations of the spindle-assembly task. The demonstration set presented here will provide the basis for testing the noise removal methods presented in later chapters.

Figure 2.5 shows the apparatus on which demonstrations of the spindle-assembly task were provided. It shows the spindle and supports, along with the sensors used to record the demonstrations. Two sensor types were used. A polhemus position sensor [84] was used to record the position and orientation of the spindle body relative to the supports (y,z,θ) and

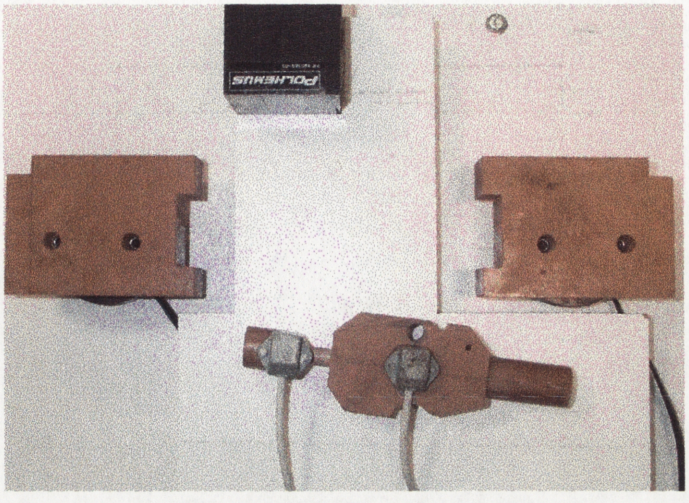


Figure 2.5: Apparatus used to capture human's demonstration of the spindle assembly task

the compression of the spindle head relative to the spindle body (δ). The Polhemus sensor determines the position/orientation of a body by transmitting electro-magnetic signal from a central transmitter to a body-attached receiver [6]. Figure 2.5 shows two receivers, one each attached to the spindle body and head, along with the main transmitter (black cube) at the top of the picture. The second mode of sensing used was force sensing. Two JR3 [60] force sensors were used to capture the forces and torques applied to each support. This information was used to calculate the force and torque applied by the demonstrator to the spindle itself. In Figure 2.5, the force sensors are obscured by the supports themselves, however the cables used to connect each force sensor back to a central computer are clearly visible. Two additional components of the demonstration apparatus are shown in Figure 2.5; first the *base-board* - the raised section of white board sitting between, and in front of, the supports; and second the *block* - the block of craftwood in which the spindle is mounted. We have noted previously that only motion in the horizontal plane is important for spindle-assembly task completion. The base-board and block components

[6] This signal can be affected by metallic objects that are in close proximity to the sensor, and was the reason for constructing the spindle-assembly task from *Craftwood*.

were introduced to make it easier for the human to demonstrate in the horizontal plane. The base-board provided a horizontal platform at the correct height on which the spindle could be moved, and the block promoted sliding (rather than rolling) of the spindle when it was moved. The final component in the demonstration apparatus was a *Wind River Systems* microcomputer running the real-time operating system VX-works [132]. This component was used to capture and process the data obtained from the polhemus and force sensors. All software used was specially written in-house for these experiments.

A total of six demonstrations of the task were provided by the human. These demonstrations saw the human move the spindle from the start position to the goal position using assembly sequences chosen by himself. A number of features of the demonstration process need to be explained. These are:

- **Number of demonstrations provided:** We saw in Chapter 1 that, as an implicit programming method, PbD benefits when a large number of demonstrations are provided. This is the case because more demonstrations usually mean a richer set of examples showing how to complete the task. However, there is a limit to the number of demonstrations a human can comfortably provide. Six demonstrations of the task were provided in our experiments. We chose the number six because we feel this is about the maximum number of demonstrations that users in a service robotic environment would be willing to provide.

- **Restricting human use of vision:** Human vision provides a powerful sensing medium for achieving assembly tasks. Vision allows the human to make decisions about the relative positions between objects in the assembly so that appropriate actions can be applied. In contrast, our robotic system does not have such information available. Position and force sensors were used to record the demonstration. Information from the position sensor provides only the absolute position of the spindle, rather than its position relative to surrounding objects. Force information can only be used to determine relative positioning of objects when contact between the

objects exists. Hence, if allowed to use vision, the human will select actions on the basis of information that is not available to the robot [7].

Two solutions to the dilemma exist. The first is to capture the demonstration using vision (ie. using cameras) and then replicate human vision sensing capabilities in the robotic system. However, replicating human vision sensing means determining the relative positioning of objects in the workspace from a video stream of the workspace scene, and is not at all straight-forward. The second possible solution is to restrict the humans use of vision by blindfolding him during the demonstration. This solution has the effect of making the human operate with the same perception information as the robot, ie. He must use reaction forces (sensed through the fingertips), and absolute position of the spindle (estimated based on arm pose) as the means by which appropriate actions in the task are chosen. Blindfolding the human was the solution we took for our experiments here.

- **Previous task experience:** Some authors in PbD promote the use of a task *expert* to provide the demonstrations [117]. For example, in a factory environment, shop-floor personnel may have analysed tasks in the manufacturing process in great detail. Such experts may know exactly how the robot should perform a task, and demonstrate accordingly. In contrast, end-users in service-robotic environments would have rarely analysed in such detail the task they wish to program. To reflect this fact in our experiments, a person with limited experience of the task was chosen as the demonstrator. Initially the person had no experience of the task. Prior to the demonstration session, this person was allowed to become familiar with the task by handling it, and by practicing a number trial executions of the task. In this way the demonstrator was allowed to become familiar with the task without being an expert.

[7] Ensuring that information used by the human to make control decisions is also available to the robot is a well known requirement in PbD. For example, Asada and Liu explore this issue in detail for PbD of a grinding task in [116].

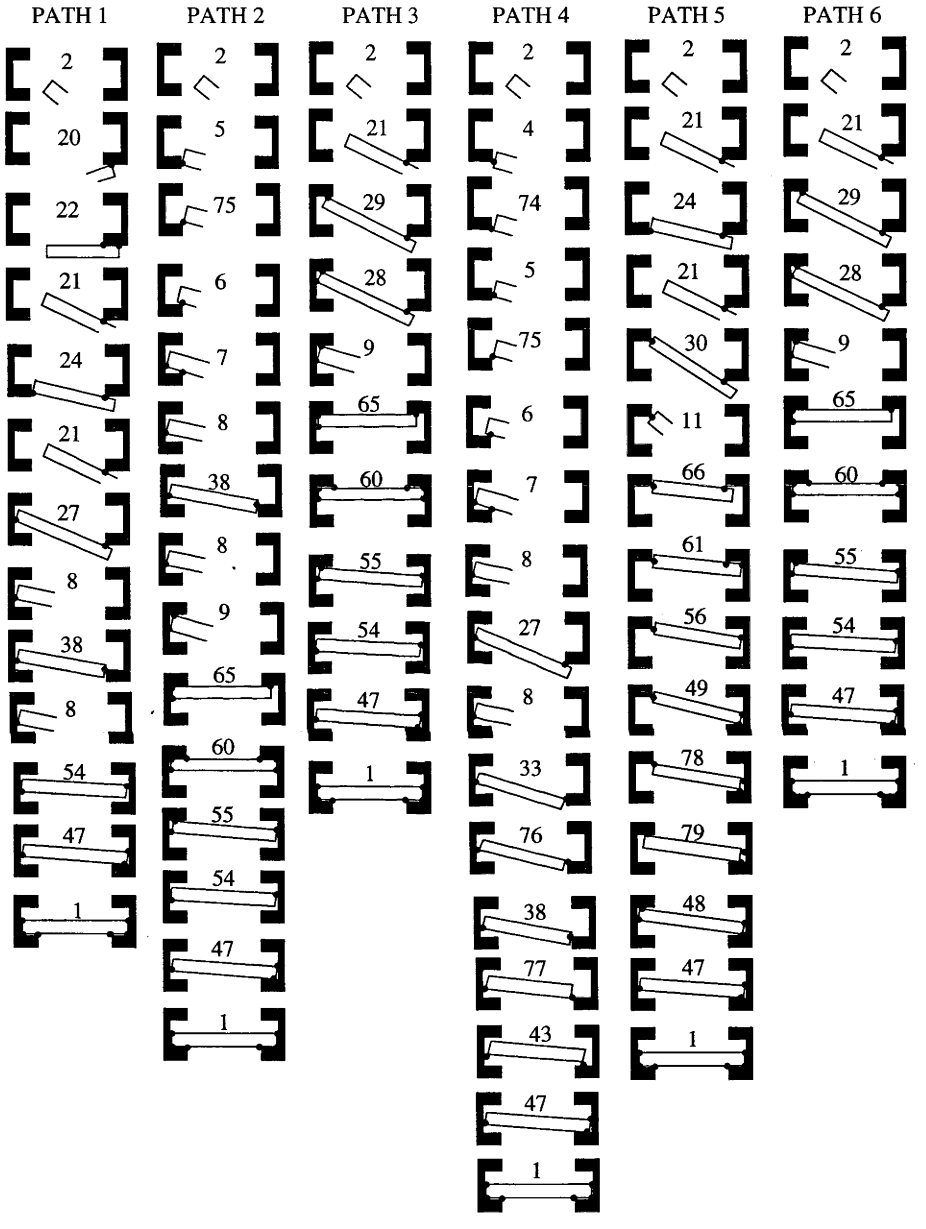


Figure 2.6: The set of state sequences demonstrated by the human in the spindle-assembly task

The six demonstrations provided by our demonstrator are shown in Figure 2.6. Each demonstration is presented as a sequence of discrete states. Each state in the sequence is labelled with a number, and is shown diagrammatically as a contact formation between the spindle and supports. The number for a state indicates the relevant value of the index i for that state, ie. the state labelled with "2" refers to state γ_2^A, etc. Notice how every demonstration commences in the no-contact state (ie. in the start state for the task - state 2) and ends in the goal state (state 1). For the remainder of this thesis, we will refer to the demonstration sequences shown in Figure 2.6 from left to right as D_1 to D_6.

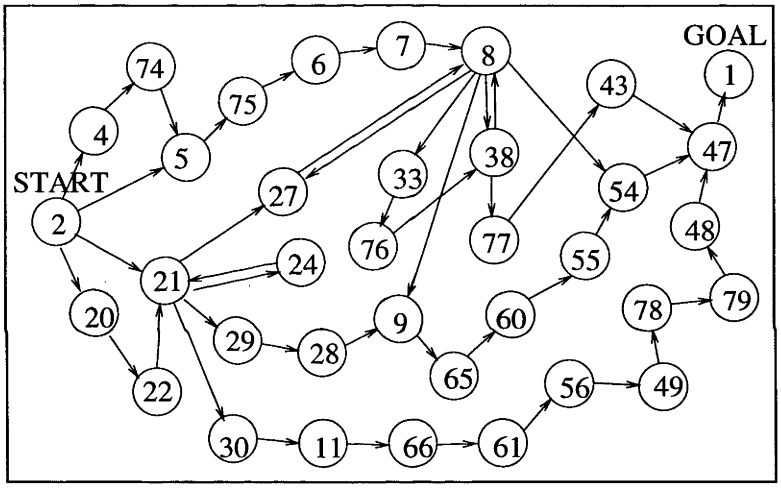


Figure 2.7: The discrete state automaton $\mathcal{A}_D$, constructed as that part of $\mathcal{A}$ visited in the demonstration set D_1 to D_6

Recall how we introduced in Section 2.3 the concept of an automaton $\mathcal{A}$ to represent the discrete-event system in the HDS. We present in Figure 2.7, that part of $\mathcal{A}$ visited in the demonstration set. That is, we show in Figure 2.7 the nodes of states, and the arcs of events, visited in the demonstration. Note that the arcs and nodes in Figure 2.7 themselves form an automaton. We will refer in future to this automaton as $\mathcal{A}_D$. In addition, note that state 2 in the task is a special state because it is the only state that does not define contact between the spindle and supports. We will find it useful in later chapters to refer to the automaton of states in the task that define contact. That is,

we shall refer to the automaton in Figure 2.7 less state 2, as $\mathcal{A}_D^*$. $\mathcal{A}_D^*$ then forms an automaton of contact-defining states that were visited in the demonstration.

2.6 Conclusion

The purpose of this chapter was to provide a foundation for work that follows in this thesis. Four sections were presented. First, we introduced the spindle-assembly task to be programmed using PbD. Next, we showed how the skill in achieving the task could be modeled as a Hybrid Dynamic System. Third, we presented the concept of Configuration Space as a means for modeling the task itself. Fourth and finally, a set of human-generated demonstrations for the spindle-assembly task were introduced. With this chapter complete, we are now in a position to present work in the thesis that is new to the literature.

Chapter 3

Configuration Space Derivation from Demonstration

3.1 Introduction

Access to task-specific, geometric information is an important component of the noise removal process in PbD. Such information allows intelligent decisions to be made about whether demonstrated actions contain noise. While geometric information about the task is desirable, it is not often available a-priori in the informal environments typical of service robotics. In this chapter our aim is to obtain geometric information about a task from the demonstration phase of PbD. We show how a partial C-space representation of the task is obtainable from demonstration. C-space is a representation of task geometry that focuses on motion constraints between task objects, rather than on the geometric properties of objects themselves. We note that a *partial* representation is obtained because our method derives only that part of C-space visited in the demonstration.

The remainder of this chapter is set out as follows. In Section 3.2 we formulate exactly the problem to be solved. We break the main problem down into two sub-problems: (i) for each state in the demonstration set, finding the constraints on spindle motion that exist, and (ii) finding an equation for the C-surface defined by each constraint found in (i). In

Section 3.3, the assumptions on which our work is based are presented. Specifically, we make three assumptions about information returned by the Process Monitor component of our PbD system. Section 3.4 presents a solution to sub-problem (i). We present a method that derives the constraint set for each state visited in the demonstration. Section 3.5 presents a solution to sub-problem (ii). Statistical regression analysis is used to determine a C-surface equation for the set of unique constraints identified in Section 3.4. Finally, we end the chapter with Section 3.6, where we state our conclusions for the work.

3.2 Problem Formulation

Our interest is in deriving a partial representation of C-space for the spindle-insertion task. We introduced in the previous chapter an automaton $\mathcal{A}_D^*$: the automaton of contact-defining states visited in the demonstration. Recall from Chapter 2 that γ_i^A denoted the i^{th} distinct state in task. Assume for our work in this chapter that γ_i^A was a demonstrated state that defines contact between the spindle and supports, ie. that a node for γ_i^A exists in $\mathcal{A}_D^*$. We saw in Chapter 2 that each contact-defining state in the HDS defines a C-surface in C-space. Let c_i^A be the C-surface of γ_i^A. Then our problem in this chapter can be stated as: find an algebraic equation that describes each c_i^A in C-space. Denote this equation as the *C-surface equation* for γ_i^A.

Let the dimension of C-space be n. Then the problem of deriving a C-surface equation for γ_i^A is a complex one, because c_i^A can be a surface ranging in dimension from zero to "$n-1$". However, the problem is simplified if we note that any c_i^A can be specified as the intersection of C-surfaces of dimension n–1. Denote any C-surface in C-space of dimension $n-1$ as a *primitive C-surface.* Recall that a state is defined by a unique set of constraints (remember, for example, how Figure 2.4 showed the constraint sets for a number of states in the spindle-assembly task, including the constraint set for state 80 as $\{a-7, e-1\}$). Let Ω_i^A be the constraint set defining γ_i^A. That is, $\Omega_i^A = (\rho_{i1}^A, \ldots, \rho_{ij}^A, \ldots, \rho_{in_j}^A)$, where ρ_{ij}^A is the j^{th} constraint in state γ_i^A, and n_j is the number of constraints in γ_i^A. Each constraint

ρ_{ij}^A in Ω_i^A, in general, results in the loss of one spindle dof [1], and hence defines a primitive C-surface in C-space. Let c_{ij}^A be the primitive C-surface defined by ρ_{ij}^A. Then we identify that c_i^A can be specified as:

$$c_i^A = \bigcap_{j=1}^{n_j} c_{ij}^A \tag{3.1}$$

That is, to fully define the surface c_i^A, our methods in this chapter need to:

(a) Find the constraint set Ω_i^A for each state γ_i^A visited in the demonstration, and

(b) Once a constraint set for γ_i^A is found, derive an equation for the primitive C-surface defined by each constraint in that set.

Then, by applying equation 3.1, c_i^A can be determined, and the region of C-space visited in the demonstration will have been derived. The remainder of this chapter is devoted to presenting solutions to problems (a) and (b). However, prior to presenting a solution for problem (a) in Section 3.4, and for problem (b) in Section 3.5, we first introduce in the next section some assumptions made in this work about information returned by the Process Monitor component of our PbD system.

3.3 Process Monitor Assumptions

We saw in the previous chapter that a functional PM can be obtained from a pre-demonstration training phase according to work in [114], [85], or [36]. We assume for our work in this chapter that such a PM exists. That is, during each demonstration, we have a functional PM that will return at each discrete state visited in the demonstration, a state number that uniquely identifies the state from all others in the task. In addition, we assume the PM also returns two other pieces of information at each new state: whether

[1] that is, when each constraint in the set cannot be described as a combination of the others

a constraint was gained or lost, and the *type* of that constraint. That is, at each new state we assume the PM returns the triple:

$[\gamma_i^A, gl, ct]^T$

where

- $\gamma_i^A \in \mathbb{N}$ denotes the state number of the new state.
- $gl \in \{-1, 1\}$ denotes an increase or decrease by one in the number of constraints present in the new state compared to the previous state. $gl = 1$ denotes a constraint gain, while $gl = -1$ denotes a constraint loss.
- ct denotes the *type* of the constraint that was gained or lost. Lozano-Perez [127] showed for a planar task involving two objects that two types of constraint exist: (i) constraints resulting from a contact between and edge of the first body with a vertex of the second, and (ii) constraints resulting from a vertex of the first body with an edge of the second. Because the spindle-insertion task involves two pairs of objects in contact (ie. spindle head with left support, and spindle body with right support) four contact types exist for this task [2]. These are:

  1. spindle head edge in contact with vertex of left support (hev)
  2. spindle head vertex in contact with edge of left support (hve)
  3. spindle body edge in contact with vertex of right support (bev)
  4. spindle body vertex in contact with edge of right support (bve)

  Note how we introduce four symbols: hev, hve, bev and bve, to reference each of these constraint types. For now, we are content only to introduce the concept of four constraint types for the spindle-assembly task. Later, in Section 3.5.1 we will

[2]Note that we do not consider constraints resulting from head/right-support or body/left-support contacts, since, as stated in Chapter 2, we assumed a method of task completion where these contacts do not occur.

explore in more detail how these constraint types come about, and why we separate constraints into these four distinct types.

A number of significant questions exist regarding our assumptions in this section. One relates to whether the PM could return information encoded in *gl* and *ct*. At this stage of the work, we have not explored in detail the possible methods for determining *gl* or *ct*. However, we have some ideas as to how this information could be derived. Take *gl* for example. We have seen how there is a well defined relationship between contact gain and loss between two objects, and the gain or loss of constraint caused by one object on the motion of the other. Gains or losses in contact may be identifiable using reasoning on contact forces and spindle position/velocity. An alternate approach may be to use training coupled with "learning" methods (eg. neural networks), as used in [114] and [43] to recognise discrete states. For this thesis, we make the assumption that *gl* and *ct* are returned by the PM. We leave the details of possible methods to achieve this information as future work.

Another question regarding our assumptions in this section relate to whether a PM could be trained from demonstration to perfectly process monitor a task. For example, Hovland [36] trained a process monitor from demonstration that achieved a 95 percent recognition success rate. For our work in this thesis, we have assumed a process monitor with a 100 percent recognition success rate. It is conceivable that our methods to follow could be augmented to cope with an imperfect process monitor, however, at this stage of the research, we leave such an augmentation as future work.

3.4 Finding a Constraint Set for each Demonstrated State

We have stated our assumptions about information available from the PM. We are now in a position to present a solution for the first of the two problems identified in Section 3.2: finding a constraint set Ω_i^A for each state γ_i^A that was demonstrated. We note that Ω_i^A refers to the true set of constraints that exist in γ_i^A. The methods we now present provide

an *estimate* of the true constraint set. We refer to the estimate of Ω_i^A produced by our methods as $\hat{\Omega}_i^A$. Recall how we denoted the j^{th} constraint in Ω_i^A as ρ_{ij}^A. Then we denote as $\hat{\rho}_{ij}^A$ the estimate of ρ_{ij}^A produced by the methods we present.

We achieve $\hat{\Omega}_i^A$ by using information returned by the PM during the demonstration phase of PbD. In Chapter 2 we introduced a set of six demonstrations D_1 to D_6 of the task. Let D_p denote generally one of the demonstrations in the set. We saw that a D_p consists of a sequence of states. Our presentation of HDS modeling in Chapter 2 showed how the r^{th} state in an assembly sequence was denoted as γ_r. Then we denote the states in D_p as $\gamma_1^{D_p} \ldots \gamma_r^{D_p} \ldots \gamma_{r_n}^{D_p}$, where $\gamma_r^{D_p}$ is the r^{th} state in D_p, and where r_n is the number of states in the sequence. Note then the correspondence between γ_i^A and a $\gamma_r^{D_p}$ in D_p. State γ_i^A can occur in many places in the demonstration set. For example, γ_5^A was shown in Figure 2.4 as the 5th distinct state in the task, but it was also shown in Figure 2.6 as the 2nd state visited in demonstration D_2, and as the 4th state visited in D_4, ie. for demonstration set D_1 to D_6, state γ_5^A corresponds to states $\gamma_2^{D_2}$ and $\gamma_4^{D_4}$. Our interest here is in deriving a constraint set for each γ_i^A visited in the demonstration. However, we introduce the notation $\gamma_r^{D_p}$ because our methods below require a distinction to be made between the same γ_i^A occurring at different stages in the demonstration set. For the same reason, we introduce additional notation for $\hat{\Omega}_i^A$ and $\hat{\rho}_{ij}^A$. That is, we denote $\hat{\Omega}_r^{D_p}$ as our constraint set estimate for the r^{th} state in D_p, and $\hat{\rho}_{rs}^{D_p}$ as the s^{th} constraint in $\hat{\Omega}_r^{D_p}$.

Three steps are involved in determining $\hat{\Omega}_i^A$ for each γ_i^A that was demonstrated. They are:

- Generation
- Numbering
- Merging

Sections 3.4.1, 3.4.2, and 3.4.3 describe the details of each step.

3.4.1 Generation

The role of the Generation Phase is to construct an initial version of $\hat{\Omega}_r^{D_p}$ that contains only the type of each constraint in $\gamma_r^{D_p}$. That is, after the generation phase, each $\hat{\rho}_{rs}^{D_p}$ in $\hat{\Omega}_r^{D_p}$ references a symbol (ie. one of bev, bve, hev, or hve) that describes the type of the true constraint $\rho_{rs}^{D_p}$. In practice, the Generation Phase operates as follows. As a demonstration proceeds, each new state $\gamma_r^{D_p}$ in the demonstration produces from the PM the triple $[\gamma_i^A, gl, ct]^T$. Then we can construct, at each state change, an initial version of $\hat{\Omega}_r^{D_p}$ according to:

A. if a constraint was gained between $\gamma_{r-1}^{D_p}$ and $\gamma_r^{D_p}$ (ie. $gl = +1$) then

1. set $\hat{\Omega}_r^{D_p}$ equal to $\hat{\Omega}_{r-1}^{D_p}$
2. append to $\hat{\Omega}_r^{D_p}$ the constraint type ct of the new constraint

B. if a constraint was lost between $\gamma_{r-1}^{D_p}$ and $\gamma_r^{D_p}$ (ie. gl = -1) then

1. set $\hat{\Omega}_r^{D_p}$ equal to $\hat{\Omega}_{r-1}^{D_p}$
2. remove from $\hat{\Omega}_r^{D_p}$ the constraint type ct of the lost constraint

C. For the case of the first state in each demonstration (ie. $\gamma_1^{D_p}$) , we set $\hat{\Omega}_1^{D_p}$ to be empty since we always start in the no contact state, ie. where no constraints on spindle motion exist.

To help clarify our presentation of the generation, numbering and merging phases, we introduce in Table 3.1 an example of the process. Part (a) of the table shows the sequence of constraint sets that were visited in D_1, where each constraint is referenced as a letter-number pair (recall how Figure 2.2(a) decodes these letter-number pairs into the spindle-support contact formations that cause each constraint). Part (b) of Table 3.1 shows the triple returned by the PM at each state change. Notice how the triple contains, a state number ($\gamma_r^{D_p}$), whether a constraint was gained or lost compared to the previous state (gl),

and the type of the constraint gained or lost (*ct*). Part (c) of the table shows an example of how the generation phase operated on D_1. As per condition **C.** of our generation phase algorithm, the constraint set estimate of the first state in D_1 (ie. state 2) is set to be empty (signified by *emp.* in the table). From there, each new constraint set estimate is determined by adding or removing from the preceding constraint set estimate the *ct* returned by the PM. Note that, although not shown in the table, the generation phase was also applied to demonstrations D_2 to D_6. The result was an initial version of $\hat{\Omega}_r^{D_p}$ for each $\gamma_r^{D_p}$ in the demonstration set.

3.4.2 Numbering

After the generation phase, each constraint estimate $\hat{\rho}_{rs}^{D_p}$ refers to a constraint type, ie. one of the symbols bev, bve, hev, or hve. The role of the numbering phase is to make each $\hat{\rho}_{rs}^{D_p}$ refer to an actual constraint, rather than only to its type. We achieve this phase by appending an index to the symbol referenced by each $\hat{\rho}_{rs}^{D_p}$. Then, after numbering, each $\hat{\rho}_{rs}^{D_p}$ references a unique symbol-index combination that forms our first estimate of the true constraint $\rho_{rs}^{D_p}$.

The numbering phase cannot be achieved by simply appending distinct indices to the symbol of each $\hat{\rho}_{rs}^{D_p}$. Such a scheme would identify the constraints in every state as distinct. We know from the definition of a discrete state in the HDS that sequential states in an assembly sequence differ in their constraint sets by only one constraint. That is, sequential states will have *some* constraints in common. For example, Table 3.1-part(a) shows in D_1 how constraint h-3 exists in the sequentially demonstrated states of 22,21,24,21, and 27. Then a critical aspect of the numbering phase is to ensure that h-3 is identified as the same constraint in each of the states in this sequence, ie. it can be identified as the same constraint by appending to the bev symbol in each of these states the same index value.

We complete the numbering phase as follows. First, four counters nc_{bev}, nc_{bve}, nc_{hev}, and nc_{hve} are created to keep track of the indices we post-pend during this phase. These counters are initialized at the commencement of the numbering phase according to:

(a) Demonstration 1

$\Omega_r^{D_p}$	n.a.	d-13	d-13 h-3	h-3	h-3 b-8	h-3	h-3 b-10	b-10	b-10 i-4	b-10	b-10 d-16	b-10 d-16 c-15	b-10 d-16 c-15 f-6

(b) PM Returns:

$\gamma_r^{D_p}$	2	20	22	21	24	21	27	8	38	8	54	47	1
gl	n.a.	+1	+1	-1	+1	-1	+1	-1	+1	-1	+1	+1	+1
ct	n.a.	bve	bev	bve	hve	hve	hve	bev	bev	bev	bve	bve	bev

(c) After Generation:

$\hat{\Omega}_r^{D_p}$	emp.	bve	bve bev	bev	bev hve	bev	bev hve	hve	hve bev	hve	hve bve	hve bve bve	hve bve bve hev

(d) After Numbering:

$\hat{\Omega}_r^{D_p}$	emp.	bve-1	bve-1 bev-1	bev-1	bev-1 hve-1	bev-1	bev-1 hve-2	hve-2	hve-2 bev-2	hve-2	hve-2 bve-2	hve-2 bve-2 bve-3	hve-2 bve-2 bve-3 hev-1

(e) Demonstration 2

$\Omega_r^{D_p}$	emp.	b-8	... ...	b-10	b-10 i-4	b-10	b-10 a-11	b-10 a-11 d-17	b-10 a-11 d-17 d-16	b-10 a-11 d-16	b-10 d-16	b-10 d-16 c-15	b-10 d-16 c-15 f-6

(f) PM Returns

$\gamma_r^{D_p}$	2	5	...	8	38	8	9	65	60	55	54	47	1
gl	n.a.	+1	...	-1	+1	-1	+1	+1	+1	-1	-1	+1	+1
ct	n.a.	hve	...	hve	bev	hve	hve	bve	bve	bve	hve	bve	hev

(g) After Numbering:

$\hat{\Omega}_r^{D_p}$	emp.	hve-3	... ...	hve-4	hve-4 bev-4	hve-4	hve-4 hve-5	hve-4 hve-5 bve-4	hve-4 hve-5 bve-4 bve-5	hve-4 hve-5 bve-6	hve-6 bve-6	hve-6 bve-6 bve-7	hve-6 bve-6 bve-7 hev-3

(h) Set H:

(i) After Merging step 1:		(ii) After Merging step 2:
(27 bev-1 = bev-3	(47) hve-2 = hve-6	bev-1 = bev-3
(27) hve-2 = hve-4	(47) bve-2 = bve-6	bev-2 = bev-4
(8) hve-2 = hve-4	(47) bve-3 = bve-7	hve-2 = hve-4 = hve-6
(38) hve-2 = hve-4	(1) hve-2 = hve-6	bve-2 = bve-6
(38) bev-2 = bev-4	(1) bve-2 = bve-6	bve-3 = bve-7
(54) hve-2 = hve-6	(1) bve-3 = bve-7	
(54) bev-2 = bev-4	(1) hev-1 = hev-3	

(i) Demonstration 2 after Merging step 3

$\hat{\Omega}_r^{D_p}$	emp.	hve-3	... ...	hve-2	hve-2 bev-2	hve-2	hve-2 hve-5	hve-2 hve-5 bve-4	hve-2 hve-5 bve-4 bve-5	hve-2 hve-5 bve-2	hve-2 bve-2	hve-2 bve-2 bve-3	hve-2 bve-2 bve-3 hev-3

Table 3.1: D_1 and D_2 as examples for the generation, numbering and merging phases

A. $nc_{bev} = 1$; $nc_{bve} = 1$; $nc_{hev} = 1$; $nc_{hve} = 1$;

Then, commencing with the second [3] state in a demonstration D_p, ie. $\gamma_2^{D_p}$, we achieve numbering for each following state $\gamma_r^{D_p}$ according to:

B. if a constraint was gained between γ_{r-1}^{Dp} and $\gamma_r^{D_p}$ (ie. if $gl = +1$) then

1. set index of $\hat{\rho}_{rs}^{D_p}$ equal to $\hat{\rho}_{r-1,s}^{D_p}$ for each $\hat{\rho}_{r-1,s}^{D_p}$ in $\hat{\Omega}_{r-1}^{D_p}$
2. identify the additional constraint in $\hat{\Omega}_r^{D_p}$ that is not in $\hat{\Omega}_{r-1}^{D_p}$. Call it $*\hat{\rho}_{r,s}^{D_p}$
3. post-pend to $*\hat{\rho}_{r,s}^{D_p}$ an index equal to the value of the numbering counter of its type, (eg. if it is of type bev, then the numbering counter of its type is nc_{bev})
4. increment the numbering counter used in step **B.3** by one

C. if a constraint was lost between γ_{r-1}^{Dp} and $\gamma_r^{D_p}$ (ie. gl = -1) then

1. identify the constraint in $\hat{\Omega}_{r-1}^{D_p}$ that is not in $\hat{\Omega}_r^{D_p}$ as the lost constraint. Call it $*\hat{\rho}_{r,s}^{D_p}$
2. set index of $\hat{\rho}_{rs}^{D_p}$ equal to $\hat{\rho}_{r-1,s}^{D_p}$ for each $\hat{\rho}_{r-1,s}^{D_p}$ in $\hat{\Omega}_{r-1}^{D_p}$ that is not $*\hat{\rho}_{r,s}^{D_p}$

Upon completion of the numbering phase for each D_p, each counter (ie, nc_{bev}, etc.) is incremented by one to reflect that constraints present in the last state of D_p are unrelated to those in the second state of D_{p+1}. That is:

D. nc_{bev}++; nc_{bve}++; nc_{hev}++; nc_{hve}++;

We show in Table 3.1-part (d), an example of how the numbering phase operated in practice on D_1. The two main components in our numbering phase algorithm are conditions **B.** (ie. when a constraint was gained) and **C.** (when a constraint was lost). Table 3.1-part(d) shows examples of both cases. An example of a gained constraint case is state 22. Here the gained constraint is identified as the bev constraint, and it is given an index of 1,

[3] The first state in a demonstration is always the no contact state and so has no constraints to number.

equal to the value of nc_{bev} at the time. An example of a lost constraint case is state 21. Here the bve constraint in state 22 is determined as the lost constraint. The constraint set estimate for state 21 is then made equal to the constraint set estimate of state 22 less the bve-1 constraint. At the completion of numbering in D_1, $n_{bve} = 3$, $n_{bev} = 2$, $n_{hve} = 2$ and $n_{hev} = 1$. Each counter is then incremented by one, and numbering commences in D_2.

Occasionally there are cases where step **C.1** of the algorithm cannot be completed. This occurs for any $\gamma_r^{D_p}$ where there exists two or more constraints in $\hat{\Omega}_{r-1}^{D_p}$ of the same type as the lost constraint. In order to present an example of this situation, we show in part (g) of Table 3.1 how the numbering phase operated on demonstration D_2 (note that the sequence of constraint-sets, and PM output, for D_2 are shown in parts (e) and (f) of the table). We highlight, as an example, state 54 in D_2. It cannot be determined if hve-4 or hve-5 was lost in the transition to this state from state 55. In such cases, we adopt the conservative solution of numbering the "undecidable" constraint with the next available counter value. For example, in the case of hve-4 and hve-5 in state 55, an index of 6 was post-pended to the hve constraint in state 54. This is a conservative approach that treats the undecidable constraint as a completely new constraint. We shall see in the next section how the merging phase can allow the true identity of an undecidable constraint to be determined.

3.4.3 Merging

We noted in the numbering phase how sequential states in an assembly sequence contain many constraints in common. A key aspect of numbering was then to ensure that constraints in sequential states were identified as the same constraint, where appropriate. However, the numbering process has not taken into account an important, additional fact. We know that the constraint set for the same state in different demonstrations must be the same. For example, we know that the constraint set for state 8 in both D_1 (Table 3.1-part(d)) and D_2 (Table 3.1-part(g)) must be identical. Yet after numbering, our constraint set estimates for state 8 are {hve-2} in D_1 and {hve-4} in D_2. The role of the

merging phase is to identify and remedy this type of situation.

The merging phase involves three steps. In the first step we determine which $\hat{\rho}_{rs}^{D_p}$ refer to constraint estimates corresponding to the same constraint in reality. In the second step, we construct a set H with elements h_l: $l \in \mathbb{N}$ that are sets of $\hat{\rho}_{rs}^{D_p}$ identified in step 1. In the third and final step, we update the $\hat{\Omega}_r^{D_p}$ produced by the numbering phase to reflect the information in H. The first step in merging is achieved as follows. First concatenate D_1 to D_n into D_T:

A. $D_T = D_1, D_2, \ldots, D_{n_d}$

where n_d is the number of demonstrations in the demonstration set. Then for each state γ_r^D in D_T [4]:

B. for every other state $\gamma_{r+q}^D : q \in \mathbb{N}$ that occurs in D_T after γ_r^D, if $\gamma_r^D = \gamma_{r+q}^D$ then

1. identify corresponding constraints in γ_r^D and γ_{r+q}^D
2. create a set in H for each pair identified in step B.1.

Table 3.1-part(h)(i) shows the result of applying step one of the merging process to D_1 and D_2. Here states 27, 8, 38. 54, 47 and 1 provided the basis for building H (ie. they were the states in D_1 that were also in D_2). Taking state 38 as an example, we see that the occurrence of this state in both D_1 and D_2 has resulted in the identification of hve-2 and hve-4 as equal, and of bev-2 and bev-4 as equal.

The second step in the merging process removes redundant information in H. It does this by combining any h_l with common elements. It is achieved by comparing any two sets h_a and h_b in H. If $h_a \cap h_b$ is not empty then h_a and h_b are replaced with the set $h_a \cup h_b$. Table 3.1-part(h)(ii) shows the result of the second merging step. All repeat sets existing in H after step one are removed. In addition, hve-4 and hve-6 are identified as equal via the union of two h_l with the common element of hve-2.

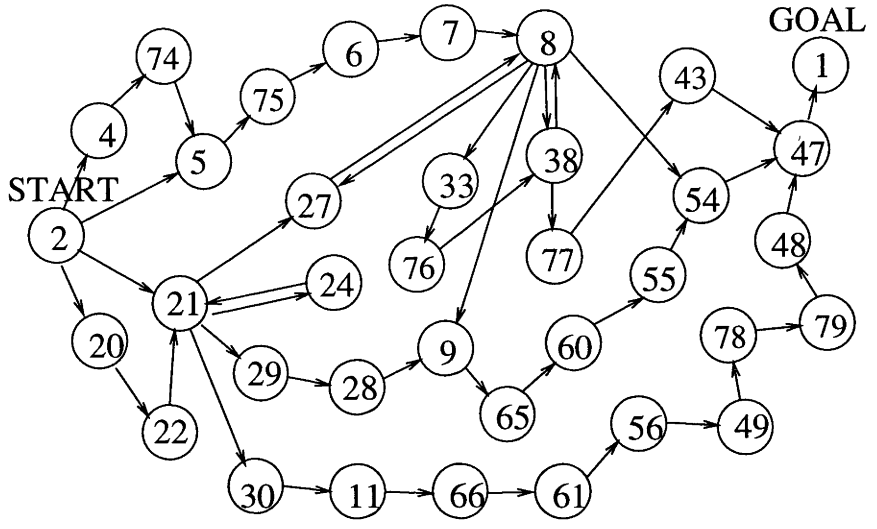
[4] Where we drop the p sub-script to D in $\gamma_r^{D_p}$ since we have concatenated all demonstration sequences into one state sequence.

The third step of merging updates the indices of every $\hat{\rho}_{rs}^{D_p}$ to reflect the information in H. If $\hat{\rho}_{rs}^{D_p}$ exists in some h_l in H, then we replace it with the element in h_l with the lowest index. Table 3.1-part(i) shows how merging step three has worked in practice. It shows the D_2 part of D_T after merging step three was completed. Note how hve-4 in state-8 has been relabeled as hve-2 to reflect that it corresponds the same constraint (ie. b-10) as in state 8 in D_1. In addition, recall from the numbering phase how an index of 6 was assigned to the hve constraint in state 54 in D_2. Remember how this occurred because we could not determine if hve-4 or hve-5 had been lost. Merging has determined that in fact constraint hve-5 was lost, and step three of the merging process has relabeled hve-6 to be hve-2 (which is equal to hve-4).

Once the merging phase is complete, our overall problem of determining the $\hat{\Omega}_i^A$ for each γ_i^A in the demonstration set is straight-forward. Our three phases of generation, numbering, and merging have determined a constraint set estimate $\hat{\Omega}_r^{D_p}$ for each $\gamma_r^{D_p}$. Then, for each state γ_i^A, we can simply read off $\hat{\Omega}_i^A$ as $\hat{\Omega}_r^{D_p}$ where $\gamma_r^{D_p}$ is equal to γ_i^A.

3.4.4 Results

Figure 3.1 shows the results of applying the generation, numbering, and merging process (*gnm-process*) to our demonstration set D_1 to D_6. It shows the automaton of states visited in the demonstration (we presented this automaton previously in Figure 2.7). In addition, for each state in the automaton, the figure shows the constraint set estimate $\hat{\Omega}_i^A$ derived by the gnm-process. For example, the gnm-process has determined that a constraint bev-1 exists in state 21. It has determined that the same constraint also exists in states 22, 24, 27, 28, 29 and 30. Recall that an important part the gnm-process was recognising the same constraint existing in different states as being the same. It turns out that constraint estimate bev-1 was correctly identified as the same constraint (h-3) in every state where it existed. However, we note that the gnm-process was not always able to correctly identify a constraint. To clarify our results in this regard, we present in Table 3.2 the set of true constraints existing in the demonstration set, along with the constraint



$\gamma_i^{\mathcal{A}}$	$\hat{\Omega}_i^{\mathcal{A}}$	$\gamma_i^{\mathcal{A}}$	$\hat{\Omega}_i^{\mathcal{A}}$
21	{bev-1}	74	{hev-5, hve-3}
29	{bev-1, hve-5}	75	{hve-3, hev-1}
11	{hev-8}	7	{hev-1, hve-2}
8	{hve-2}	20	{bve-1}
4	{hev-5}	22	{bve-1, bev-1}
5	{hve-3}	24	{bev-1, hve-1}
76	{hve-2, bve-12, bev-2}	33	{hve-2, bve-12}
38	{hve-2, bev-2}	43	{hve-2, bve-3}
30	{bev-1, hev-8}	28	{bev-1, hve-5, hve-2}
9	{hve-5, hve-2}	77	{hve-2, bev-2, bve-3}
6	{hev-1}	66	{hev-8, bev-9}
65	{hve-5, hve-2, bve-4}	60	{hve-5, hve-2, bve-4, bve-2}
55	{hve-5, hve-2, bve-2}	54	{hve-2, bve-2}
47	{hve-2, bve-2, bve-3}	61	{hev-8, bev-9, bve-2}
56	{hev-8, bve-2}	49	{hev-8, bve-2, bve-3}
78	{hev-8, bve-2, bve-3, hve-13}	79	{bve-2, bve-3, hve-13}
48	{bve-2, bve-3, hve-13, hve-14}	27	{hve-2, bev-1}
1	{hve-2, bve-2, bve-3, hev-1}		

Figure 3.1: Results of the generation, numbering and merging phases applied to demonstration set D_1 to D_6

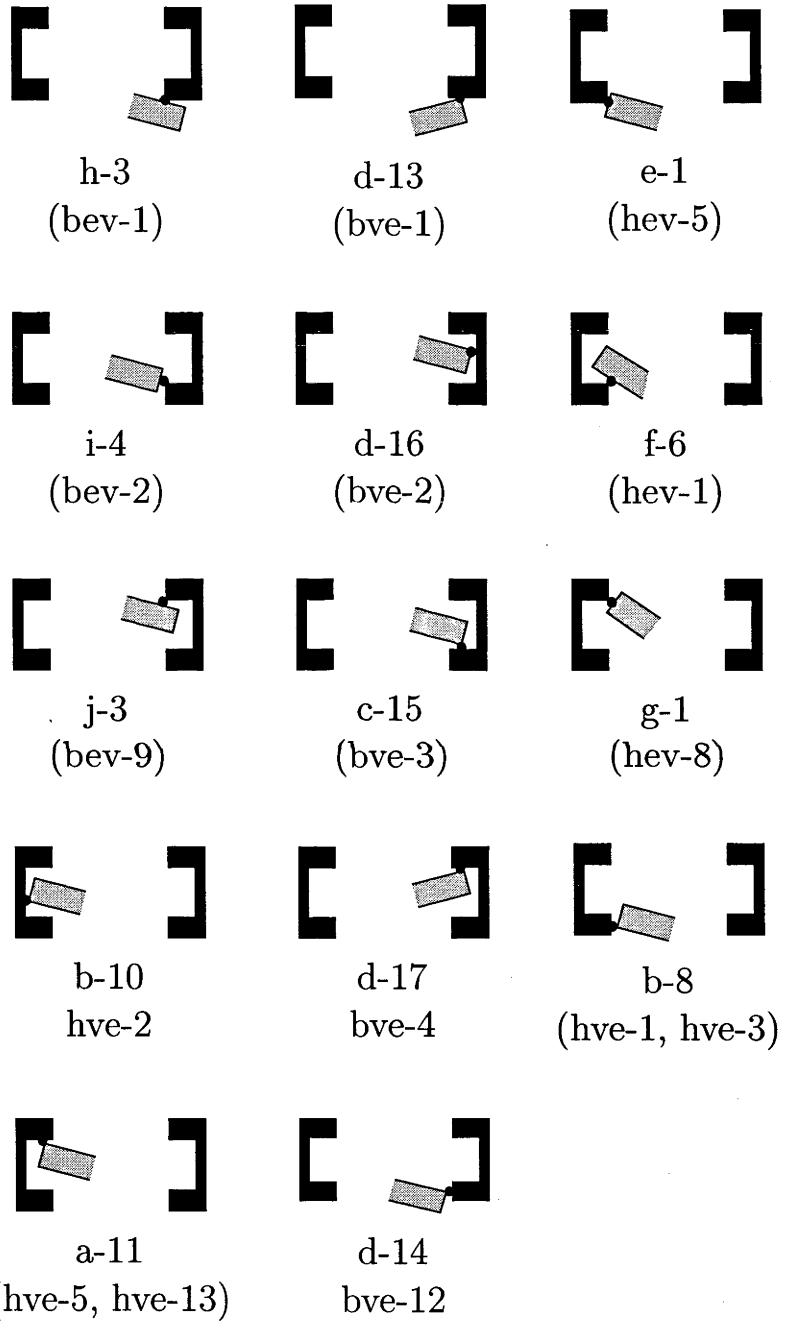


Table 3.2: The set of distinct constraints existing in the demonstration set

estimates derived for each by the gnm-process. Note how two estimates for each of the constraints b-8, and a-11 were determined by the gnm-process. For example, constraint b-8 was identified as both hve-1 and hve-3. In the majority of states where it existed, b-8 was identified as hve-3 (ie. in states 4, 5 and 75). Only in state 24 was it identified as hve-1. We refer to hve-1 and hve-3 as *repeat estimates* for b-8. The merging phase is designed to prevent the occurrence of repeat estimates. However in some cases, the state sequence visited in the demonstration can thwart the merging phase. For example, repeat states can occur where a state is visited one-off in isolation, eg. state 24. The merging phase may then not be able to fully ratify a constraint existing in that state with the same constraint existing in other states in the demonstration set. However, we note two things about the presence of repeat estimates. First, in most cases our gnm-process can correctly identify constraints. Since only a single constraint is gained or lost at each state change in a state sequence, a constraint will generally exist in a number of states, giving the merging phase an opportunity to identify it correctly. For example, in our experiments the majority of constraints were identified correctly. Table 3.2 shows that 12 out of the 14 constraints existing in the demonstration set were correctly identified in every state where they existed. The second thing we note about repeat estimates is that, while not optimal, we shall see later in this chapter, and in the next, how the existence of repeat estimates for a constraint do not prevent us from achieving our desired goal of noise removal in PbD.

3.5 Deriving Equations for Primitive C-surfaces

In the previous section we determined a constraint set estimate $\hat{\Omega}_i^A$ for each state in the demonstration set. In this section we solve the second part of the C-space derivation problem. Recall from Section 3.2 the second part of the problem. That is, we have determined for each demonstrated state γ_i^A a set of constraints $\hat{\Omega}_i^A$, where the j^{th} constraint in the set is $\hat{\rho}_{ij}^A$. Our problem now is to find an equation describing the primitive C-surface defined by $\hat{\rho}_{ij}^A$. The problem to be solved is simplified if we note that many $\hat{\rho}_{ij}^A$ are common

to a number of states in the demonstration set. For example, Figure 3.1 showed how hve-2 is a $\hat{\rho}_{ij}^{A}$ common to states 8, 76, 38, etc. Obviously we need only derive a primitive C-surface equation for the distinct set of $\hat{\rho}_{ij}^{A}$ in the demonstration set. That is, denote as Ω^* the distinct set of constraint estimates derived by the gnm-process in the previous section, and observe that Ω^* will have the general form:

$$\Omega^* = \{\rho_1^*, \ldots, \rho_m^*, \ldots, \rho_{n_m}^*\} \tag{3.2}$$

where ρ_m^* is the m^{th} distinct constraint estimate found by the gnm-process, and n_m is the number of distinct constraint estimates found. Denote as c_m^* the primitive C-surface defined by ρ_m^*. Then our problem in the second half of this chapter can be stated as: find the equation describing c_m^* for all $m = 1 \ldots m = n_m$.

We base our approach for deriving the equation of a c_m^* on statistical regression analysis. We described in Chapter 2 how the assembly process can be represented as a path traced out in C-space. Recall how this path will generally traverse on different C-surfaces in C-space. Then our idea for deriving the equation of a c_m^* is to use regression analysis on data points from demonstrated paths lying on c_m^*. Two things are required for each c_m^* before the regression analysis can take place. They are:

- A regression model: The regression model is a generic equation for c_m^*. By *generic* we mean the form of the equation is known, but that its parameters are not.
- A data set: That is, the set of points recorded from the demonstration where the human was traversing on c_m^*.

3.5.1 The Regression Model and Data Set

Determining the data set for c_m^* is straightforward. We note two things: (i) a continuous stream of data is output by the position sensor during the demonstration. By using discrete events identified by the PM as a trigger, this data stream can be divided into segments, where each segment corresponds to a distinct state in the demonstration sequence. The

second thing we note is (ii) that Figure 3.1 shows a set of constraints for each demonstrated state in the task. Then, a data set for c_m^* can be formed by simply concatenating segments of data from (i) for any state in Figure 3.1 where constraint ρ_m^* is present.

Determining a regression model for c_m^* is more complex. We previously introduced in Section 3.3 the idea of a constraint *type* for the spindle-assembly task. Four distinct constraint types were introduced: bev, bve, hev and hve. The reason for separating constraints into these types can now be made clear: constraints of the same type have the same regression model. That is, by returning the type *ct* of a constraint, the PM determines which regression model should be used for each c_m^*. Since four constraint types for the spindle-assembly task exist, four regression models are possible for c_m^*. We now present the details of how the regression model for each constraint type was derived.

At the basis of our regression model derivation are observations by Lozano-Perez. For a planar assembly task involving two polyhedral objects, two possible constraint types can exist in Ω^* [127]. If one object represents a workpiece (eg. spindle body) and the other the environment (right support), then the first constraint type is caused by a *vertex* of the manipulated workpiece in contact with an *edge* of the environment (ie. type bve). We show in Figure 3.2(a) how the c_m^* of this type of ρ_m^* is described by the vector equation [5]:

$$({}_a\mathbf{A} + {}_b\mathbf{C} - {}_a\mathbf{B}) \cdot {}_a\mathbf{n} = \mathbf{0} \tag{3.3}$$

The second constraint type is formed by an *edge* of the workpiece in contact with a *vertex* of the environment (type bev). We show in Figure 3.2(b) how the c_m^* of this type of ρ_m^* is described by the vector equation:

$$({}_a\mathbf{A} + {}_b\mathbf{B} - {}_a\mathbf{C}) \cdot {}_b\mathbf{n} = \mathbf{0} \tag{3.4}$$

Equations (3.3) and (3.4) form the set of possible regression models for the c_m^* that exist in

[5] We use the notation ${}_aA$ to mean vector A given with respect to frame F_a, ${}_bC$ to mean vector C with respect to frame F_b, etc.

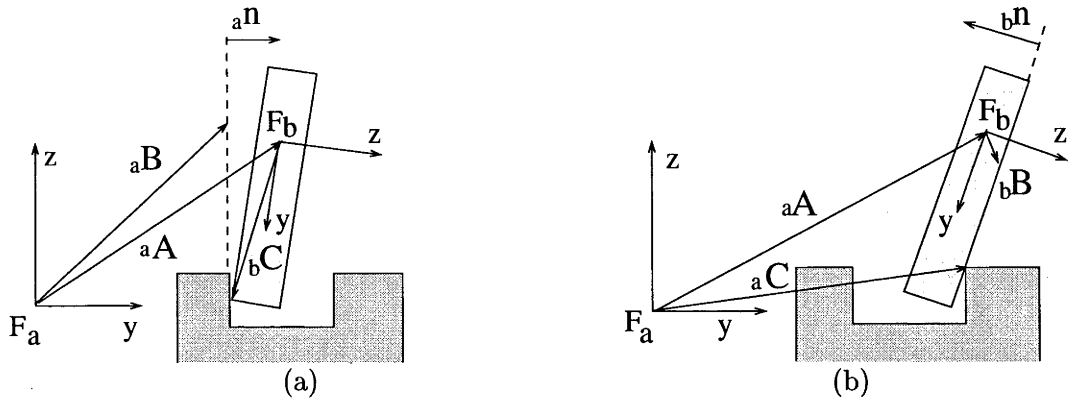


Figure 3.2: Regression model derivation for constraints caused by spindle body contact with the right support

a planar task involving two rigid-body objects. These vector equations form the regression models for constraints in the spindle insertion task caused by the spindle-body in contact with the right-support. That is, Equation (3.3) is the regression model for constraints of type bve, and Equation (3.4) is the regression model for constraints of type bev. Both equations (3.3) and (3.4) can be expanded to give scalar equations of the form:

$$\phi_1(y, z, \theta; b, c, d, e, f) = 0 \tag{3.5}$$

where y,z, and θ are the position and orientation of the manipulated body, and b,c,d,e and f are the regression model's unknown parameters. We note that the parameters in Equation (3.5) have physical meaning. For example, pair (c, d) gives the position of the vertex in the contact, relative to frame F_b in Equation (3.3), and relative to frame F_a Equation (3.4). This fact allows us to obtain the actual value of parameters by measurement; something we use later to verify the accuracy of parameter values obtained from the regression analysis.

For any planar task with a single-manipulated object, Equations (3.3) and (3.4) form the regression models of all constraints in the task. The spindle-assembly task is a planar task consisting of two manipulated objects (the spindle head and spindle body) with a single degree of freedom between them. Then a set of four regression models exists for the spindle insertion task. We have seen how the first two models correspond to constraint

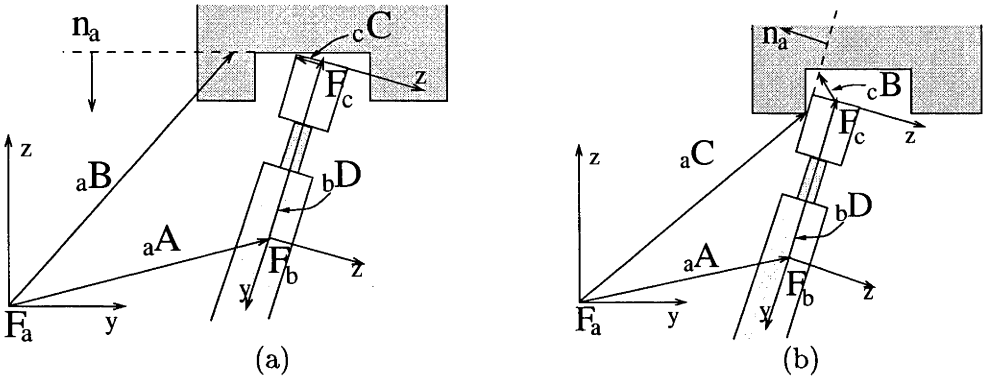


Figure 3.3: Regression model derivation for constraints caused by spindle head contact with the left support

types bev and bve. The second two models correspond to constraint types hev and hve. We show in Figure 3.3(a) how the regression model for constraints of type hev are given by the vector equation:

$$({}_a\mathbf{A} + {}_b\mathbf{D} + {}_c\mathbf{C} - {}_a\mathbf{B}) \cdot {}_a\mathbf{n} = \mathbf{0} \tag{3.6}$$

and in Figure 3.3(b) how the regression model for constraints of type hve are given by:

$$({}_a\mathbf{A} + {}_b\mathbf{D} + {}_c\mathbf{B} - {}_a\mathbf{C}) \cdot {}_b\mathbf{n} = \mathbf{0} \tag{3.7}$$

Both vector equations (3.6) and (3.7) can be expanded to give scalar equations of the form:

$$\phi_2(y, z, \theta, \delta; b, c, d, e, f) = 0 \tag{3.8}$$

where the additional variable δ in (3.8) compared to (3.5) describes the position of the spindle head relative to the spindle body. We note that our regression models (3.5) and (3.8) are non-linear in parameters b, c, d, e and f. However each model can be made linear in the set of transformed parameters $B_2 \ldots B_n$ by appropriate rearrangement of variables

y,z,θ, and δ into a set of transformed variables $X_1 \ldots X_n$ [57] [6]. That is, we can write a linear regression model for each constraint type in the general form:

$$[1, B_2, \ldots, B_n][X_1, \ldots, X_n]^T = 0 \tag{3.9}$$

Note that we linearize each model in this way in order to simplify the regression problem to be solved in the next section.

3.5.2 Regression Analysis

With the model and data set determined for a c_m^*, the regression analysis can proceed. We first form a system of linear equations out of the model and data set, of the form:

$$\mathbf{X}[1, B_2, \ldots, B_n]^T = 0 \tag{3.10}$$

where $\mathbf{X}$ is the data set, whose i^{th} row we denote as $X_{1_i}, \ldots, X_{n_i}$. Values in the data set $\mathbf{X}$ will contain some level of error since they are measured values returned by the Polhemus position sensor [7]. The idea in regression is to form a system of equations (3.10) that is over-constrained. The over-constraint is then used to minimize the effect of error in the data set $\mathbf{X}$ by finding parameters $B_2, \ldots, B_n$ that see the model best fit the points in the data set. There are a number of ways that *best fit* can be defined [131]. We choose the total least squares fitting method [44] (also known as linear orthogonal regression). Here the best fit is defined to occur when the Sum Square Error (SSE) is minimized, where SSE is given by:

$$SSE = \sum_{i=1}^{q} \frac{(X_{i_1} + B_2 X_{i_2} + \ldots + B_n X_{i_n})^2}{(1 + B_2^2 + \ldots + B_n^2)} \tag{3.11}$$

[6] Recall that n denotes the dimension of C-space.

[7] Measurement made by any sensor will contain some level of error. For example, because measurements made by the Polhemus sensor are transmitted as an electromagnetic signal, they can have errors introduced by metallic objects or stray magnetic fields existing in the vicinity of the sensor.

where q is the number of rows in the data set. Geometrically the approach can be interpreted as fitting to the data set, a hyper-plane which minimizes the sum of the squared Euclidean distances between each point in the data set and the hyper-plane. To solve equation (3.11) for the unknown parameters B_2 to B_n, we use the method based on Singular Value Decomposition [21] outlined in [44]. We have chosen to use the total least squares fitting approach because it is the most suitable method for our situation for the following reasons. First, all variables $X_1 \ldots X_n$ in the model contain error. This is in contrast to the more widely adopted fitting approach of ordinary least squares [57], where only one variable in the model is assumed to contain error. Second, it is reasonable to assume that the error in each variable is independent of the error in other variables, and that the error in all variables are normally distributed with zero mean and equal variance.

3.5.3 Results

Table 3.3 shows the results of using regression analysis to find primitive C-surface equations for the spindle-assembly task. The first column in the table lists each distinct primitive C-surface c_m^* in the demonstration set, while the second and third columns show the contact formation and constraint ρ_m^* defining each. Columns four to eight show two rows of parameter values for each c_m^*: an upper row showing the parameter estimates obtained by regression analysis (where the hat on each symbol b,c,d,e,f signifies that each value is an *estimate*), and a lower row showing a set of parameter values obtained by measurement (ie. the bottom row shows the true parameter values). Finally, column nine in the table lists the states that contributed to the data set of each c_m^*.

The parameter estimates determined by regression can be seen to range in their accuracy. In some cases the estimates were excellent, eg. c_{13}^*, c_1^*, c_{14}^*, c_9^* while in others they were less accurate. eg. c_2^*, c_3^*, c_4^*, c_7^*, c_{12}^*. Two requirements for accurate parameter estimates exist. The first is a sufficient amount of data, ie. that the system of equations formed by the data set in the regression analysis is sufficiently over-constrained. In our case, the position sensor was capable of data output at a rate of 120 Hz, so sufficient

c_m^*	Contact Formation	ρ_m^*	$\hat{b}$ / b	$\hat{c}$ / c	$\hat{d}$ / d	$\hat{e}$ / e	$\hat{f}$ / f	States in Data Set
c_1^*		bev-1	0.008 0.011	0.524 0.525	0.227 0.223	0.019 0	-0.998 1	(D_1) 22,21,24,21,27 (D_2) 27 (D3) 21,29,28 (D_4) 27 (D_5) 21,24,21,30 (D_6) 21,29,28
c_2^*		bev-2	0.002 0.011	0.612 0.540	0.316 0.223	0.992 1	0.127 0	(D_1) 38 (D_2) 38 (D_4) 76,38,77
c_3^*		bev-9	-0.023 0.011	0.523 0.570	0.310 0.223	0.730 0	0.634 1	(D_5) 66,61
c_4^*		bve-1	0.466 0.525	0.086 0.086	0.016 0.011	-0.972 -1	0.234 0	(D_1) 20,22
c_5^*		bve-2	0.183 0.213	0.096 0.086	0.002 0.011	0.033 0	0.999 1	(D_1) 54,47,1 (D_2/D_3/D_6) 55,54,47,1 (D_4) 47,1 (D_5) 79,48,47,1
c_6^*		bve-3	0.581 0.540	0.078 0.086	-0.004 0.011	0.996 1	0.088 0	(D_1) 47,1 (D_2) 47,1 (D_3) 47,1 (D_4) 77,43, 47,1 (D_5) 49,78,79,48,47,1 (D_6) 47,1
c_7^*		bve-4	0.447 0.570	-0.067 0.086	-0.118 0.011	-0.997 -1	0.079 0	(D_2) 65,60 (D_3) 65,60 (D_6) 65,60
c_8^*		bve-12	0.191 0.223	0.118 0.086	0.043 0.011	0.001 0	0.996 1	(D_4) 33,76
c_9^*		hev-2	0.018 -0.011	0.564 0.540	0.372 0.373	0.019 0	-0.998 -1	(D_2) 75,6,7 (D_4) 75,6,7
c_{10}^*		hev-5	0.024 0	0.512 0.525	0.353 0.373	-0.999 -1	-0.105 0	(D_4) 4,74
c_{11}^*		hev-8	-0.030 0	0.538 0.570	0.393 0.373	-0.985 -1	-0.120 0	(D_5) 11,66,61,56,49,78
c_{12}^*		hve-1	0.470 0.373	-0.097 0	0.157 -0.011	0.157 0	-0.988 -1	(D_1) 24 (D_5) 24
c_{13}^*		hve-2	0.377 0.383	0.005 0	-0.012 -0.011	0.009 0	-0.997 -1	(D_1)27,8,38,8,54,47,1 (D_3/D_6) 28,9,65,60, 55,54,47,1 (D_2) 7,8,27,8,38,8,9,65,60,55, 54,47,1 (D_4) 7,8,27,8,33,76,38,77,43,47,1
c_{14}^*		hve-3	0.366 0.373	-0.007 0	-0.014 -0.011	0.084 0	-0.996 -1	(D_2) 5,75 (D_4) 74,5,75
c_{15}^*		hve-5	0.599 0.560	-0.006 0	-0.014 0.011	0.084 -1	-0.996 0	(D_2) 9,65,60,55 (D_3) 9,65,60,55 (D_6) 9,65,60,55
c_{16}^*		hve-13	0.764 0.560	0.015 0	0.101 0.011	-0.980 -1	0.259 0	(D_5) 79,48

Table 3.3: Results of regression analysis for the spindle-assembly task

data was generally available for all c_m^*. The second requirement for accurate parameter estimation is a good range of data. ie. that the demonstrator traces out paths over a wide range on the C-surface. This was the reason for less accurate estimates in our case. For c_m^* with a number of paths over distinct areas of the C-surface, eg. c_{13}^*, c_1^*, c_9^*, the parameter estimates were excellent. However, for cases where the range of data was more limited, less accurate parameter estimates resulted.

There were two reasons why paths of limited range were traced out on a c_m^* in the demonstration. The first was because the c_m^* was only briefly visited, eg. c_3^*, c_4^*, c_{12}^*, c_{16}^*. For example, constraint bve-1 was briefly visited; existing in only two states (20 and 22) in demonstration D_1. This has resulted in the less accurate estimates for c-surface c_4^* shown in the table. What is required for this type of c_m^* is a larger demonstration set so that more paths on distinct parts of the C-surface become available. That is, the demonstration must contain sufficient information about a region in C-space if our method is to determine an accurate representation of the region. The second reason for limited path range on a c_m^* was because the geometry of the task limited the range of motion that could be demonstrated, ie. that the c_m^* only exists over a small region in C-space. For example, column nine of Table 3.3 shows that many paths contained constraint bve-3, so one would expect a good range of paths on c_6^*, and a set of precise parameter estimates. However, the table shows a difference between the parameter estimate values and their true values for bve-3 of up to 10 percent. In this case, the range of motion that can be demonstrated is naturally limited by the geometry of the task, ie. the spindle cannot move very far from an orientation aligned with the z-axis because it is lying between the rebates in each support. Although parameter estimates for this type of c_m^* can differ by up to 15 to 20 percent from the true values in some cases, these estimates still do in fact provide an accurate description of the c_m^* over the limited range of motion allowed by the task. That is, for noise removal purposes these parameter estimates provide an accurate description of the c_m^*. Noise removal means deriving noise-free paths that lie on a c_m^*. Parameter estimates that describe a c_m^* well over the limited range allowed by the task

are useful because our derived noise-free path will move onto a new C-surface (ie. the assembly process will move into a new discrete state) before reaching regions on a c_m^* described badly by the parameter estimates. The majority of the c_m^* with less accurate parameter estimates in Table 3.3 do so for this reason, eg. c_2^*, c_5^*, c_6^*, c_7^*, c_8^*, c_{11}^*, c_{15}^*. So many c_m^* of this type exist for the spindle-assembly task because only a very limited range of spindle motion is possible when it is close to fully inserted. Note how the c_m^* just listed correspond to constraints that occur when the spindle is lying between the rebate in each support.

Recall that in some cases the gnm-process of Section 3.4 resulted in the presence of repeat constraint estimates. For example, we saw how hve-1 and hve-3 were repeat constraint estimates found for the true constraint b-8 (see Table 3.2). It was stated at the time that repeat estimates do not prevent us from achieving noise removal in PbD. The results in Table 3.3 show why this is the case. For regression analysis, we treat the repeat estimates of a constraint as completely distinct constraints, ie. parameter estimates are obtained separately for each. Since each repeat estimate has a data set containing points generated by the same true constraint, the parameter estimates obtained for each should all reflect the true parameter values of that constraint. However the problem with repeat estimates, and the reason we identified them in Section 3.4 as being undesirable, is the data set used for each will be smaller than it could have been, ie. smaller than the data set formed by combining data sets of each repeat estimate. Smaller data sets can mean a less diverse range of points on a c_m^*, and so can lead to less accurate parameter estimates. For example, more accurate parameter estimates for both c_{15}^* and c_{16}^* may have resulted if they had been correctly identified as the same constraint.

Accurate Estimates where Most Required We have seen how our method provides a set of primitive C-surface equations useful for achieving noise removal in PbD. Prior to concluding work in this chapter, we identify an important feature of our method regarding its applicability to noise removal in PbD. We note that our method has the natural

tendency to generate an accurate description of c_m^* where noise removal is most in need. We most desire noise removal for c_m^* in two categories. First, for c_m^* pivotal to completing the task. In our spindle insertion task, c_{13}^* is a c_m^* of this type. It is visited at some point in every demonstration, and must be visited for the task to be successfully completed. We desire an accurate description of such c_m^* so that "noise-free" paths derived for the robot on these c_m^* exactly reflect the true topology of C-space. The derived paths must be of high quality because they will be used often and are critical to the completion of the task. Our approach tends to provide accurate estimates for this type of c_m^* because the constraints of such c_m^* are generally visited often in the demonstration, leading to a large data set with good range. Constraints hve-2, bev-1, bve-3, and bve-2 are pivotal constraints in the spindle-assembly task. Note how our method has provided accurate parameter estimates for the c_m^* of these constraints.

The second category of c_m^* where we desire noise removal are those on which particularly noisy paths were demonstrated. Our approach will tend to produce an accurate description of these c_m^* because noisy paths by definition tend to visit diverse parts of a C-surface. For example, we highlight the parameter estimates obtained for c_{10}^* shown in Table 3.3. These estimates are quite accurate given that only one path from the demonstration set was available for the regression analysis. The reason for accurate parameter estimates here was because a relatively noisy path was demonstrated. The human produced a path that saw the spindle orientation move from between 6.5 and 39 degrees to the z-axis, and 0 mm to 11.2 millimeters of spindle compression. In comparison we see the parameter estimates for c_4^* are less accurate, even though roughly the same amount of data was available in that case. The motion demonstrated on c_4^* was relatively noise free, with the spindle moving in close to a direct line through state 20 to state 22.

3.6 Conclusion

We have presented an approach for determining a partial knowledge of C-space for a task from demonstration. It was presented as a means for obtaining geometric information about a task for PbD in service robotic environments. We noted that an apriori-known, geometric task model is not generally available in such environments. The method we presented had the advantage over usual geometric-model-based approaches to constructing C-space of being demonstration based. That is, it provides a valid, alternative method for constructing C-space when a geometric model of the task is not available. A limitation of the approach is that sufficient information about a region in C-space must be available in the demonstration before an accurate description of that region can be determined. For the spindle-assembly task we saw in the majority of cases that sufficient information in the demonstration existed. An accurate description of C-space topology then resulted. However, in a small number of cases, less-accurate descriptions resulted because a region in C-space was visited only briefly in the demonstration. Our aim in this thesis is to derive noise-free, efficient and reliable control commands for the robot. That is, we should avoid deriving commands that traverse regions of C-space visited only briefly in the demonstration, since our description of that region may not be accurate. We will present later, in Chapter 5, a method for selecting task-level execution strategies so that control commands with exactly this property are avoided.

Chapter 4

Low-level Control Command Synthesis from Demonstration

4.1 Introduction

A core part of PbD is determining low-level control commands to be used by the robot in its execution of the task. These control commands must be based on the skillful actions used by the human in the demonstration. However, it is well known that humans can include in the demonstration, actions that are sub-optimal. Then demonstrated actions should not be used directly as control commands for the robot. Methods in PbD are required to identify and remove any sub-optimality from demonstrated actions so that robot control commands encode efficient and skillful execution of the task. The focus of this chapter is on presenting such a method. We saw in the previous chapter how a partial knowledge of C-space could be derived for a task. Our work in this chapter is based around that knowledge of C-space. We present a method that uses C-space information to determine from demonstration, a noise-free set of control commands that will see the robot efficiently complete the task.

This chapter is set out as follows. In Section 4.2 we formulate more precisely the problem to be solved. We identify that the control commands to be derived consist of

two parts: a force control part, and a position control part. Section 4.3 concentrates on synthesising the position control part of the command. A method based on the well known road-map approach to path planning is presented. We use the method to derive position control commands that encode efficient execution of the spindle-assembly task. In section 4.4 we synthesise the force control part of the control command. A force control command is derived that allows the robot to manipulate task parts in contact without chattering, losing desired contacts, or gaining undesired contacts. Finally, we end the chapter with Section 4.5, where the conclusions for this work are stated.

4.2 Problem Formulation

Our aim is to derive a set of control commands that see the robot complete the entire task. However, recall from Chapter 2 the details of our HDS skill model. Remember how the Event Path Planner (EPP) component of the HDS encoded as a *desired event path* a sequence of discrete events to achieve the task. It was then the role of the Discrete Event Controller (DEC) to determine a control command to achieve each event in that sequence. Then, in the context of HDS modeling, our problem here is to derive a functional DEC from demonstration. That is, our problem is to determine a noise-free control command that achieves each event in the desired event path.

Determining a desired event path that will achieve the task is the topic of chapter 5 in this thesis. However, we note here that work in Chapter 5 determines a desired event path using only demonstrated events. That is, to prepare for work in chapter 5, our problem here is to synthesize a DEC that can determine a control command between any two states in $\mathcal{A}_D$ for which a demonstrated event exists (we showed $\mathcal{A}_D$ in Figure 2.7). Recall from Chapter 2 how we denoted the desired event path as σ. Denote as τ_w^σ the w^{th} event in the sequence of events that is σ. Remember also from Chapter 2 how we denoted the control command output by the DEC as $u(t)$. Then our problem here is can be stated as deriving a $u(t)$ to achieve each τ_w^σ in σ. We note that σ specifies a sequence of events that

will complete the task, but that it also indirectly specifies a sequence of states, ie. the sequence of states traversed if all events in σ are achieved. Denote this state sequence as the *desired state path*, and assign it the symbol λ. Then each τ_w^σ is a transition between two states in λ. Denote the states between which τ_w^σ passes as γ_w^λ and γ_{w+1}^λ. Then our problem of deriving a $u(t)$ that triggers event τ_w^σ corresponds to finding a $u(t)$ that can be applied in state γ_w^λ in order to reach state γ_{w+1}^λ.

We saw in Chapter 2 how the DEC outputs $u(t)$ to the continuous controller component in the HDS, and that we selected a hybrid position/force controller as our continuous controller. Then the DEC derived in this chapter must output a control command consisting of both position and force control parts, ie. $u(t)$ must be of the form:

$$\mathbf{u}(t) = [\mathbf{P}(t), \mathbf{F}(t)]^T$$

where for our work here, we specify $\mathbf{P}(t)$ as a sequence of 3×1 position vectors $\mathbf{p}$ of the form:

$$\mathbf{p} = [y, z, \theta]^T$$

and $\mathbf{F}(t)$ as a sequence of 3×1 [1] force vectors of the form:

$$\mathbf{f}_p = [F_y, F_z, T]^T$$

where F_y and F_z signify force in the y and z directions, T the torque in the direction perpendicular to the y-z plane, and $\mathbf{f}_p$ is the force to be commanded at configuration $\mathbf{p}$ in the task.

Let $\dot{\mathbf{p}}_p$ be the time derivative of $\mathbf{P}(t)$ at $\mathbf{p}$. Then, a well known requirement on vectors $\dot{\mathbf{p}}_p$ and $\mathbf{f}_p$ is that they must be orthogonal [73, 53, 69]. That is, it is not generally possible to command force and velocity in the same direction. De Schutter et. al. [108] provide a method for determining orthogonal vectors $\dot{\mathbf{p}}_p$ and $\mathbf{f}_p$. We saw how our problem involves deriving a $u(t)$ in state γ_w^λ. Let $\mathbf{p}$ be the current task configuration in γ_w^λ, and recall that in C-space, $\mathbf{p}$ will specify a point on the C-surface of γ_w^λ. Denote the C-surface of γ_w^λ as c_w^λ. Then De Schutter's work identified that C-space is a manifold, in which exist

[1] While C-space for our spindle-assembly task has dimension four, both $\mathbf{p}$ and $\mathbf{f}$ have dimension three because the compression δ between the spindle head and body is not directly controllable. That is, δ must be controlled indirectly though control in the y, z, and θ directions.

C-surfaces, which themselves are manifolds. He identified that a tangent space can be determined at point $\mathbf{p}$ on our C-surface c_w^λ, and that $\dot{\mathbf{p}}_p$ must lie in the tangent space. He identified that $\mathbf{f}_p$ must lie in the dual space of the tangent space. Work by De Schutter means our problem of deriving a noise-free control vector $u(t)$ in γ_w^λ can be restated as two sub-problems:

(i) determine a noise-free position control command $\mathbf{P}(t)$ containing vectors $\mathbf{p}$ that specify points in C-space lying close together and exactly on c_w^λ, ie. points in $\mathbf{p}$ lying close together and exactly on c_w^λ mean that $\dot{\mathbf{p}}_p$ will always lie tangent to c_w^λ. By *exactly on* we mean that each $\mathbf{p}$ in $\mathbf{P}(t)$ satisfies the equation of c_w^λ. Remember that γ_w^λ is a state visited in the demonstration, and so we have an equation for c_w^λ from work in the previous chapter.

(ii) determine a noise-free force control command $\mathbf{F}(t)$ containing an $\mathbf{f}_p$ for each $\mathbf{p}$ in $\mathbf{P}(t)$, where $\mathbf{f}_p$ is orthogonal to c_w^λ at $\mathbf{p}$.

We present work to achieve sub-problem (i) in Section 4.3, while work to achieve sub-problem (ii) is presented in Section 4.4.

4.3 Position Control-Command Synthesis

4.3.1 Problem Formulation

We have seen how one requirement on the method for determining $\mathbf{P}(t)$ is that all $\mathbf{p}$ must specify points lying exactly on c_w^λ. We now specify five further requirements on the method. To assist in specifying these requirements, we present in Figure 4.1 an example of what our $\mathbf{P}(t)$ derivation problem means in C-space. Recall that in Chapter 2 we denoted as C_{con} the contact defining part of C-space. Figure 4.1 shows a simple example of one possible part of C_{con} for our assembly task. The non-shaded region in the figure corresponds to a 2-D C-surface of one of the states in the task. We have labelled this C-surface as c_A, and the state to which it corresponds (in parentheses) as γ_A. Recall from

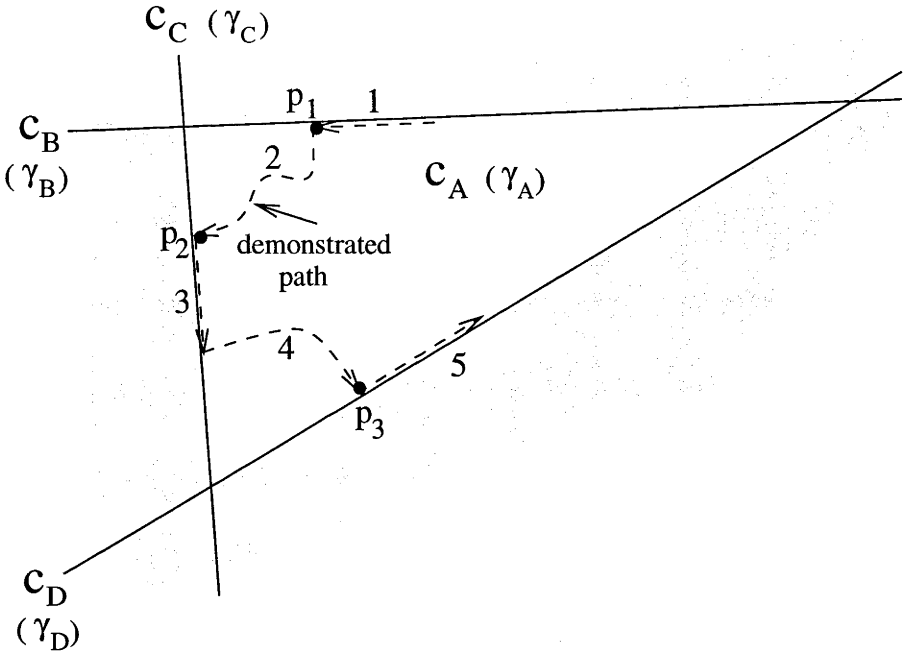


Figure 4.1: An example of C_{con} used to present five requirements on our method for position control command synthesis

Chapter 2 how the boundaries of a C-surface c_A are defined by the C-surfaces of states neighboring γ_A. For our example in Figure 4.1, lines labelled c_B, c_C, and c_D correspond to C-surfaces of states neighboring γ_A, and hence define boundaries on c_A. We have labelled (again in parenthesis) the states corresponding to c_B, c_C, and c_D as γ_B, γ_C, and γ_D. We know that in C-space, a demonstration is represented as a path that in general will visit C_{con}. We show in Figure 4.1 a demonstrated path that visits our example region of C_{con}. The demonstrated path shown consists of five segments, labelled 1 to 5. Note how different segments of the path lie on distinct C-surfaces in our example region. ie. on c_A to c_D. For example, segments 2 and 4 of the path exist on c_A.

With an example of the problem in C-space in place, we are now in a position to state the requirements on our $\mathbf{P}(t)$ derivation method. We identify that, in addition to our first requirement that all $\mathbf{p}$ lie exactly on c_w^λ, five further requirements on the method exist. They are:

1. the start and end points of $\mathbf{P}(t)$ must lie on the boundaries of c_w^λ:

   Denote the start and end points of the $\mathbf{P}(t)$ to be derived in state γ_w^λ as $\mathbf{p}_{w_s}^\lambda$ and $\mathbf{p}_{w_e}^\lambda$. Our problem is to derive a $\mathbf{P}(t)$ in γ_w^λ that will reach γ_{w+1}^λ. Clearly then, both $\mathbf{p}_{w_s}^\lambda$ and $\mathbf{p}_{w_e}^\lambda$ must lie on boundaries of c_w^λ. More specifically, our *end* point $\mathbf{p}_{w_e}^\lambda$ must lie on the boundary between c_w^λ and c_{w+1}^λ, where c_{w+1}^λ is the C-surface of γ_{w+1}^λ. The position of our *start* point $\mathbf{p}_{w_s}^\lambda$ will depend on the state previous to γ_w^λ in the desired state path, ie. on state γ_{w-1}^λ. Denote as c_{w-1}^λ the C-surface of γ_{w-1}^λ. Then our start point $\mathbf{p}_{w_s}^\lambda$ must lie on the boundary between c_w^λ and c_{w-1}^λ. Clarifying our requirement here with an example, if γ_B, γ_A, and γ_C in Figure 4.1 correspond respectively to γ_{w-1}^λ, γ_w^λ, and γ_{w+1}^λ, then the requirement on $\mathbf{P}(t)$ is that its start point must lie on the boundary c_B and its end point on boundary c_C.

2. the start and end points $\mathbf{p}_{w_s}^\lambda$ and $\mathbf{p}_{w_e}^\lambda$ must be demonstrated points:

   Many points on a boundary of c_w^λ exist. Our next requirement is that points $\mathbf{p}_{w_s}^\lambda$ and $\mathbf{p}_{w_e}^\lambda$ must be demonstrated points. A demonstrated point on the boundary between c_{w-1}^λ and c_w^λ, and between c_w^λ and c_{w+1}^λ, is guaranteed to exist because we know τ_w^σ and τ_{w+1}^σ were demonstrated events. In some cases, many demonstrated points on a boundary of c_w^λ will exist, eg. in Figure 4.1, all points in demonstrated segment 1 exist on the boundary between c_A and c_B. To make our problem more concrete in this thesis, we take the first and last demonstrated point in state γ_w^λ as the start and end points $\mathbf{p}_{w_s}^\lambda$ and $\mathbf{p}_{w_e}^\lambda$ of our control command $\mathbf{P}(t)$, eg. if our aim in Figure 4.1 is to derive a $\mathbf{P}(t)$ passing through γ_A from γ_B to γ_C, then we take p_1 as its start point and p_2 as its end point. Note that the combination of requirements 1 and 2 mean that two types of $\mathbf{P}(t)$ need to be derived by our method. First, those derived between demonstrated points where a demonstrated path exists, eg. in Figure 4.1, between points p_1 and p_2. Second, those derived between demonstrated points that are not *directly* connected by a demonstrated path, eg. between points p_1 and p_3.

3. the method must be valid for c_w^λ's of dimension ranging from 1 to n

We want the method to derive paths for any state in the task. The no contact state (ie. state 2 in $\mathcal{A}$) is a special case because it does not define a C-surface as such, but can be viewed to define a hyper-surface of dimension equal to the dimension of C-space, (ie. of dimension n). Our first requirement is that our method can derive $\mathbf{P}(t)$ on this hyper-surface. The remaining states in the task define C-surfaces ranging in dimension from $n-1$ to zero. States with C-surfaces of dimension zero correspond to spindle configurations with zero degrees of motion freedom. For example, in Figure 4.1 a C-surface of dimension zero is formed by the intersection of single-dimensional C-surfaces c_B and c_C, or by c_C and c_D, etc. Zero dimensional C-surfaces correspond to a single point in C-space. Noisy demonstrated actions cannot exist on C-surfaces of dimension zero, hence our method need not be valid for such C-surfaces. Then, in total, our method must be valid for surfaces ranging in dimension from 1 to n.

4. the method must generate a $\mathbf{P}(t)$ with $\mathbf{p}$'s lying within known boundaries to c_w^λ

   We know that boundaries on c_w^λ are formed by C-surfaces of neighboring states to γ_w^λ. According to requirements 1 and 2, our desired $\mathbf{P}(t)$ will intersect with two of these boundaries, ie. at points $\mathbf{p}_s$ and $\mathbf{p}_e$. However, a further requirement on the method is that it generate $\mathbf{p}$'s lying within all other boundaries on c_w^λ. That is, a $\mathbf{P}(t)$ intersecting with boundaries other than c_{w+1}^λ at $\mathbf{p}_s$, and c_{w-1}^λ at $\mathbf{p}_e$, will cause the assembly process to move into a state that is not γ_{w+1}^λ

5. the method must cope with the existence of unknown boundaries to c_w^λ

   Only a partial knowledge of C-space was derived in the previous chapter. That is, unknown boundaries to c_w^λ can exist. The method we present must derive a $\mathbf{P}(t)$ that traverses regions on c_{w+1}^λ where unknown boundaries are unlikely to exist.

4.3.2 Overview

Prior to presenting the details of a method that fulfills our six requirements, we first provide the reader with an overview of the approach. We have seen that not all boundaries on c_w^λ

are guaranteed to be known. We show in Figure 4.2 the same example region of C_{con} that was previously presented in Figure 4.1. Recall that c_B, c_C and c_D were known boundaries on c_A. In Figure 4.2 we show an additional boundary on c_B (labelled c_E) that is unknown (ie. it was not visited in the demonstration). Then we note that some portion of a known boundary may exist behind an unknown boundary, eg. the segments of c_B and c_D labelled B3 and D3, and hence do not really divide an "obstacle-free" region from an "obstacle-defining" region. We note that a boundary on c_A is guaranteed to divide obstacle-free and obstacle-defining regions along any demonstrated segment that exists on a boundary of c_A, eg. along segments 1, 3, 5. We call such segments, *boundary segments.* We observe that if the C-surface is of finite size (as is usually the case), then a point immediately in front of the boundary segment will be obstacle free, eg. points Q_1, Q_2, and Q_3. We use this observation as the basis for growing obstacle free regions on the C-surface. We grow a free region in front of each boundary segment of a C-surface, and use path planning [65] techniques to derive noise-free paths within the region. We note that if the region in front of a boundary segment is grown very small, then we are guaranteed that it will be obstacle free. However a small region is of limited use for path planning purposes. Hence we grow a region of useful size and accept that the region will only likely be obstacle free. We call such a region a *likely free region.* We use a road-map type approach to path planning within each likely free region, similar to those presented in [66, 81, 124]. Points are randomly generated within the region, and a simple path planner is used to create a connectivity graph $\mathcal{L}$ that records which points have an obstacle free path between them. Apart from points in likely free regions, we also know that points in demonstrated paths interior to the known boundaries of a C-surface are obstacle free, eg. for c_A in Figure 4.2, points in segments 2 and 4. We call such segments *interior segments.* Again we use a simple path planner to create a connectivity graph $\mathcal{D}$ that records the connectivity between points in different interior segments. Finally we combine graphs $\mathcal{L}$ and $\mathcal{D}$ into a graph $\mathcal{K}$ that represents the connectivity of all obstacle free points on the C-surface. We identify the nodes in $\mathcal{K}$ that represent the start and end points of $\mathbf{P}(t)$. We search for the

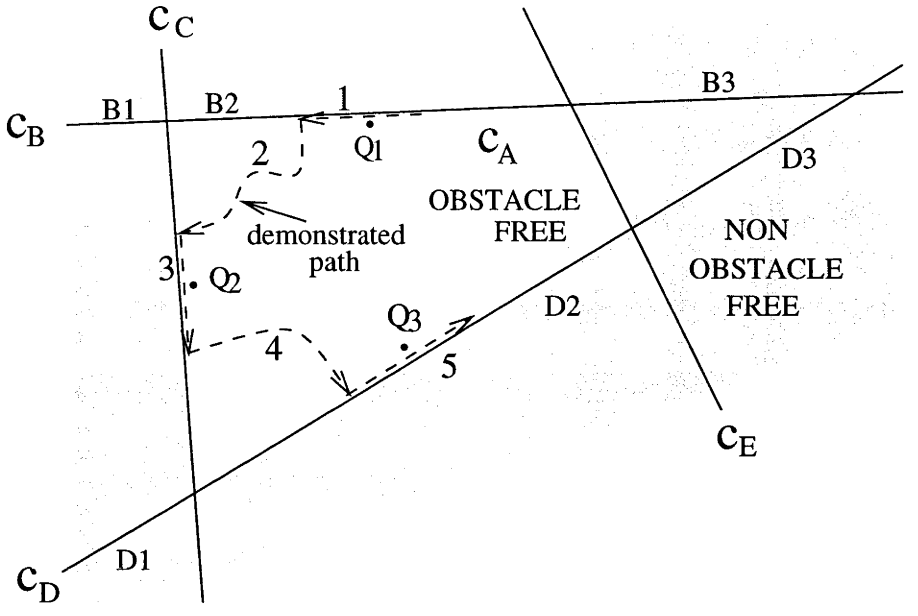


Figure 4.2: Demonstration segments identify regions on a C-surface that are likely to be obstacle free

minimum cost path existing in $\mathcal{K}$ between these nodes to give the final noise-free path. The process can be divided into four distinct steps. They are:

- Creating boundary segments
- Growing likely free regions
- Creating interior segments
- Creating a connectivity graph $\mathcal{K}$, and searching $\mathcal{K}$ for our final noise-free $\mathbf{P}(t)$

We present the details of each step in the following four sub-sections.

4.3.3 Creating Boundary Segments

Three steps are required to create boundary segments for γ_w^λ, (i) finding boundary states to γ_w^λ, (ii) identifying *raw* boundary segments for the boundary states found in (i), and (iii) projecting points in raw boundary segments to create a set of *clean* boundary segments.

To achieve (i), recall that each state in the HDS is defined by a set of constraints. Denote as Ω_w^λ the set of constraints that define γ_w^λ. Then we choose a *boundary state* of γ_w^λ as any other demonstrated state that is defined by the set of constraints Ω_{bnd}, where $\Omega_{bnd} \supset \Omega_w^\lambda$. That is, constraints present in γ_w^λ will also be present in its boundary states. Step (ii) is then straightforward. We select as a raw boundary segment, any path demonstrated in the boundary states of γ_w^λ. Step (iii) is required because points in raw boundary segments will not generally lie exactly on the C-surface of the boundary state. That is, the regression analysis of Chapter 3 derived C-surface equations that *best-fit* raw demonstration data. We create clean boundary segments by "orthogonally projecting" all points in a raw boundary segment onto the C-surface of the boundary state (in detail, we achieve this orthogonal projection for each point in a raw boundary segment by finding - using standard optimisation techniques - the point on the C-surface lying a minimum distance from the "raw" point). From now on, we will refer to clean boundary segments simply as *boundary segments.*

We show as an example in Figure 4.3 the results of creating boundary segments for state 8 of the spindle insertion task. The figure shows points in boundary segments for four boundary states of state 8. It shows these points in state 7 in Figure 4.3(a), in state 33 in Figure 4.3(b), in state 9 in Figure 4.3(c), and in state 54 in Figure 4.3(d). Note that we present boundary segment points in the figure as spindle configurations in the physical workspace rather than as single points in C-space. This is necessary due to the difficulties involved with graphically presenting points in a 4-D space. Note also that only a subset of points in each boundary segment are presented for reasons of clarity, where the subset was chosen to reflect the range of points existing in the boundary segment.

Two main things can be said about our examples in Figure 4.3. First, boundary states to state 8 have been correctly identified. Set $\{hve - 2\}$ is the constraint set that defines state 8. Figure 3.1 confirms that the constraint sets of states 7, 33, 9, and 54 also contain constraint $\{hve - 2\}$. Second, points in each boundary state result in a precise contact between the spindle and supports. That is, configurations shown in Figure 4.3 do not

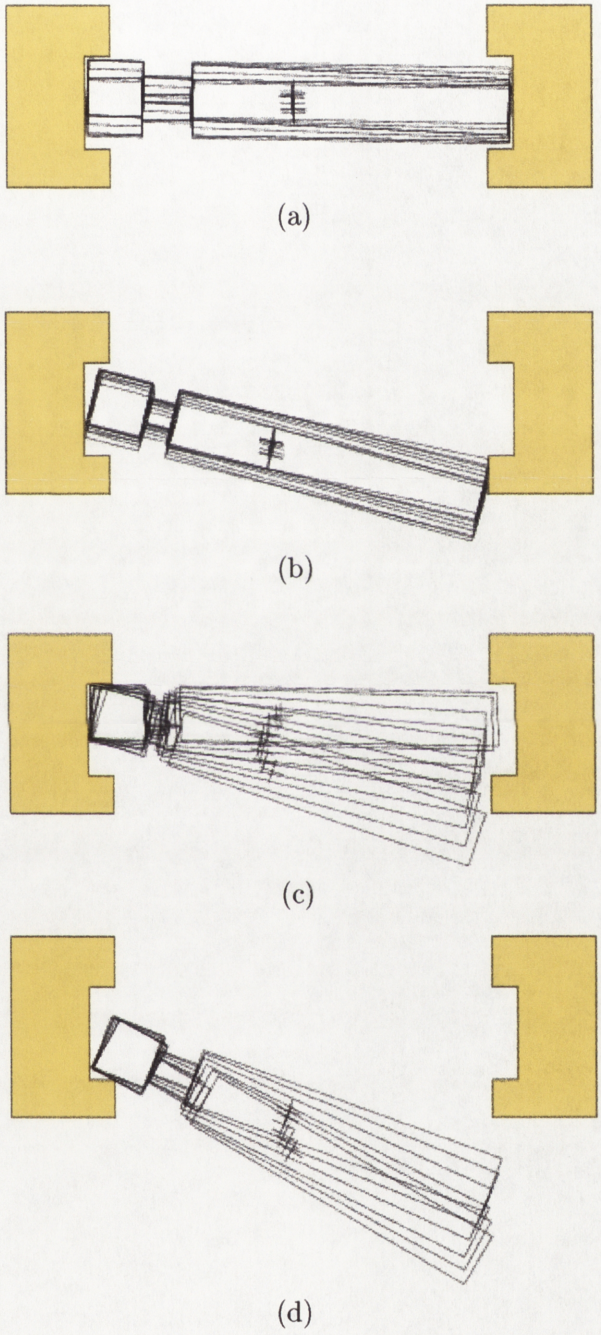


Figure 4.3: Four examples of boundary segments for state 8, (a) in state 7, (b) in state 38, (c) in state 9, and (d) in state 54

see the spindle lose contact with, or pass into the supports. This is a consequence of the projection step in the process, ie. we ensured that points in the final boundary segments lay exactly on the C-surface of the boundary state.

4.3.4 Growing Likely Free Regions

We grow a likely free region by generating a region of points on our C-surface c_w^λ immediately in front of a boundary segment. Each point in the region is determined as follows. First, a point R in the boundary segment is randomly selected. Next, a distance value dst[2] is randomly chosen using the uniform probability distribution over the interval (0, md]. Parameter md denotes a maximum distance value, and determines how big the likely free region is grown. A point in the likely free region is then determined by generating a point Q that (a) lies a distance dst from R, and (b) lies on our C-surface c_w^λ. We saw how, in addition to (a) and (b), point Q must also satisfy the condition that it lie within known C-surface boundaries. We refer to the region on c_w^λ lying within known boundaries as the *bounded region* for c_w^λ. For example, we present in Figure 4.4 a 2-D C-surface with six boundaries c_1 to c_6, each described by equations $\phi_1 = 0$ to $\phi_6 = 0$ respectively. Here, the bounded region consists of the union of regions labelled 1, 2, and 3. Note how the bounded region defines a set of points on c_w^λ that are obstacle free, ie. we wish to generate only Q lying within the bounded region of c_w^λ. We now present the details of how we ensure that only Q lying inside the bounded region are generated.

Let $\phi_{bnd} = 0$ be the equation of the C-surface for γ_{bnd}, a boundary state to γ_w^λ. That is, equation $\phi_{bnd} = 0$ defines a known boundary on our C-surface c_w^λ. Then we identify that any point Q we generate will satisfy one of the following conditions:

$$-\mathsf{eps} < \phi_{bnd}|_Q < \mathsf{eps} \tag{4.1}$$

[2]Note that we use symbols in plain upright text to denote parameters of our $\mathbf{P}(t)$ derivation method.

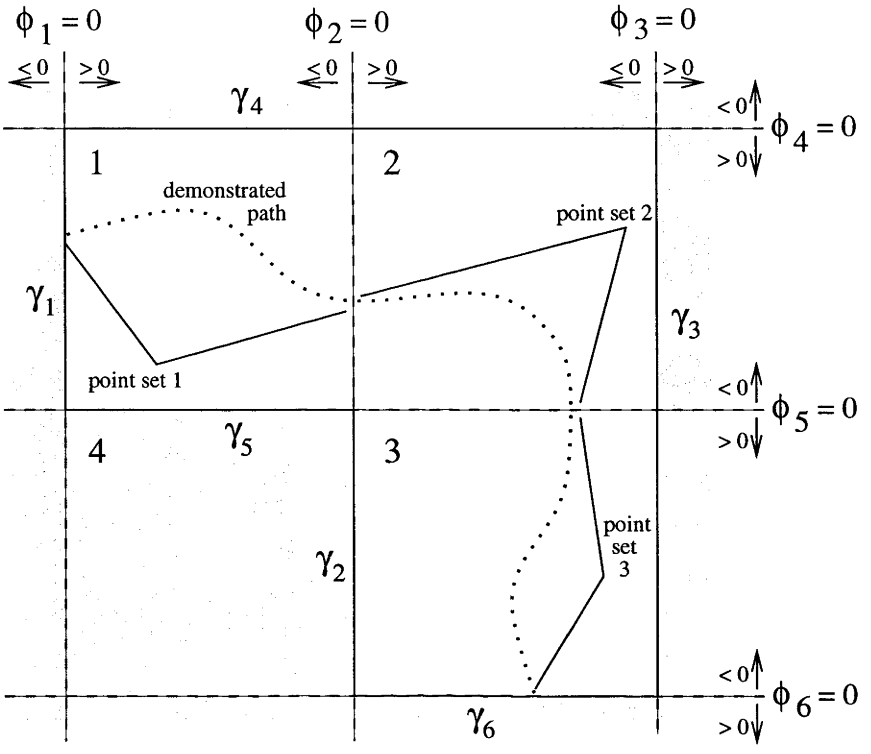


Figure 4.4: Example of how demonstrated points determine a set of valid bounded subregions

$$\phi_{bnd}|_Q \leq -\text{eps} \tag{4.2}$$

$$\phi_{bnd}|_Q \geq \text{eps} \tag{4.3}$$

where $|_Q$ denotes the equation ϕ_{bnd} evaluated at point Q, and where eps is a parameter of small value. If (4.1) is satisfied, then Q lies on, or very close to the boundary, while if (4.2) or (4.3) are satisfied, Q lies to one side of the boundary. We are never interested in Q's that satisfy (4.1). A Q lying exactly on the boundary is not obstacle free, and a Q lying very close to the boundary may not be obstacle free (recall that the equations we derived for C-surfaces in 3 are best-estimates that contain some error). Rather, we are interested in generating Q that satisfy either (4.2) or (4.3). Note that the choice of whether we want

Q to satisfy (4.2) or (4.3) will depend on which side of the boundary is obstacle free.

In general, there will be more than a single known boundary to our C-surface c_w^λ. Let n_b denote generally the number of known boundaries on c_w^λ. Denote the equations of boundaries 1 to n_b respectively as:

$$\phi_{bnd\_1} = 0, \quad \phi_{bnd\_2} = 0, \quad \ldots \quad , \phi_{bnd\_n_b} = 0 \tag{4.4}$$

We note that a Q which satisfies one or more equations in (4.4) will lie on a boundary of the bounded region. We saw that we are not interested in such Q. Rather, we want Q that lie on the obstacle free side of all boundaries by at least a distance eps. That is, we must form a set of inequalities by *casting* each equation in (4.4) to be an inequality of the form (4.2) or (4.3). Call such a set of inequalities a *boundary inequality set.* Then we note that a boundary inequality set defines a region on our C-surface c_w^λ. For example, one boundary inequality set for the boundary equations $\phi_1 = 0$ to $\phi_6 = 0$ of our example in Figure 4.4 would be:

$$\begin{aligned} &\phi_1 \geq \mathsf{eps}, \quad \phi_2 \leq -\mathsf{eps}, \quad \phi_3 \leq -\mathsf{eps}, \\ &\phi_4 \geq \mathsf{eps}, \quad \phi_5 \leq -\mathsf{eps}, \quad \phi_6 \leq -\mathsf{eps} \end{aligned} \tag{4.5}$$

Then the region on the 2-D C-surface in Figure 4.4 that is defined by the boundary inequality set (4.5) is region 1. That is, a point Q is guaranteed to lie in region 1 if it satisfies all inequalities in (4.5). We note that in general the bounded region on c_w^λ cannot be specified by a single boundary inequality set. Denote as a *bounded sub-region* the region defined by a single boundary inequality set, ie. region 1 in Figure 4.4 is a bounded sub-region. Then, we identify that the bounded region of a C-surface can be specified as a union of bounded sub-regions. For example, the bounded region in Figure 4.4 is given by the union of bounded sub-regions 1, 2, and 3. A point Q can then be tested to see if

it lies within the bounded region by testing to see if it lies within any of its component bounded sub-regions. The question is then one of how to select the set of bounded sub-regions that make up our bounded region. Note that not all bounded sub-regions that can be generated for a certain set of boundary equations will be obstacle free. For example, region 4 in Figure 4.4 is a valid bounded sub-region, however region 4 defines an obstacle.

Our solution is to use the set of demonstration points on the C-surface to determine what bounded sub-regions are valid. Let μ be a possible bounded sub-region for c_w^λ, and Υ be the boundary inequality set that defines μ. Then we say that μ is a valid bounded sub-region if there exists a point in the demonstration set on c_w^λ which satisfies all inequalities in Υ. For example, in Figure 4.4 the demonstrated points in point-set 1 show region 1 to be a valid bounded sub-region because they satisfy all inequalities in (4.5). Similarly, demonstration point sets 2 and 3 show regions 2 and 3 as valid bounded sub-regions. Region 4 is not included as a valid bounded sub-region because it does not contain any demonstrated points. This approach is a conservative solution because we may miss some valid bounded sub-regions in which no demonstrated points exist. However, we take this approach because it guarantees that we do not generate Q lying in bounded sub-regions that define obstacles.

We now present some examples of the likely free region generation process. We show in Figure 4.5 the points in four likely free regions that were generated on the C-surface $c^{\mathcal{A}_8}$ (ie. the C-surface of state 8 in $\mathcal{A}$. The likely free regions shown were grown from boundary segments in state 7 (Figure 4.5(a)), in state 38 (Figure 4.5(b)), in state 9 (Figure 4.5(c)), and in state 54 (Figure 4.5(d)). Note that for clarity, we present only a subset of all points generated for each likely free region.

There are five main things to note about the spindle configurations in Figure 4.5. First, all configurations correspond to points in C-space that have been generated a distance eps away from boundaries on $c^{\mathcal{A}_8}$. For example, in Figure 4.5(c) all configurations maintain some distance between the base of the spindle and the lower support. Second, there exist for all boundary segments in Figure 4.5, precise contacts between the spindle head and

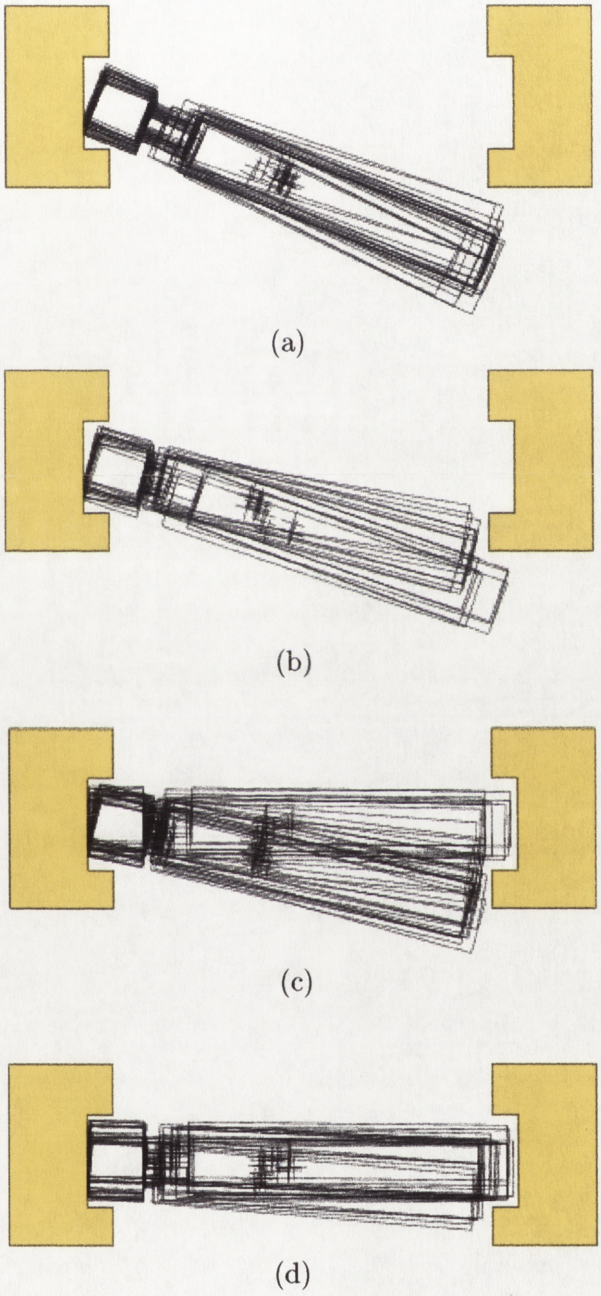


Figure 4.5: Spindle configurations generated for a number of likely free regions in state 8. Figures (a), (b), (c) and (d) correspond to likely free regions generated from boundary segments in state 7, 38, 9, and 54 respectively

the rebate in the top support. This results because we enforced condition that points Q were generated lying exactly on a c_w^λ, ie. exactly on $c^{\mathcal{A}_8}$. Third, all configurations lie in a region generally close to the boundary state. Recall that configurations lying "far" from a boundary state are undesirable because they may violate unknown boundaries on a c_w^λ. Note in Figure 4.5 how we do not produce such configurations. This is a consequence of limiting points generated on $c^{\mathcal{A}_8}$ to lie a distance less than md from boundary segments. Fourth, all configurations correspond to points in C-space that do not violate known boundaries. For example, note how in Figure 4.5(c) there exists no configuration where the bottom of the spindle passes into the lower support, even though such a configuration is "close" to our boundary segment in state 9. This occurs because our method only generates points on a c_w^λ on the obstacle-free side of known boundaries. Fifth and finally, note how each of the bounded regions in Figure 4.5 define by themselves only a small obstacle-free region on $c^{\mathcal{A}_8}$, but that together they provide an obstacle free region covering most of the area of interest on $c^{\mathcal{A}_8}$.

4.3.5 Creating Interior Segments

Denote as a *raw interior segment* a path demonstrated in state γ_w^λ. Recall that we saw previously in Figure 4.1, two demonstrated path segments on c_A labelled 2 and 4. Then segments 2 and 4 are raw interior segments for cA. We use the term *interior* because we know that, except for their start and end points, raw interior segments lie within known boundaries of c_w^λ. Our aim here is to derive *clean* interior segments. There are two requirements on clean interior segments to which raw interior segments do not comply. These are (i) points in raw interior segments do not lie exactly on c_w^λ (ie. work in 3 derived equations that best fit data points), and (ii) the start and end points in raw interior segments do not lie exactly on the boundaries of c_w^λ. We achieve (i) by orthogonally projecting all points in the raw interior segment onto c_w^λ (ie. using the same technique as described for raw boundary segments in Section 4.3.3). For (ii), we take the start and end points of each raw interior segment and orthogonally project them onto their respective

boundaries. A set of clean interior segments is achieved by repeating steps (i) and (ii) for all raw interior segments on c_w^{λ}. From now on, we will refer to clean interior segments simply as *interior segments.*

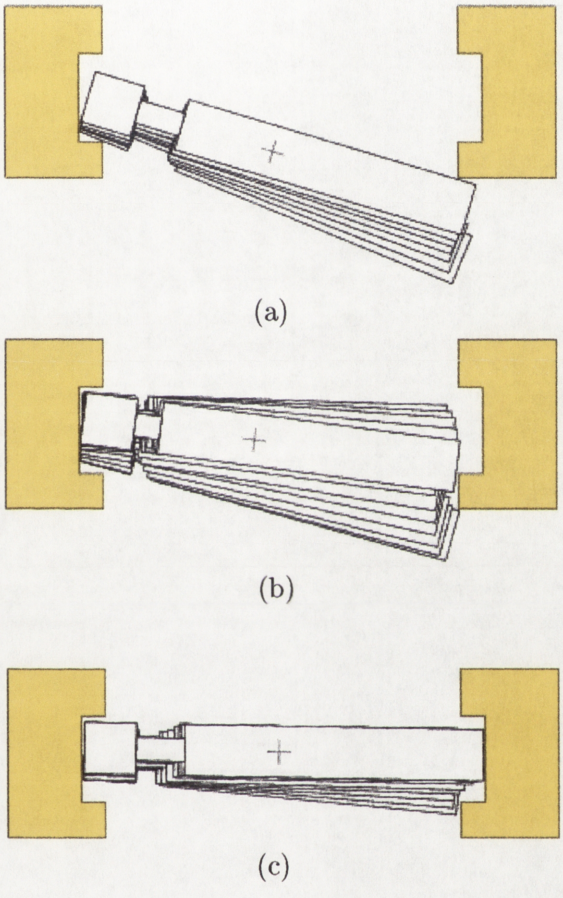


Figure 4.6: Three interior segments determined for state 8. The interior segments were derived from paths demonstrated in state 8, ie. from demonstration 4 (7-8-27) in (a), from demonstration 2 (27-8-38) in (b), and from demonstration 1 (38-8-54) in (c)

In Figure 4.6 we show an example of the interior segment generation process. The figure shows three interior segments that were produced on c_8^A. An interior segment in shown that passes through state 8: between states 7 and 27 in Figure 4.6(a), between states 27 and 38 in Figure 4.6(b), and between states 38 and 54 in Figure 4.6(c). There are three things to note about the spindle configurations in each interior segment. First, there

exist precise contacts between the spindle head and the rebate in the top support. Precise contacts result here because we orthogonally projected points in raw interior segments to lie exactly on $c_8^{\mathcal{A}}$. Second, all configurations (except the start and end configurations) in each interior segment correspond to points on $c_8^{\mathcal{A}}$ that lie within known boundaries. As such, the interior segments shown do not accidently cause spindle/support collisions that would see the assembly move into an incorrect state. Third, the start and end configurations of each interior segment lie exactly in boundary states to state 8. For example, the start and end configurations in Figure 4.6(c) correspond to points lying exactly on the C-surfaces of state 38 and 54. Start and end configurations lying in boundary states result because we orthogonally projected start and end points of raw interior segments onto the C-surfaces of boundary states to state 8.

4.3.6 Creating a Connectivity Graph

The previous two sections have been devoted to generating points on our C-surface c_w^{λ}. Two types of points were generated, points in likely free regions, and points in interior segments. We show in Figure 4.7 a possible outcome of the point generation process for a simple 2-D planar c_w^{λ}. The unshaded region in the figure corresponds to the bounded region of our 2-D C-surface. Three likely free regions on our bounded region have been identified (labelled A, B, and C). In addition, two interior segments (labelled η_1 and η_2) were also generated. Our aim in this section is to create a graph $\mathcal{K}$ that represents the connectivity between all generated points on a c_w^{λ}. For example, in our example in Figure 4.7, we wish to create a $\mathcal{K}$ that represents the connectivity between all points lying in likely free regions A, B, and C and interior segments η_1 and η_2. Such a graph should have a node to represent each point in a likely free region or interior segment. It should have arcs existing between nodes whose points are connected by an obstacle free path. In addition, we assign to each arc in the graph a value equal to the cost of traversing the obstacle free path it represents. We construct this graph $\mathcal{K}$ in three steps, (i) we create a graph $\mathcal{L}$ representing the connectivity between points in likely free regions, (ii) we create

a graph $\mathcal{D}$ representing the connectivity between points in interior segments, and (iii) we combine graphs $\mathcal{L}$ and $\mathcal{D}$ into $\mathcal{K}$.

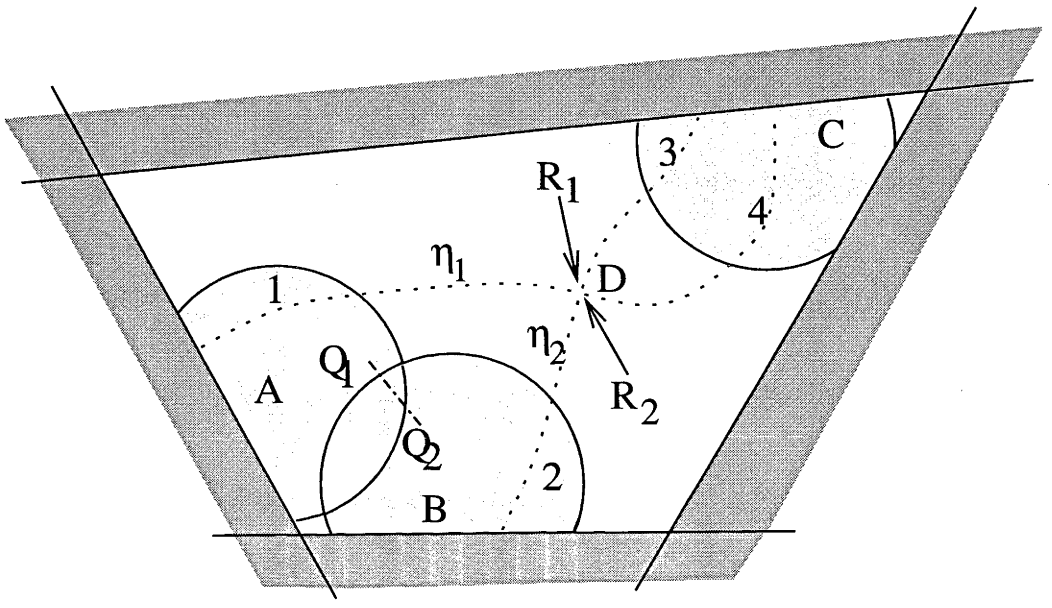


Figure 4.7: Example of points generated by our method for a simple, planar C-surface

Creating $\mathcal{L}$: We first create a point set consisting of all points in all likely free regions on c_w^λ. Then a node in $\mathcal{L}$ is created for each point in the point set. To construct arcs in $\mathcal{L}$ we must determine two things for each pair of points in the point set,

(a) if the points are connected

(b) the cost of traversing between connected points

We say that two points Q_1 and Q_2 in the point set are connected if two conditions are satisfied. First, that the straight line path between Q_1 and Q_2 is obstacle free, ie. that every point on the straight line lies within the bounded region of the C-surface. For example, points Q1 and Q2 in Figure 4.7 have a straight line path between them that lies fully inside the bounded region. Second, that the distance between Q_1 and Q_2 does not

exceed a maximum connected distance parameter mcd_l. We apply the second condition for the following reasons. First, note that our point set will consist of points from distinct likely free regions. We do not want to connect points in likely free regions that lie far apart, eg. between points in likely free regions A and C in Figure 4.7. Using this approach we are less likely to connect points on c_w^λ between which an unknown boundary exists. Second, intermediate points on the straight line between Q_1 and Q_2 will not in general lie exactly on the C-surface due to its curvature. However, if Q_1 and Q_2 do not lie too far apart, then intermediate points will lie close enough to the C-surface for the purpose of checking whether they are obstacle free. Third, applying such a condition is advantageous from a computational point of view. The number of points to be tested for connection increases rapidly as the allowed distance between the points increases. Fourth and finally, testing for connectivity only between points lying a distance less than mcd_l apart does not really detract from the end performance of our method. That is, points lying far apart that really should be connected, (eg. those existing in the same, or overlapping, likely free regions), will be connected efficiently at the final stage of the process. In general they will be connected by a compound path passing through a sequence of intermediate points, where intermediate points are separated by a distance of less then mcd_l.

Our second requirement for creating the arcs in $\mathcal{L}$ was (b) to determine their cost. We denote as ϖ_1 and ϖ_2 the nodes in $\mathcal{D}$ that represent points Q_1 and Q_2, and as ϑ the arc in $\mathcal{D}$ that connects ϖ_1 and ϖ_2. Then we calculate the cost of ϑ as the distance between Q_1 and Q_2. Distance is the appropriate measure of cost here, since we desire that a minimum cost path in the final connectivity graph $\mathcal{K}$ represents the shortest distance path on our C-surface.

Creating $\mathcal{D}$: We create $\mathcal{D}$ in two steps:

(i) create a distinct connectivity graph for each interior segment on c_w^λ (eg. distinct connectivity graphs for η_1 and η_2 in Figure 4.7)

(ii) combine graphs constructed in (a) into $\mathcal{D}$.

Step (i) is straight-forward because the connectivity of points in any interior segment is known, ie. apart from the start and end points, each point is connected to two other points, a previous point, and a following point. We create a graph for each interior segment with nodes and arcs that reflect this connectivity. We assign as costs to the arcs in the graph the distance between each sequential point pair in the interior segment.

In step (ii) we must combine the graphs created in step (i) into a single graph $\mathcal{D}$. The process of combining these graphs means deciding if and where interior segments on c_w^λ *intersect*, That is, we should create a connecting arc between two graphs $\mathcal{D}_1$ and $\mathcal{D}_2$ derived in step (i) when the interior segments η_1 and η_2 represented by these graphs are found to intersect, eg. at point D in Figure 4.7. We say that η_1 and η_2 intersect at points R_1 (in η_1) and R_2 (in η_2), when the distance between R_1 and R_2 is less than a parameter mcd_d. If this is the case, an arc is connected between the node in $\mathcal{D}_1$ that represents R_1 and the node in $\mathcal{D}_2$ that represents R_2. Then $\mathcal{D}$ is created by repeating this process for every possible R_1 and R_2 in our set of interior segments on c_w^λ.

Creating and Searching $\mathcal{K}$: We create $\mathcal{K}$ by combining graphs $\mathcal{L}$ and $\mathcal{D}$. The idea is to connect points in interior segments with those in likely free regions where an interior segment passes through a likely free region (eg. along segments labelled 1,2,3 and 4 in Figure 4.7). We create a $\mathcal{K}$ that represents such connectivity as follows. For each point R in each interior segment η, find a point Q in any likely free region that lies within a distance mcd_l away. If such Q's exist, then for each Q found, create an arc between the node that represents point R in $\mathcal{D}$, and the node that represents point Q in $\mathcal{L}$. Set the cost of the arc to the distance between the points R and Q.

Once $\mathcal{K}$ is achieved, then the solution to our problem of deriving a noise-free position control command $\mathbf{P}(t)$ is straightforward. $\mathcal{K}$ represents the connectivity of obstacle free points on c_w^λ. Two of the nodes in $\mathcal{K}$ are guaranteed to represent the start and end points $\mathbf{p}_{w_s}^\lambda$ and $\mathbf{p}_{w_e}^\lambda$ of $\mathbf{P}(t)$. Recall that we specify $\mathbf{P}(t)$ as a sequence of spindle configurations

$\mathbf{p}$, that pass between $\mathbf{p}^{\lambda}_{w_s}$ and $\mathbf{p}^{\lambda}_{w_e}$. Then to derive $\mathbf{P}(t)$, we determine a sequence of $\mathbf{p}$ by conducting a search in $\mathcal{K}$ for the minimum cost path between the nodes that represent $\mathbf{p}^{\lambda}_{w_s}$ and $\mathbf{p}^{\lambda}_{w_e}$.

4.3.7 Setting Parameters to Appropriate Values

The method we have presented for determining $\mathbf{P}(t)$ is based around a number of parameters, ie. md, dst, eps, mcd_l and mcd_d. A key issue in ensuring good performance from the method is of setting these parameters to appropriate values. The purpose of this section is to provide a guide for selecting appropriate values for the parameters in our method. Our aim here is not to provide an in-depth analysis on what are the optimal values for parameters. We leave this for future work. Rather, our aim is to present the set of simple metrics that we used for setting parameter values for the experiments presented later in this chapter.

md: Parameter md is the most important parameter in our method. Recall that the value of md determines how large a likely free region is grown. A balance must be struck in setting md between the benefit of large likely-free regions for path generation purposes, and the risk of generating points that lie on the non-obstacle-free side of unknown boundaries. We note that a good gauge on the value of md can be made by looking at the length of the boundary segment from which the likely-free region is grown. The probability of an unknown boundary passing close to our boundary segment, but not passing through it, decreases as the length of the boundary segment increases. That is, the larger the boundary segment, the larger md should be made. For our experiments in this chapter md for each boundary segment was set to a value of the length of the boundary segment multiplied by a constant value three. We found empirically that the value three provided conservatively sized likely-free regions that were still large enough to be useful for path planning purposes.

dst: Once md is determined, parameter dst is also determined. Recall that we set the value of this parameter as a random value between md and zero.

eps: The purpose of parameter eps is to ensure we do not generate points in likely-free regions close to, or exactly on, known boundaries to a C-surface. The reason for this approach was because the regression analysis of the previous chapter provided only an estimate of the true equation for each boundary on our C-surface. That is, each boundary on our C-surface will be formed by one of the primitive C-surfaces identified in the previous chapter, and our equation for that surface will contain error. A good estimate of the amount of error that exists is provided by the Root Mean Squared Error (RMSE), a metric often calculated in regression analysis for determining the "goodness of fit" [57]. Essentially the RMSE is the average distance between a fitted surface and the raw data points from which the surface was regressed. A RMSE value will result from the regression analysis for each boundary on our C-surface. Then, for our experiments in this chapter, we set eps equal to three times the RMSE value determined for a particular boundary when growing a likely free region from that boundary. Three times the RMSE will encompass 99.8 percent of the raw data points if the error in the data points is a normally distributed random variable, and so provides a conservative distance away from the boundary from which points in our likely free region can be generated.

mcd_l: Parameters mcd_l had the purpose of deciding when two points in the same, or distinct, likely free regions should be tested for connection. Recall that we did not want to connect points in likely free regions that lay far apart, ie. because of the risk that an unknown boundary to the C-surface may exist between them. Our aim was to connect points in the same likely free region, or points in distinct likely free regions lying close together or intersecting. Then, a value for mcd_l for a C-surface that we found worked well was 5 percent of the average size of likely free regions on the C-surface, ie. 0.05 times the average value of md for the likely-free regions on

the C-surface.

mcd_d: Parameter mcd_d was used to determine if two interior segments on a C-surface intersected. The value we used for this parameter was the maximum distance between any two sequential points in an interior segment on the C-surface. That is, even if interior segments happened to intersect where sequential points in one of them lay far apart, the method would still recognise that these interior segments did in fact intersect.

With a method for determining a value for each parameters in our approach, we can now present the results for deriving a set of $\mathbf{P}(t)$ for the spindle assembly task.

4.3.8 Results

We saw that our method can determine a $\mathbf{P}(t)$ between any two demonstrated points that exist on the boundary of a C-surface. Even for the small demonstration set D_1 to D_6, the many demonstrated points on boundaries in each state mean that many possible $\mathbf{P}(t)$ could be derived. In this section we choose to derive a subset of the possible $\mathbf{P}(t)$ that could be derived. We choose for this subset those $\mathbf{P}(t)$ required for experiments in the next chapter. That is, Chapter 5 has the purpose of determining a set of desired event paths σ that complete the task. We present in this section the results of deriving the $\mathbf{P}(t)$ required for robot execution of those paths.

Table 5.2 shows the event paths that will be selected in Chapter 5 for execution by the robot. We present here, in Table 4.1, the set of $\mathbf{P}(t)$ required in order to have the robot execute these paths. In total, 33 distinct $\mathbf{P}(t)$ required derivation, and for the purposes of simplified referencing, we have labelled in column 1 of the table each $\mathbf{P}(t)$ with a number from 1 to 33. We know that each $\mathbf{P}(t)$ will be applied in some state γ_w^λ. It will be applied in γ_w^λ directly after the occurrence of event τ_w^σ, with the aim of triggering in the assembly process the event τ_{w+1}^σ. We show in columns 2, 3 and 4 of Table 4.1 the γ_w^λ, τ_w^σ and τ_{w+1}^σ for each $\mathbf{P}(t)$. Note that γ_w^λ will correspond to one of the states $\gamma_i^\mathcal{A}$ in $\mathcal{A}$. The number

$\mathbf{P}(t)$	γ_w^λ	τ_w^σ	τ_{w+1}^σ	Demonstrated/ Undemonstrated	Noise Removed ?	Obstacle Free ?
1	2	start "event"	$(2,5)^{\mathcal{A}}$	Dem	no noise	Yes
2	5	$(2,5)^{\mathcal{A}}$	$(5,75)^{\mathcal{A}}$	Dem	Yes	Yes
3	75	$(5,75)^{\mathcal{A}}$	$(75,6)^{\mathcal{A}}$	Dem	no noise	Yes
4	6	$(75,6)^{\mathcal{A}}$	$(6,7)^{\mathcal{A}}$	Dem	no noise	Yes
5	7	$(6,7)^{\mathcal{A}}$	$(7,8)^{\mathcal{A}}$	Dem	No	Yes
6	8	$(7,8)^{\mathcal{A}}$	$(8,54)^{\mathcal{A}}$	Undem	n/a	Yes
7	54	$(8,54)^{\mathcal{A}}$	$(54,47)^{\mathcal{A}}$	Dem	no noise	Yes
8	47	$(54,47)^{\mathcal{A}}$	$(47,1)^{\mathcal{A}}$	Dem	no noise	Yes
9	2	start "event"	$(2,21)^{\mathcal{A}}$	Dem	Yes	Yes
10	21	$(2,21)^{\mathcal{A}}$	$(21,29)^{\mathcal{A}}$	Dem	Yes	Yes
11	29	$(21,29)^{\mathcal{A}}$	$(29,28)^{\mathcal{A}}$	Dem	no noise	Yes
12	28	$(29,28)^{\mathcal{A}}$	$(28,9)^{\mathcal{A}}$	Dem	no noise	Yes
13	9	$(28,9)^{\mathcal{A}}$	$(9,65)^{\mathcal{A}}$	Dem	No	Yes
14	65	$(9,65)^{\mathcal{A}}$	$(65,60)^{\mathcal{A}}$	Dem	no noise	Yes
15	55	$(60,55)^{\mathcal{A}}$	$(55,54)^{\mathcal{A}}$	Dem	no noise	Yes
16	54	$(55,54)^{\mathcal{A}}$	$(54,47)^{\mathcal{A}}$	Dem	Yes	Yes
17	21	$(2,21)^{\mathcal{A}}$	$(21,27)^{\mathcal{A}}$	Undem	n/a	Yes
18	27	$(21,27)^{\mathcal{A}}$	$(27,8)^{\mathcal{A}}$	Dem	no noise	Yes
19	8	$(27,8)^{\mathcal{A}}$	$(8,54)^{\mathcal{A}}$	Undem	n/a	Yes
20	8	$(27,8)^{\mathcal{A}}$	$(8,38)^{\mathcal{A}}$	Dem	Yes	Yes
21	38	$(8,38)^{\mathcal{A}}$	$(38,77)^{\mathcal{A}}$	Undem	n/a	Yes
22	77	$(38,77)^{\mathcal{A}}$	$(77,43)^{\mathcal{A}}$	Dem	No	Yes
23	43	$(77,43)^{\mathcal{A}}$	$(43,47)^{\mathcal{A}}$	Dem	no noise	Yes
24	47	$(43,47)^{\mathcal{A}}$	$(47,1)^{\mathcal{A}}$	Dem	no noise	Yes
25	21	$(2,21)^{\mathcal{A}}$	$(21,30)^{\mathcal{A}}$	Undem	n/a	Yes
26	30	$(21,30)^{\mathcal{A}}$	$(30,11)^{\mathcal{A}}$	Dem	no noise	Yes
27	11	$(30,11)^{\mathcal{A}}$	$(11,66)^{\mathcal{A}}$	Dem	No	Yes
28	66	$(11,66)^{\mathcal{A}}$	$(66,61)^{\mathcal{A}}$	Dem	no noise	Yes
29	61	$(66,61)^{\mathcal{A}}$	$(61,56)^{\mathcal{A}}$	Dem	no noise	Yes
30	56	$(61,56)^{\mathcal{A}}$	$(56,49)^{\mathcal{A}}$	Dem	No	Yes
31	49	$(56,49)^{\mathcal{A}}$	$(49,78)^{\mathcal{A}}$	Dem	no noise	Yes
32	79	$(78,79)^{\mathcal{A}}$	$(79,48)^{\mathcal{A}}$	Dem	no noise	Yes
33	47	$(48,47)^{\mathcal{A}}$	$(47,1)^{\mathcal{A}}$	Dem	no noise	Yes

Table 4.1: Results of our $\mathbf{P}(t)$ derivation method for the set of $\mathbf{P}(t)$ required for experiments in Chapter 5

shown in column 2 is the value of the index i for this $\gamma_i^{\mathcal{A}}$. For example, $\mathbf{P}(t)$ number 2 is the position control command applied in state 5, ie. state $\gamma_5^{\mathcal{A}}$ in $\mathcal{A}$. Similarly, τ_w^σ and τ_{w+1}^σ will each correspond to events in $\mathcal{A}$. Note how events in columns three and four of the table are the events in $\mathcal{A}$ corresponding to τ_w^σ and τ_{w+1}^σ.

Recall from the problem formulation section of this chapter that our method needed to derive two types of $\mathbf{P}(t)$: (a) between start and end points where a demonstrated path exists (eg. between points p_1 and p_2 in Figure 4.1), and (b) between start and end points where no direct demonstrated path exists (eg. between points p_1 and p_3 in Figure 4.1). Denote $\mathbf{P}(t)$ of type (a) as "demonstrated", and those of type (b) as "undemonstrated". We label in column 3 of Table 4.1 each $\mathbf{P}(t)$ with either "dem" or "undem" according to its type. Only five $\mathbf{P}(t)$ in total were undemonstrated, ie. $\mathbf{P}(t)$ numbers 6, 17, 19, 21, and 25. Note that all $\mathbf{P}(t)$ in Table 4.1 were derived to achieve a *demonstrated* event. A demonstrated event requires an undemonstrated $\mathbf{P}(t)$ when it is proceeded in the desired event path σ by an event that did not precede it in the demonstration set. For example, for $\mathbf{P}(t)$ number 6, both (7,8) and (8,54) were demonstrated events, but event (7,8) never directly preceded (8,54) in the demonstration set. We know that both these events occur at points in C-space lying on the boundary of C-surface $c_8^{\mathcal{A}}$. Then, since (7,8) never directly preceded (8,54), there will not exist a demonstrated path on $c_8^{\mathcal{A}}$ passing directly between these boundary points. In the paragraphs immediately following, we shall discuss the performance of our method for deriving demonstrated $\mathbf{P}(t)$. Following that, we will turn our attention to its performance for deriving undemonstrated $\mathbf{P}(t)$.

Demonstrated $\mathbf{P}(t)$: Two important questions exist for each demonstrated $\mathbf{P}(t)$ in Table 4.1. First, how much noise removal did it achieve, and second, did it really define an obstacle-free path? We show in column 6 of Table 4.1 whether noise removal occurred. Noise removal means that the derived path encoded a more direct route than the original, underlying demonstrated path. In some cases, demonstrated $\mathbf{P}(t)$ achieved significant noise removal compared to the demonstrated path, eg. $\mathbf{P}(t)$ numbers 9, 10, 2, 20 and 16.

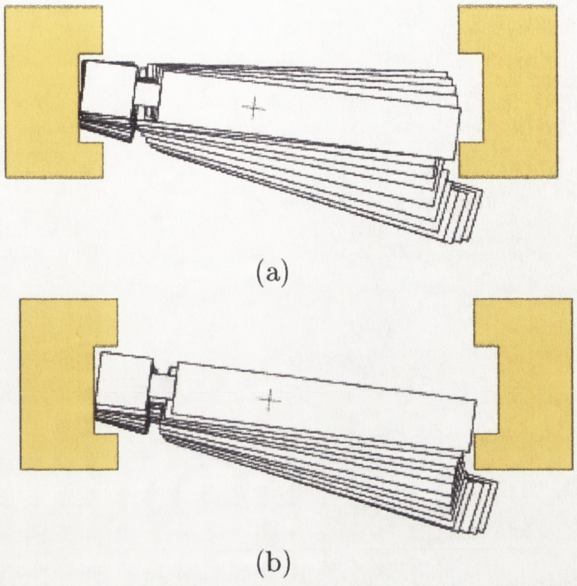
For these cases we have entered a "Yes" in column 6 of the table. Other demonstrated $\mathbf{P}(t)$ more-or-less followed exactly the demonstrated path. In some of these cases this was sensible, since the demonstrated path was noise free. We have labelled these cases in the table with "no noise", eg. $\mathbf{P}(t)$ numbers 1, 3, 4, 7, etc. In the remainder of cases, the demonstrated path did contain noise, however the derived $\mathbf{P}(t)$ still followed the demonstrated path. We have labelled these cases in column 6 of the table with a "No", eg. $\mathbf{P}(t)$ numbers 22, 27, 30, 5 and 13. Our main interest here is with those $\mathbf{P}(t)$ labelled in column 6 with a yes or a no (ie. did our method remove noise if it existed?) Notice how noise has been removed for $\mathbf{P}(t)$ derived in states that, (a) were visited often in the demonstration, and (b) have many boundary states [3]. States 2, 21, 8 and 54 each have properties (a) and (b). Notice then how the $\mathbf{P}(t)$ derived for noisy demonstrated paths in these states, ie. $\mathbf{P}(t)$ numbers 2, 9, 10, 16 and 21, have each had noise successfully removed. In contrast, Table 4.1 shows that noise was not generally removed for $\mathbf{P}(t)$ derived in states visited only once in the demonstration set, eg. states 30, 56, 77. Neither was it removed for states that were visited more than once, but that had only a small number of boundary states, eg. states 9 and 7 (state 9 has only two boundary states: states 28 and 65, while state 7 had no boundary states). The dependance for success by our method on properties (a) and (b) is a sensible outcome, since the C-surface of states with these properties will have more interior segments, and more likely free regions. Then the likely-hood of finding a $\mathbf{P}(t)$ that encodes a more direct route on the C-surface than the demonstrated path increases. That is, where our method has sufficient "information" on a C-surface, noise will be removed when deriving $\mathbf{P}(t)$. Otherwise, the method will revert to using the original demonstrated path, even if it contained noise.

The second important question regarding each demonstrated $\mathbf{P}(t)$ was whether it defined an obstacle free path. Remember that our method derives $\mathbf{P}(t)$ that are likely obstacle-free. We show in column 7 of Table 4.1 whether each $\mathbf{P}(t)$ was obstacle free. A key result in these experiments was that each demonstrated $\mathbf{P}(t)$ did in fact define an ob-

[3] Recall our definition of a *boundary state* in Section 4.3.3.

stacle free path. We have seen that three distinct type of demonstrated $\mathbf{P}(t)$ exist in Table 4.1. First, $\mathbf{P}(t)$ derived where no noise existed. This type of $\mathbf{P}(t)$ follows a demonstrated path, and we know that demonstrated paths are obstacle free. Second, $\mathbf{P}(t)$ derived in states that were visited often in the demonstration, and had many boundary states. It is to be expected that $\mathbf{P}(t)$ for this type of state are obstacle free. The presence of many boundary states means that likely-free-regions will generally cover the C-surface well in regions where we wish to generate $\mathbf{P}(t)$. For example, for state 8 we know that boundary segments exist in states 7, 27, 33, 9, 38 and 54, and we saw in Figure 4.5 how the likely free regions generated from these segments cover the C-surface of state 8 very well. Then for $\mathbf{P}(t)$ of this type, if the demonstrated path on which a $\mathbf{P}(t)$ is based does contain noise, the good coverage of points of the C-surface means that an alternate, "noise-free" path for $\mathbf{P}(t)$ will most likely be found. The final $\mathbf{P}(t)$ type in Table 4.1 are those where noise has not been removed. We noted how our method reverted to using the underlying demonstrated path for this type of $\mathbf{P}(t)$. That is, although these $\mathbf{P}(t)$ contain noise, they follow a demonstrated path, and will therefore be obstacle free.

We have seen how for demonstrated $\mathbf{P}(t)$ our method will remove noise if sufficient information on a C-surface exists. If sufficient information does not exist, then it reverts to using the underlying demonstrated path. This type of outcome is attractive since, as we have seen, it tends to produce $\mathbf{P}(t)$ that are obstacle free. An important influence on achieving this outcome was the conservative value we set for parameter md. Recall how we chose in Section 4.3.7 a metric that resulted in conservative values for md. Conservative values of md mean our method will only remove noise if sufficient information exists. The alternative approach of setting md aggressively may result in an increased number of demonstrated $\mathbf{P}(t)$ where noise is successfully removed, eg. noise free paths may be derived for $\mathbf{P}(t)$ numbers 5, 13, 18, etc. However, the risk with such an approach is that $\mathbf{P}(t)$ can then become non-obstacle-free, since points in likely free regions would be generated far from their boundary segments. We prefer, and recommend, the approach of setting md conservatively. In this way, the method has the tendency to produce $\mathbf{P}(t)$ that are



(a)

(b)

Figure 4.8: An example of the process showing (a) an original demonstrated path containing noise, and (b) the noise-free path that resulted

obstacle free, and will only remove noise in cases where it has sufficient information to do so.

We have talked in general about the results of our method for deriving demonstrated $\mathbf{P}(t)$. To provide the reader with a more concrete idea about how our method worked in practice, we now provide an example. $\mathbf{P}(t)$ number 20 was chosen as a good example of how our method removed noise from a demonstrated path. $\mathbf{P}(t)$ number 20 encodes a path in state 8 between states 27 and 38. We show in Figure 4.8(a) the demonstrated path on which this $\mathbf{P}(t)$ was based. Note how the demonstrated path contains significant noise in that it is overly long. The path resulted because the demonstrator became confused about the position of the spindle relative to the supports. He made initial contact between the spindle and right support in state 27, and his aim was then to reach state 65 (see Figure 2.6) by compressing the spindle and moving it around the front lip of the rebate in the right support. However, on not successfully finding state 65, his reaction was to retract the spindle back towards himself, resulting in state 38. The path derived for $\mathbf{P}(t)$ number 20 by our method is shown in Figure 4.8(b). Note how this path follows a much more

efficient route than the demonstrated path shown in Figure 4.8(a). In fact, the derived $\mathbf{P}(t)$ encodes a path that is 62 percent the length in C-space of the demonstrated path, ie. by using $\mathbf{P}(t)$ instead of the demonstrated path, noise can be avoided and an increase in robot task performance of 62 percent can be obtained.

Undemonstrated $\mathbf{P}(t)$: An important question regarding each demonstrated $\mathbf{P}(t)$ was: did it remove noise? Undemonstrated $\mathbf{P}(t)$ define a completely new path on a C-surface and so do not "remove noise" from a demonstrated path as such [4]. Then, we enter in column 6 of Table 4.1 for each undemonstrated $\mathbf{P}(t)$ a label "n/a". The key questions regarding undemonstrated $\mathbf{P}(t)$ are whether (i) a path can actually be found, and (ii) if one can be found, if it really is obstacle free. Question (i) is an issue because there does not exist a demonstrated path between the start and end points of the undemonstrated $\mathbf{P}(t)$ we wish to derive. That is, unlike for demonstrated $\mathbf{P}(t)$, we cannot fall-back on using the demonstrated path if insufficient information on the C-surface exists. Question (ii) is an issue for undemonstrated $\mathbf{P}(t)$ (as it was for demonstrated $\mathbf{P}(t)$) since our method derives commands using points in likely free regions, and these points are not guaranteed to be obstacle free.

Recall that Table 4.1 shows five undemonstrated $\mathbf{P}(t)$, ie. $\mathbf{P}(t)$ numbers 6, 17, 19, 21 and 25. It turned out that paths for each of these $\mathbf{P}(t)$ could be found, and that these paths did in reality define obstacle free paths. We show this fact by entering a "yes" for these $\mathbf{P}(t)$ in column 7 of Table 4.1. The main reason for this result was because these $\mathbf{P}(t)$ were derived in states visited often in the demonstration. For example, $\mathbf{P}(t)$ numbers 6, 17, 19 and 25 were derived in states 8 and 21. We have seen previously how there exist many likely free regions on the C-surfaces of these states. For example, we saw in Figure 4.5 some of the likely free regions for on the C-surface of state 8. We show in Figure 4.9 the path encoded by one of the undemonstrated $\mathbf{P}(t)$ derived in state 8. This path

[4] Although it will become clear in the following chapter how the derivation of undemonstrated $\mathbf{P}(t)$ is an important requirement of our method for removing noise at the *task-level*.

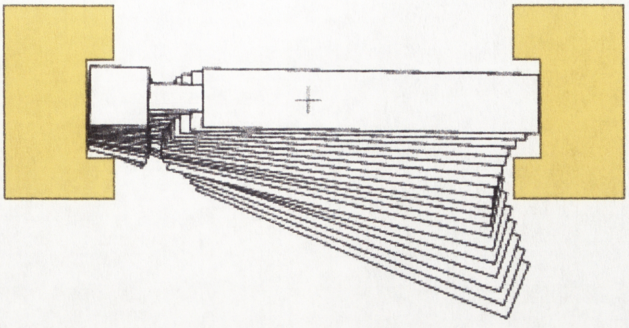


Figure 4.9: An undemonstrated $\mathbf{P}(t)$ derived to traverse in state 8 between states 7 and 54

is encoded by $\mathbf{P}(t)$ number 6, and passes through state 8 between states 7 and 54. This path visits diverse parts of c_8^A, and its derivation required the existence of all the likely free regions shown in Figure 4.5.

Recall our conservative approach to setting the value of parameter md. Such an approach risks our method not finding a path when deriving undemonstrated $\mathbf{P}(t)$ in states visited infrequently in the demonstration. This is the case since such states have few likely free regions, or interior segments on their C-surfaces. Then the start and end points of the desired undemonstrated $\mathbf{P}(t)$ may not be connected by intermediate points. If this is the case, a path for the desired undemonstrated $\mathbf{P}(t)$ may not be found. In such cases the value of md must be set large so that connection can be made. however the risk then is that $\mathbf{P}(t)$ encoding non-obstacle free paths result. This is limitation of our method. It is not suitable for deriving undemonstrated $\mathbf{P}(t)$ in infrequently visited states in the demonstration. For our experiments in this chapter it turned out that we did not need to derive such undemonstrated $\mathbf{P}(t)$. Recall how the $\mathbf{P}(t)$ derived in this chapter were those required by our method for producing a desired event path in the next chapter. We shall see in the next chapter how our method there, in general, will only require undemonstrated $\mathbf{P}(t)$ to be derivied in states visited often in the demonstration.

We have shown that our approach is a viable method for position control command synthesis. However, prior to moving onto the second problem in this chapter of deriving

a force control command $\mathbf{F}(t)$, we note the relatively small number of points shown in the $\mathbf{P}(t)$ of Figures 4.8 and 4.9. That is, from an implementation point of view, the $\mathbf{P}(t)$ derived by our method could not be used directly in a position control servo loop that will run at a speed in the order of 100-300 Hz. What is required are intermediate points to the ones generated by our method. We denote as *via points* each point in a $\mathbf{P}(t)$ that is derived by our method. We note that via points can be of two types, those originating from interior segments, and those originating from likely free regions. Then between via points originating from the same interior segment on a C-surface, we use as intermediate points the points recorded in the demonstration. Between via points originating from likely free regions, or originating from distinct interior segments, we adopt the well known technique of fitting a cubic spline for generating intermediate points (as reported in [53]). Up until now we have referred to a sequence of via points as $\mathbf{P}(t)$. From now on we shall use $\mathbf{P}(t)$ to denote the combined sequence of via points and intermediate points, as determined by the methods just stated. With this part of our $\mathbf{P}(t)$ synthesis method clarified, we are now in a position to move on to the second problem in this chapter: deriving force control commands.

4.4 Force Control-Command Synthesis

The previous section presented a method for determining the position control part $\mathbf{P}(t)$ of the control command $\mathbf{u}(t)$. Recall from Section 4.2 how the second part of $\mathbf{u}(t)$ consisted of a force control command $\mathbf{F}(t)$. Our aim in this section is to present a method that will derive an $\mathbf{F}(t)$ for each $\mathbf{P}(t)$ presented in the previous section. We noted in the closing part of the previous section that a $\mathbf{P}(t)$ will contain via points of two types, (i) those originating from interior segments, and (ii) those originating from likely free regions. We note then that, for the purposes of force command synthesis, $\mathbf{P}(t)$ can be divided into segments of three types. To help make our presentation of these types clearer, we present in Figure 4.10 a simple example of a $\mathbf{P}(t)$ that could have been derived in the previous section.

The figure shows a $\mathbf{P}(t)$ traversing between two boundaries on a C-surface. Via points in the $\mathbf{P}(t)$ are shown in the figure with crosses, circles and asterisks. Crosses represent via points in $\mathbf{P}(t)$ originating from likely free regions, Circles and asterisks represent via points originating from interior segments, ie. circles represent via points originating from one interior segment on our C-surface, while asterisks represent via points from another, distinct interior segment.

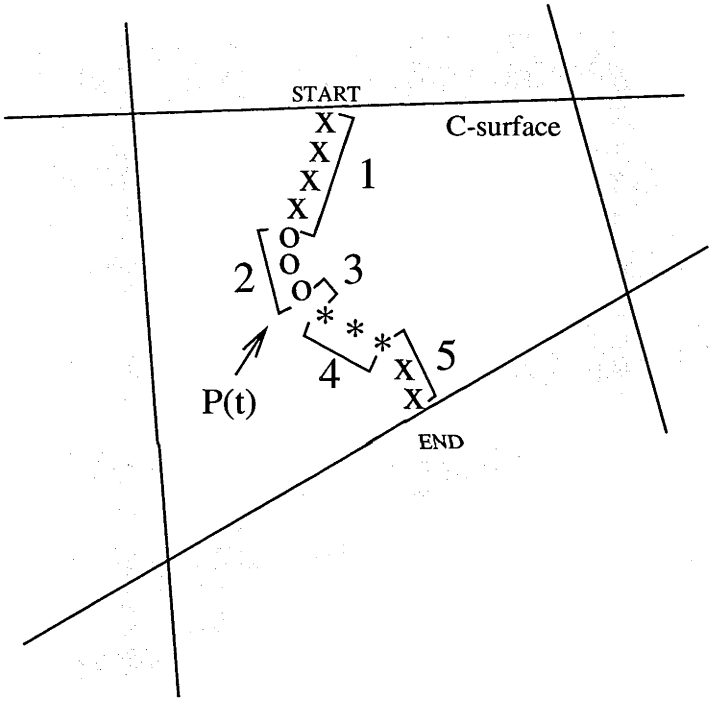


Figure 4.10: An example of how $\mathbf{P}(t)$ can be divided into segments of distinct types

Figure 4.10 shows how a $\mathbf{P}(t)$ can be split into segments of three distinct types: (a) segments containing points originating only from interior segments, eg. segments in Figure 4.10 labelled 2 and 4, (b) segments containing points originating only from likely free regions, eg. segments labelled 1 and 5, and (c) segments existing in $\mathbf{P}(t)$ between two points originating from distinct interior segments, eg. the segment in Figure 4.10 labelled 3. For segments of type (a) we have a demonstrated force sequence on which to base a

force command. We note that in general it is sub-optimal to directly use the demonstrated force as a force command for the robot, for reasons we state below. For segments in $\mathbf{P}(t)$ of type (b) and (c) we do not have a demonstrated force sequence on which to base our force command. That is, for segments of these types, we must synthesise a force command from scratch. Hence the content of this section divides neatly into two parts. That is:

- force command synthesis by improving demonstrated force commands
- force command synthesis from scratch

4.4.1 Force Command Synthesis based on Demonstrated Force

Let $\mathbf{p}$ be a point in a segment of type (a) in $\mathbf{P}(t)$. Denote as $\breve{\mathbf{f}}_p$ the force vector recorded in the demonstration at $\mathbf{p}$. Then our aim in this section is to determine a "clean" force command $\mathbf{f}_p$ from $\breve{\mathbf{f}}_p$. There are two reasons why we do not use $\breve{\mathbf{f}}_p$ as $\mathbf{f}_p$ directly. The first is because force sensors introduce noise into $\breve{\mathbf{f}}_p$ during the demonstration recording process. Force sensors are notorious for introducing a high frequency noise component into recorded signals. We do not wish to include the noise component of $\breve{\mathbf{f}}_p$ in $\mathbf{f}_p$. Second, $\breve{\mathbf{f}}_p$ will in general contain a friction component. Its friction component lies in the tangent hyperplane to our C-surface c_w^λ at $\mathbf{p}$. We saw in Section 4.2 that $\mathbf{f}_p$ should lie in the dual space of the tangent space. That is, we need to remove the friction component from $\breve{\mathbf{f}}_p$ in order to produce $\mathbf{f}_p$. The details of how we produce $\mathbf{f}_p$ from $\breve{\mathbf{f}}_p$ are now presented.

Denote as $\mathcal{F}_p$ the dual space of the tangent space of c_w^λ at $\mathbf{p}$. Then we remove the friction component from $\breve{\mathbf{f}}_p$ by orthogonally projecting $\breve{\mathbf{f}}_p$ onto $\mathcal{F}_p$. Recall that in the previous chapter we derived for every possible c_w^λ a set of primitive C-surfaces whose intersection defines c_w^λ. Assume that γ_w^λ in λ corresponds to γ_i^A in A, and denote as $\phi_{\rho_{ij}^A}$ the equation determined in the previous chapter for the j^{th} primitive C-surface of γ_w^λ [5] Then we note that a basis S of $\mathcal{F}_p$ is given by:

[5] Recall that we used j to index distinct constraints in a state. Then we also use j to index primitive C-surfaces because there exists one primitive C-surface for each distinct constraint present in γ_w^λ.

$$S = \{\nabla\phi_{\rho_{i1}^A}(\mathbf{p}), \nabla\phi_{\rho_{i2}^A}(\mathbf{p}), \ldots, \nabla\phi_{\rho_{ij}^A}(\mathbf{p}), \ldots\} \tag{4.6}$$

where ∇ is the gradient function [87], and where $\nabla\phi_{\rho_{ij}^A}(\mathbf{p})$ denotes function $\nabla\phi_{\rho_{ij}^A}$ evaluated at point $\mathbf{p}$. Basis S can be converted into an orthonormal basis $\acute{S}$ using the *Gram-Schmidt Process* [38], where we denote the j^{th} element of $\acute{S}$ simply as s_j. That is:

$$\acute{S} = \{s_1, s_2, \ldots, s_j, \ldots\} \tag{4.7}$$

Then a vector ${}^{prj}\breve{\mathbf{f}}_p$ equal to the projection of the demonstrated force $\breve{\mathbf{f}}_p$ onto the force control space $\mathcal{F}_p$ is calculated as:

$${}^{prj}\breve{\mathbf{f}}_p = \langle\breve{\mathbf{f}}_p, s_1\rangle s_1 + \langle\breve{\mathbf{f}}_p, s_2\rangle s_2 + \ldots + \langle\breve{\mathbf{f}}_p, s_j\rangle s_j + \ldots \tag{4.8}$$

where ${}^{prj}\breve{\mathbf{f}}_p$ is the projected force vector, and where $\langle\breve{\mathbf{f}}_p, s_j\rangle$ denotes the Euclidean inner product between vectors $\breve{\mathbf{f}}_p$ and s_j. Vector ${}^{prj}\breve{\mathbf{f}}_p$ gives a force command at $\mathbf{p}$ of correct direction, however the magnitude of ${}^{prj}\breve{\mathbf{f}}_p$ will still contain a noise component. That is, ${}^{prj}\breve{\mathbf{f}}_p$ determines the direction of our final force command $\mathbf{f}_p$, but that we must remove noise from the magnitude of ${}^{prj}\breve{\mathbf{f}}_p$ in order to produce a final $\mathbf{f}_p$.

Recall that in this section, point $\mathbf{p}$ originates from a segment in $\mathbf{P}(t)$ of type (a), ie. a segment whose points all originate from a single interior segment on our C-surface. Clearly there can be many such segments in a $\mathbf{P}(t)$. Denote as η one such segment in $\mathbf{P}(t)$. We know a force sequence will have been recorded from the demonstration for η. We have just presented a method for converting each force in this sequence to one that is orthogonal to our C-surface. Denote this "orthogonalised" force sequence as ${}^{prj}F^{\eta}(t)$. We have noted that the magnitude of each force in ${}^{prj}F^{\eta}(t)$ will still contain a noise component after the projection process, ie. ${}^{prj}F^{\eta}(t)$ will not in general define a smooth and continuous signal devoid of spikes and abhorrations. However, a good estimate of the noise free magnitude of each force in ${}^{prj}F^{\eta}(t)$ can be obtained by applying smoothing techniques. Many smoothing

techniques exist in the literature [33], however we found that a simple window averaging scheme with a number of repeats provided a smooth and continuous final force command suitable for that part of $\mathbf{P}(t)$ that is η.

4.4.2 Force Command Synthesis without Demonstrated Force

In contrast to segments of type (a), points in segments of type (b) and (c) were not demonstrated, and therefore have no demonstrated force on which to base a force command. Assume now that $\mathbf{p}$ is a point in $\mathbf{P}(t)$ that originated from a segments of type (b) or (c). Then we know a force command $\mathbf{f}_p$ at $\mathbf{p}$ must lie orthogonal to our C-surface c_w^λ (ie. the direction of $\mathbf{f}_p$ is known), however the magnitude of $\mathbf{f}_p$ is unknown. We determine an appropriate magnitude for $\mathbf{f}_p$ as follows.

In Section 4.4.1 we denoted as η a segment in $\mathbf{P}(t)$ of type (a). For this section, let η now denote a segment in $\mathbf{P}(t)$ of type (b) or (c). Assume η is a sequence containing n_l points $\mathbf{p}$, and denote a $\mathbf{p}$ in η generally as $\mathbf{p}_l^\eta$, ie. η is the sequence $\mathbf{p}_1^\eta, \ldots \mathbf{p}_l^\eta, \ldots \mathbf{p}_{n_l}^\eta$. To each side of η in $\mathbf{P}(t)$ there will exist a point originating from an interior segment [6]. Denote as $\mathbf{p}_0^\eta$ and $\mathbf{p}_{n_l+1}^\eta$ the interior segment points existing on each side of η in $\mathbf{P}(t)$. We derived noise-free force commands for $\mathbf{p}_0^\eta$ and $\mathbf{p}_{n_l+1}^\eta$ in Section 4.4.1. Let the force commands derived for $\mathbf{p}_0^\eta$ and $\mathbf{p}_{n_l+1}^\eta$ be the vectors $\mathbf{f}_0^\eta$ and $\mathbf{f}_{n_l+1}^\eta$. Then to determine an appropriate magnitude for the force command $\mathbf{f}_l^\eta$ at $\mathbf{p}_l^\eta$, we interpolate between the known magnitudes of $\mathbf{f}_0^\eta$ and $\mathbf{f}_{n_l+1}^\eta$. Let f_0^η and $f_{n_l+1}^\eta$ denote the magnitudes of $\mathbf{f}_0^\eta$ and $\mathbf{f}_{n_l+1}^\eta$ respectively. Then, as a first step, we calculate the length L_l along η between $\mathbf{p}_0^\eta$ and $\mathbf{p}_l^\eta$ as:

$$L_l = \sum_{i=0}^{l} \| \mathbf{p}_{i+1}^\eta - \mathbf{p}_i^\eta \| \tag{4.9}$$

where $\| \mathbf{p}_{i+1}^\eta - \mathbf{p}_i^\eta \|$ denotes the distance between the two sequential points in η, $\mathbf{p}_i^\eta$ and

[6] For segments of type (c) this will be by definition true. For segments of type (b), it will be true since we only ever generate a $\mathbf{P}(t)$ between demonstrated start and end points lying on the boundary of our C-surface.

$\mathbf{p}_{i+1}^{\eta}$. We determine the magnitude of the force command at $\mathbf{p}_l^{\eta}$ by using the well known interpolation technique of fitting a cubic spline [53]. Denote the distance rate of change of force magnitude along η at $\mathbf{f}_0^{\eta}$ as $\dot{f}_0^{\eta}$ and at $\mathbf{f}_{n_l+1}^{\eta}$ as $\dot{f}_{n_l+1}^{\eta}$. Then our requirements on the spline are that its initial and final magnitudes are f_0^{η} and $f_{n_l+1}^{\eta}$, and that it has initial and final derivatives of magnitude $\dot{f}_0^{\eta}$ and $\dot{f}_{n_l+1}^{\eta}$. To find a cubic spline that satisfies such requirements we must solve the set of differential equations:

$$\begin{aligned} f_0^{\eta} &= a_0 \\ f_{n_l+1}^{\eta} &= a_0 + a_1 L + a_2 L^2 + a_3 L^3 \\ \dot{f}_0^{\eta} &= a_1 \\ \dot{f}_{n_l+1}^{\eta} &= a_1 + 2a_2 L + 3a_3 L^2 \end{aligned} \tag{4.10}$$

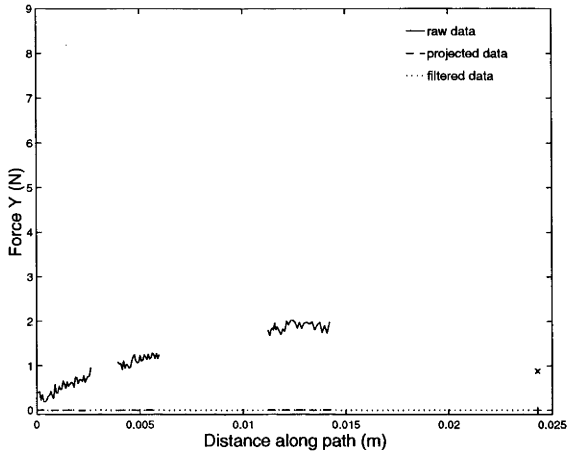
for the parameters a_0, a_1, a_2 and a_3 of the spline, where L denotes the distance along η between $\mathbf{p}_0^{\eta}$ and $\mathbf{p}_{n_l+1}^{\eta}$ (as per equation (4.9)). Solving the differential equations (4.10) for a_0, a_1, a_2 and a_3 gives the solution:

$$\begin{aligned} a_0 &= f_0^{\eta} \\ a_1 &= \dot{f}_0^{\eta} \\ a_2 &= \frac{3}{L^2}(f_{n_l+1}^{\eta} - f_0^{\eta}) - \frac{2}{L}\dot{f}_0^{\eta} - \frac{1}{L}\dot{f}_{n_l+1}^{\eta} \\ a_3 &= \frac{-2}{L^3}(f_{n_l+1}^{\eta} - f_0^{\eta}) - \frac{1}{L^2}(f_{n_l+1}^{\eta} + f_0^{\eta}) \end{aligned} \tag{4.11}$$

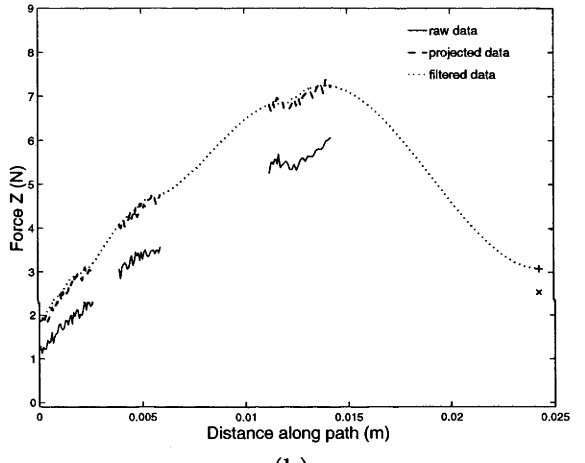
The magnitude of our force command $\mathbf{f}_l^{\eta}$ at $\mathbf{p}_l^{\eta}$ in η is then calculated simply as:

$$f_l^{\eta} = a_0 + a_1 L_l + a_2 L_l^2 + a_3 L_l^3 \tag{4.12}$$

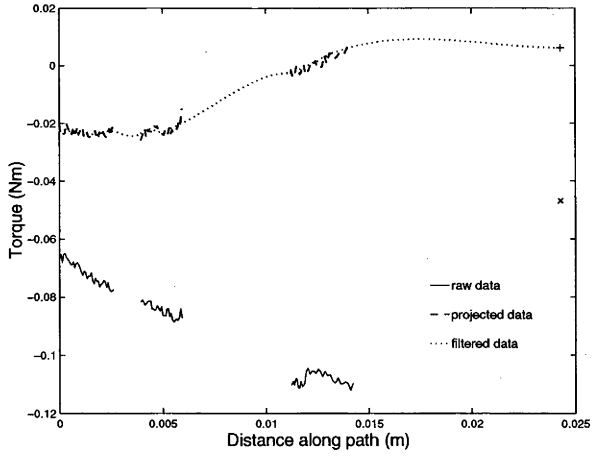
where recall that L_l was calculated according to equation (4.9).



(a)



(b)



(c)

Figure 4.11: Force signals produced by each step of the $\mathbf{F}(t)$ derivation process for $\mathbf{P}(t)$ number 20

4.4.3 Results

Our force command derivation method was applied to produce an $\mathbf{F}(t)$ for each $\mathbf{P}(t)$ derived in Section 4.3.8 (ie. in Table 4.1). The result was a smooth and continuous force command for use in experiments presented in the next chapter. To provide the reader with an idea of how the method worked in practice, we present details of the $\mathbf{F}(t)$ derivation process for $\mathbf{P}(t)$ number 20 in Table 4.1. Recall that $\mathbf{P}(t)$ number 20 was a position control command derived to traverse state 8 between states 27 and 38, and how we showed the path traced-out in the physical workspace by this command in Figure 4.8(b). In total, 18 distinct configurations are shown in Figure 4.8(b), ie. this $\mathbf{P}(t)$ contained 18 distinct points. Of interest here is the make-up of points in $\mathbf{P}(t)$ number 20, ie. which points originated from interior segments, and which points originated from likely free regions. We note that point sequences 1-2, 4-5, 9-10 and 18 originated from interior segments, ie. these sequences define η of type (a) in our $\mathbf{P}(t)$. The remainder of points in this $\mathbf{P}(t)$ form segments of type (b) and (c).

We show in Figure 4.11 the results of $\mathbf{F}(t)$ derivation for the $\mathbf{P}(t)$ number 20. Three plots are shown, ie. (a), (b), and (c): one for each direction of control. Plots (a) and (b) show forces in the y and z-directions respectively, while plot (c) shows torque. The solid line in each plot shows the original demonstrated force sequence for each η of type (a) in $\mathbf{P}(t)$. Notice how these force signals contain noise. Recall in Section 4.4.1 how we produced a projected force signal by projecting each force in a demonstrated force sequence onto the dual space of the tangent space. The dashed line in each plot represents a projected force signal. One of the main aims of the projection step was to remove the friction component from demonstration force data. Notice then how the method has determined that all demonstrated force in the y-direction was due to friction between the spindle head and left support. That is, the projected force signal in the y-direction for each η of type (a) has been set to zero. Clearly only a force in the positive z direction, and a torque, should be commanded for configurations in this $\mathbf{P}(t)$, and this is reflected

in Figure 4.11. The final steps in the force command derivation process were to smooth projected force sequences, and to interpolate between them in order to produce a final force command $\mathbf{F}(t)$. The dotted line in each plot shows the force signal resulting from these final two steps. Notice how the noise in each projected force sequence has been removed by smoothing. For example, overlying each projected force sequence in plot (b) is a line representing the smoothed force sequence. Compared to the projected force sequence, notice how the smoothed force sequence forms a smooth and continuous signal devoid of spikes and/or high frequency noise. The spline interpolation stage had the purpose of determining a magnitude for $\mathbf{F}(t)$ where undemonstrated points existed in $\mathbf{P}(t)$ (ie. for η of type (b) and (c)). Figure 4.11 shows how our final $\mathbf{F}(t)$ is formed by fitting splines between the smoothed force sequences of each η of type (a) in $\mathbf{P}(t)$.

One aspect of Figure 4.11 that requires further explanation regards via point 18 (ie. the final point) in $\mathbf{P}(t)$. Note how this η contains only one via point, and so only a single demonstrated force/torque value is available in the force command synthesis process (this force/torque is marked as an "x" in each plot). The projection step is still possible as per normal (the projected force for this η is marked as a "+" in each plot), however we cannot apply the smoothing step in the process, ie. the "smoothed" value of force for this point is set as the projected value. In addition, the spline interpolation step of the process is complicated because a derivative of the spline at this point is not known. As can be seen in the figure, our solution to this dilemma was to specify the derivative for the spline at this point as zero. Such an approach is suitable when an η of this type occurs as the final point in $\mathbf{P}(t)$. However, the approach may not be suitable for such η when they occur in the middle of $\mathbf{P}(t)$. Our approach in these cases was simply to take as the derivative for the spline, the average value of the derivatives calculated for the η's of type (b) previous, and following, our "single point" η. Such an approach allowed $\mathbf{F}(t)$ to pass through the force value recorded for our single point η, while still resulting in an overall smooth and continuous force trajectory.

4.5 Conclusion

This chapter has been about deriving noise-free, low-level control commands for the robot from demonstration. We presented a method that derives noise-free position and force control commands for output to a hybrid force-position continuous controller. The first part of the chapter addressed position control-command synthesis. We used HDS modeling to convert the problem from finding a noise free position control command for the entire task into finding one on the C-surface of each state in a desired state path. Our approach identified two types of obstacle free points on any C-surface. First, points that were visited in the demonstration, and second, points in likely free regions. We represented the connectivity of these points using a connectivity graph. A noise-free position control command on the C-surface was then obtained by searching for the minimum cost path in the graph. We noted three major advantages of this approach compared to previous approaches: (i) that it made no base assumptions about the location or topology of demonstrated paths (ie. the start and end points of demonstrations did not have to coincide), (ii) that paths could contain undemonstrated points, and (iii) that it allowed paths to be derived between demonstrated points that were not directly connected by a demonstrated path. A weakness of the method was that it cannot guarantee that a non-obstacle-free path will be derived. However, experimental results showed that the method had the tendency to derive only obstacle-free paths if it was tuned appropriately. For demonstrated $\mathbf{P}(t)$, the method would revert to using the demonstrated path when insufficient information on a C-surface existed for it to derive a noise-free path. For undemonstrated $\mathbf{P}(t)$, the method could derive a path, so long as it was on a C-surface where substantial information existed, ie. one that was well visited in the demonstration. We noted as a feature of our method in the next chapter, the tendency to only require undemonstrated $\mathbf{P}(t)$ on well visited C-surfaces.

The second part of this chapter was about deriving noise-free force control commands. Force commands consisted of a sequence of force vectors, one for each point in the posi-

tion control command. The derivation of two types of force vector were identified: those required for points on a C-surface that were demonstrated, and those required for undemonstrated points. A direction for the force command in both cases was obtained using the result by De Schutter that it must lie in the dual space of the tangent space to our position on the C-surface. Determining a noise-free magnitude for force commands of demonstrated points was achieved by applying a simple smoothing technique. A magnitude for force commands of undemonstrated points was determined by fitting a cubic spline between the known magnitude of smoothed force commands for demonstrated points. Final derived force commands were found to provide a smooth and continuous signal that in experiments resulted in the robot achieving motion between parts in contact without chattering, and without accidently losing the desired contact.

Chapter 5

Selecting Optimal Task-Level Strategies from Demonstration

5.1 Introduction

Until now, work addressing the problem of demonstration sub-optimality in PbD has focussed on optimising low-level control commands. An important, additional aspect of this problem regards selecting an appropriate *task-level strategy* for use by the robot. There will usually be a number of task-level strategies demonstrated by the human. Some of these may be more optimal for the robot to execute than others. Then, a good idea is to select demonstrated strategies that will see the robot best perform the task. In this chapter we present a method to achieve exactly this end. We identify a number of different strategies that were demonstrated by the human in the spindle-assembly task. Robot performance of the task was found to range in its optimality, depending on what demonstrated strategies it adopted. Further, different strategies were found to result in task performance with optimality in different areas, eg. some resulted in reliable task execution, others in fast execution, etc. The method we will present allows the selection of the most optimal demonstrated strategy for use by the robot. In addition, it allows the demonstrator to specify what aspect of optimality is most important during the selection

process, ie. the demonstrator can choose as more important, reliable execution over fast execution, etc.

The chapter is set-out as follows. In Section 5.2 we formulate more exactly the problem to be solved. We show how a task-level strategy can be described as a sequence of discrete events in our HDS skill model. This description leads us to formulate the problem to be solved as finding a desired event path σ that results in optimal performance of the task. In Section 5.3 our solution to the problem is presented. Robot performance is broken down into four distinct areas. A desired event path σ is selected as the one predicted to result in the best robot performance of the task in the four performance areas on average. In addition, we show how desired event paths can be selected that have the robot perform optimally in each of the four individual performance areas. Section 5.4 presents the results of applying our method on the demonstration set D_1 to D_6 of the spindle-assembly task. In Section 5.4.1 we present the paths selected by our method. We discuss why each path was selected, and whether its selection was appropriate. In Section 5.4.2 we implement selected paths on the robot. The purpose of this section is to show that selected paths do in reality cause the robot to perform as predicted by our method. Finally, we end the chapter with Section 5.5, where we state our conclusions for this work.

5.2 Problem Formulation

We have introduced the idea of a *task-level strategy*, and how different task-level strategies can be used to complete a task. However, we have not yet clarified exactly what we mean by a task-level strategy. Much of this section is about defining this term more precisely. As a first step in this process, we present as an example, the set of task-level strategies demonstrated by the human in the spindle-assembly task.

The demonstration set D_1 to D_6 contains a number of different human-demonstrated strategies. We note that two main phases of the spindle-assembly task exist, (i) spindle head insertion into the left support, and (ii) spindle body insertion into the right support.

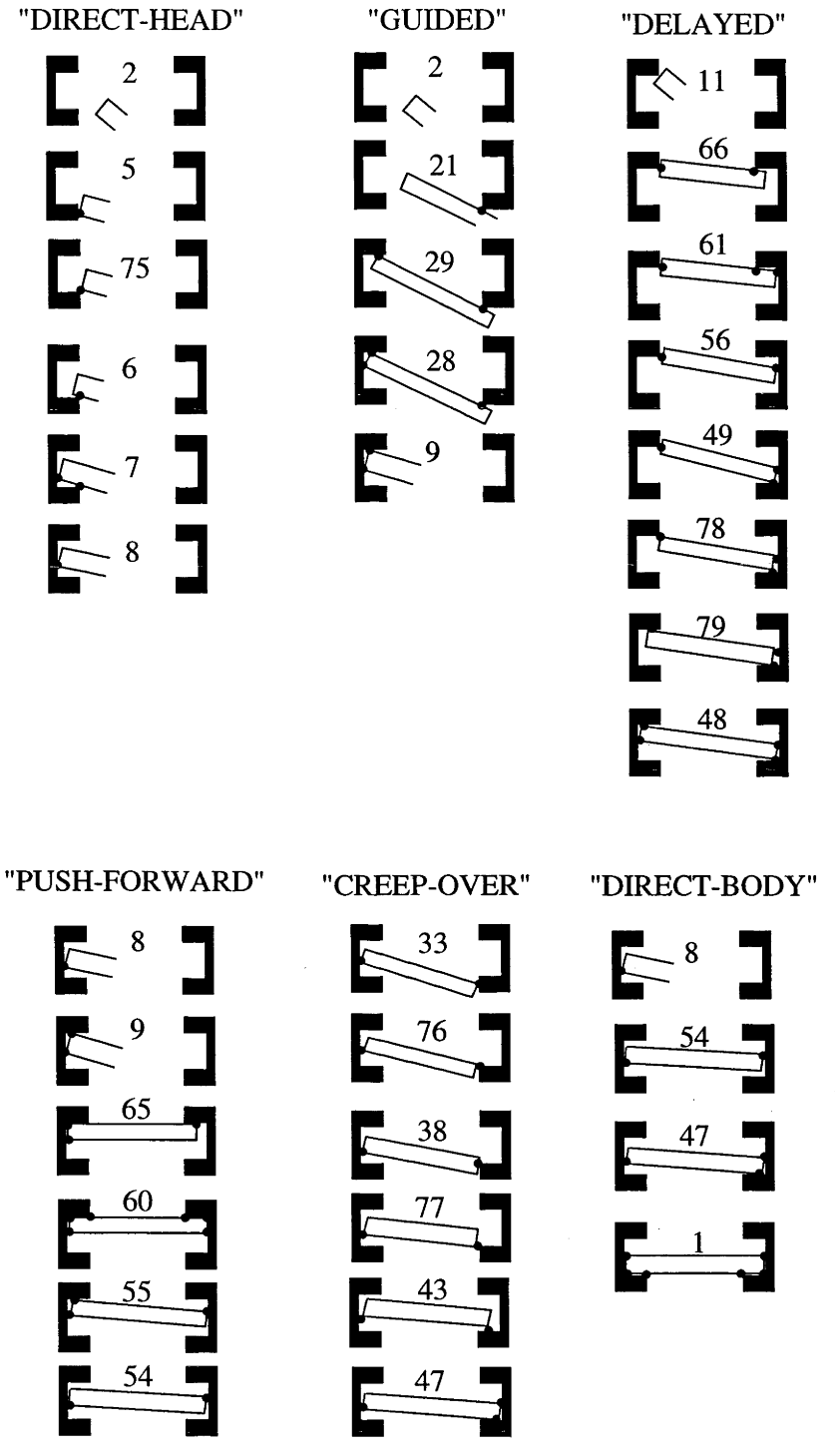


Table 5.1: The task-level execution strategies demonstrated for the spindle-assembly task

We group the strategies used by the demonstrator in each of these phases into categories in Table 5.1. The demonstrator has used three main strategies to insert the spindle head into the left support. First he has taken the "direct-head" approach, where the spindle head is moved directly towards the rebate in the left support, ie. the state sequence 5-75-6-7 more-or-less allows the spindle to move in a straight line between the spindle's start configuration and its inserted configuration in state 8. Second, he has demonstrated a "guided" approach. Here the contact in state 21 is used as a guide to achieve spindle head insertion. Since the demonstrator was blindfolded, the contact in state 21 helped him to correctly aim the spindle head toward the rebate in the right support. Third, a "delayed" approach was demonstrated. Here, contact between the spindle head and left support was maintained in state 11 while the spindle body was inserted into the rebate of the right support. That is, the final insertion of the spindle head into the rebate of the left support was *delayed* until the spindle body was fully inserted into the right support.

Table 5.1 also shows the strategies used by the demonstrator to insert the spindle body into the right support. Note that at this stage in the task the spindle head is in contact with the left support, however the blind-folded demonstrator is not certain of spindle head's position relative to the right support until contact between the two is made. Table 5.1 classifies into three main types the strategies used by the demonstrator to make this contact. First, he has used the "push-forward" approach, where contact is first made in state 60. Using this strategy, the demonstrator pushes the spindle body forward, pivoting about a contact between the spindle-head and left support until state 60 is achieved. Second, he has used a "creep-over" approach. Here, contact between the spindle body and right support is first made in state 33. Insertion of the spindle body into the rebate is then achieved by "creeping-over" the front lip of the right support (ie. through states 76, 38, 77, and 43) and into the rebate. Third and finally, the demonstrator has used the "direct-body" approach. Contact is made by moving the spindle body directly toward the base of the rebate in the right support, ie. without touching either side of the rebate.

The preceding discussion shows how many distinct task-level strategies can be demon-

strated by the human. Our problem in this chapter is to find a method for selecting the most appropriate demonstrated strategy for use by the robot. A key issue in solving this problem regards finding a precise description of a task-level strategy. An appropriate approach in this regard is suggested by our discussion above of strategies used in the spindle-assembly task. Notice how we have described each strategy in Table 5.1 as a sequence of discrete states. Each state sequence defines a sequence of events that must occur in the task for the particular strategy to be achieved. Then, an obvious description of a task-level strategy, and the one we shall use, is as a sequence of discrete events. For example, the "direct-head" assembly approach shown in Table 5.1 can be described as the sequence of events (2,5), (5,7), etc [1]. With a precise description of a task-level strategy in place, our problem of finding the best demonstrated strategies for the robot can be stated more concretely. That is, we must find a sequence of discrete events that will have the robot perform the task in an optimal fashion. Recall from Chapter 2 that finding such a sequence is exactly the role of the Event Path Planner (EPP) component in our HDS model. We denoted at that time such a sequence as σ, and called it the *desired event path.* Then our problem in this chapter is really to synthesise a functional EPP from demonstration. That is, given a demonstration or set of demonstrations for a task, the EPP we derive must determine a σ that results in optimal robot performance of the task. The remainder of this chapter is devoted to presenting a solution to this problem.

5.3 Selecting a Desired Event Path

Recall from Figure 2.3 the components besides the EPP in the robotic system, ie. the PM, DEC, CC and manipulator. Having the EPP select a desired event path means predicting the performance of each of these components for every event in the demonstration set. That is, we should select for inclusion in σ, demonstrated events that will be performed

[1] Recall from Chapter 2 how an event τ_k^A in $\mathcal{A}$ can be described as an ordered pair (i_1, i_2), where $\gamma_{i_1}^A$ and $\gamma_{i_2}^A$ are states in $\mathcal{A}$.

most optimally by these components. Note that we must *predict* the performance of demonstrated events by these components because at this stage in the PbD process we have no experience of how the robot will actually perform the task. A demonstration has been provided, and we are now trying to decide on a task-level strategy for use by the robot. In Section 5.3.2, a framework for predicting robot performance of demonstrated events is presented. However, prior to our discussion there, we first present in the next section, a definition of exactly what we mean by optimal robot performance.

5.3.1 Defining Optimal Robot Performance

In order to find the event path that will result in optimal robot performance of the task, the concept of optimal performance must first be defined. To this end, we define optimal robot performance as performance with the following characteristics:

1. low execution time

2. high reliability

3. low control effort

Our aim is to select events for σ that are performed by the robot with these three characteristics. However in addition, we note the capability of the human to analyze his performance of the task. A human will often identify and repeat often in the demonstration any strategies for task execution that he finds works well. We assume that strategies found to work well by the human will also work well for the robot, ie. that copying what the demonstrator did often can lead to more optimal robot performance of the task. So, in addition to selecting events that see the robot perform with characteristics 1.,2., and 3., our aim is also to select events for σ that were:

4. demonstrated often

There are also two important, additional reasons why copying the demonstrator is a good approach when selecting task level strategies for the robot. First, recall from

Chapter 3 how we concluded that our method for deriving C-space was more accurate in describing regions of C-space visited often in the demonstration. That is, we would prefer in this chapter to derive a task level strategy that traverses regions in C-space visted more often in the demonstration. Second, recall from Chapter 4 how our DEC method had greater possibilities for noise removal in regions of C-space that were visited often in the demonstration, ie. again, selecting a task level strategy traversing regions of C-space visited often in the demonstration will be beneficial. With optimal performance defined, we can now present a method for predicting the level of robot performance of demonstrated events in each of these performance areas.

5.3.2 The Path Selection Framework

Recall that each arc in automaton $\mathcal{A}_D$ represents an event that was demonstrated. We limit our method in this chapter to finding a σ that contains only demonstrated events, ie. each event in σ will be represented by one of the arcs in $\mathcal{A}_D$. Our idea for determining σ is to assign to each arc in $\mathcal{A}_D$ a cost, where the cost assigned reflects how well the robot will perform the event represented by that arc. A high cost is assigned to the arcs of events predicted to result in poor performance, while a low cost is given to the arcs of events predicted to be performed well. We then determine σ by conducting a search in $\mathcal{A}_D$ for the minimum cost path between the start and goal states in the task. Since a minimum cost path is selected, only the events resulting in the best possible robot performance will be included in σ.

The obvious question with the approach regards how the cost of each arc should be determined. Remember, the value of the cost we assign is our prediction of how the components in the robotic system will, as a whole, perform the event represented by that arc. Recall that τ_k^A denotes generally one of the events in the task. For the purposes of our discussion in this chapter, let τ_k^A be an event that was demonstrated, and let (i_1, i_2) be the ordered pair denoting this event, ie. that τ_k^A is the event triggering a transition in the task from state $\gamma_{i_1}^A$ to state $\gamma_{i_2}^A$. We assign a cost to the arc of τ_k^A as follows. First,

a cost is assigned individually in each of the four performance areas: time, reliability, control effort, and number of times demonstrated. Sections 5.3.2(a), 5.3.2(b), 5.3.2(c) and 5.3.2(d) describe for each performance area how this step is achieved. An overall cost for the arc is then calculated from the costs assigned in the individual performance areas. Section 5.3.2(e) describes the details of how the overall cost is calculated.

(a) Time

Strictly speaking, an event in the HDS is instantaneous and takes zero time. It is the infinitesimal amount of time taken to pass between two distinct contact formations in the task. For our work here, we are interested in how long an event takes to *achieve*. That is, how long after the previous event in the assembly process does it take for us to have event τ_k^A occur. Three components to this time exist:

1. *time for discrete event control in* $\gamma_{i_1}^A$: how long for the DEC to determine a control command $u(t)$ for use in state $\gamma_{i_1}^A$ that will cause event τ_k^A to occur
2. *time for continuous control in* $\gamma_{i_1}^A$: time for the CC/manipulator to traverse $u(t)$
3. *time to process monitor* τ_k^A: time for the PM to recognise that τ_k^A has occurred

We note that item 2. forms the dominant component in the time required to achieve τ_k^A. That is, the majority of time required by the robotic system to achieve event τ_k^A will result from having to change the physical configuration of the task. Work by Hovland and McCarragher [36] showed that the time required for process monitoring is small. They used a multilayer perceptron neural net as a PM, and showed that process monitoring in real time is possible, (ie. for a reasonably complex asymmetrical insertion task, events were recognised in the order of 1.5 milliseconds on standard computer hardware). The time required for discrete event control using our method of Chapter 4 will be greater. The method can be computationally intensive, especially if a large number of points in the connectivity graph for $\gamma_{i_1}^A$ were generated. However, discrete event control need not occur

in real time. Indeed, a more sensible approach would be to do the processing between the demonstration and execution phases of the PbD process, ie. after a desired event path σ has been determined by the EPP, but prior to task execution. An appropriate $u(t)$ for each event in $\mathcal{A}_D$ would then be available for use by the robot at execution time.

The dominance of the CC in determining the time required to achieve τ_k^A means that our metric for predicting performance in the time area is simplified. We identify that the time required to achieve event τ_k^A will depend mainly on the amount the task configuration is required to change for the event to occur, ie. on the "distance" in C-space along the path traversed by the demonstrator to achieve this event. Then an obvious measure for predicting the time required for an event is the distance in C-space along the path used to achieve the event in the demonstration. Calculating this distance is straightforward, ie. we saw with Equation 4.9 in the previous chapter the details of how such a calculation can be made. Note that where more than a single path for event τ_k^A was demonstrated, the average length of the paths were taken. Denote this average length as L_{av}^k. Then, our metric for calculating a cost in the time performance area Ct_k for τ_k^A in $\mathcal{A}_D$ is:

$$Ct_k = L_{av}^k \tag{5.1}$$

(b) Reliability

The reliability of execution by the robotic system will depend on the reliability of each of the components in the system. That is, its reliability can be separated into three separate components:

1. The reliability of the PM in identifying τ_k^A correctly when it occurs
2. The reliability of the DEC in selecting an obstacle-free u(t) that will cause τ_k^A to occur
3. The reliability of the CC in precisely following u(t) so that an erroneous event (ie. one that is not τ_k^A) does not accidently occur.

The main source of execution failure in robotic systems stems from item 3. Regarding item 1, Hovland and McCarragher [36] showed that a 95 percent success rate for Process Monitoring can be achieved, even with quite small training sets for the Multi Layer Perceptron. Results from our experiments (which we will present later in this chapter) using this type of PM are consistent with their findings. Regarding item 2, the DEC is also something that can be achieved with reliability. We saw in the previous chapter how, if parameters to the method were set conservatively, every command produced by our DEC was valid and obstacle-free. Failure caused by item 3 result mainly from flexibility and backlash in the robot and workpiece (ie. the spindle). The relative positioning of the workpiece and the environment can then not be precisely known or controlled. Failure by this mode is well known in robotics, especially for tasks involving contact between task objects [53]. A strength of the hybrid force-position control regime is that it uses constraints provided by contact in the task to guide assembly motion. That is, if many constraints are present, then motion can be guided, and more reliable execution of the task achieved. Then, a good measure for predicting the reliability of event τ_k^A is the constraint on motion provided in state $\gamma_{i_1}^A$, ie. for the spindle assembly task, the dof of the spindle in state $\gamma_{i_1}^A$. Spindle dof provides one element in a measure of how well motion is *guided.* However, for motion to be guided, movement in $\gamma_{i_1}^A$ must occur. That is, from a reliability point of view, we would like τ_k^A to be included in the desired event path if $\gamma_{i_1}^A$ contains demonstrated paths that also involve significant movement. The amount of movement defined by a path is measured as the distance along the path in C-space. We defined the measure for this distance in Section 5.3.2(a) as L_{av}^k. Then a metric for calculating a cost in the reliability area Cr_k for τ_k^A is calculated as:

$$Cr_k = dof_{\gamma_{i_1}} \times L_{av}^k \times S_{\gamma_{i_1}} + 1 \tag{5.2}$$

where $dof_{\gamma_{i_1}}$ is the spindle degree of freedom in state γ_{i_1}, and $S_{\gamma_{i_1}}$ is the number of neighboring states to state γ_{i_1} in $\mathcal{A}_D$. We multiply $dof_{\gamma_{i_1}} \times L_{av}^k$ with $S_{\gamma_{i_1}}$ because incorrect

traversal of γ_{i_1} could cause any event out of γ_{i_1} to occur, ie. we could incorrectly enter into any states neighboring γ_{i_1} in $\mathcal{A}_D$ [2]. We add the value 1 so that a state with zero spindle degrees of freedom is not assigned a zero cost. An arc with zero cost in the automaton can cause problems for our search algorithm when trying to find the minimum-cost path.

(c) Control Effort

By control effort we mean the computational effort in controlling the assembly process. Recall how the HDS modeled assembly skill as the repeated application of the same skill "loop", ie. recognising a contact formation, determining a command to reach the next desired contact formation, applying that command, recognising the new contact formation, etc, etc. For each loop, computational effort is required. The PM must process force and position data in order to recognise the event, a control command $\mathbf{u}(t)$ must be selected by the DEC, and the CC must control the robot so that $\mathbf{u}(t)$ is physically achieved in the workspace. Clearly, from a computational point of view, the less times we need to repeat this loop, the better. Then a good measure of control effort for task completion using a certain event path is the number of events in that path. That is, a cost in the control-effort area Ce_k for transition τ_k^A is calculated simply as:

$$Ce_k = 1 \tag{5.3}$$

Then, any event path selected by our method will have a cost in the control-effort performance area equal to the length of the path itself.

(d) Number of Demonstrations

Determining a cost Cn_k for τ_k^A in the number of demonstrations performance area is straight forward. The cost should be set inversely proportional to N_k, the number of times transition τ_k^A was demonstrated. It should be set *inversely* proportional to N_k so

[2] In fact we could incorrectly enter any neighboring states to γ_{i_1} in $\mathcal{A}$, however we will not in general fully know $\mathcal{A}$. We will only have knowledge of that part of $\mathcal{A}$ that is $\mathcal{A}_D$.

that τ_k^A with good performance in this area (ie. those demonstrated often) are assigned a low cost in $\mathcal{A}_D$, while those demonstrated less often are assigned a high cost. Then a metric for calculating a cost in the number-of-demonstrations performance area Cr_k for τ_k^A is calculated as:

$$Cn_k = (N^{max} - N_k)^{dof} + 1 \tag{5.4}$$

where N^{max} is the number of times the most demonstrated transition was demonstrated. We raise $(N^{max} - N_k)$ to the power of dof (the dof of the task, ie. four in our case) so that the cost assignment is not linear with respect to $(N^{max} - N_k)$. That is, more repeated events are given a much lower cost than infrequently demonstrated events. This cost assignment structure is necessary so that a short path with infrequently demonstrated events is not selected over a longer path with more frequently demonstrated events. Analysis suggests that the dof of the task is a good value to use in the indexation, ie. because the length of path through $\mathcal{A}_D$ will be proportional to dof. Note also in Equation 5.4 that we have added a value of 1. This is to ensure that Cn_k can never equal zero, ie. when τ_k^A is the most demonstrated event in the task. An arc with zero cost in $\mathcal{A}_D$ can be problematic for a search algorithm when searching in for a minimum-cost path.

(e) Determining an Overall Cost

We identified four performance areas in which we categorise robot performance. Note then that a transition may result in good robot performance in one area and poor performance in another. A method is required to calculate the overall performance of an event. To determine the overall performance of an event we introduce the following measure:

$$C_k = W_t Ct_k + W_r Cr_k + W_e Ce_k + W_n Cn_k \tag{5.5}$$

where C_k is the overall cost calculated for τ_k^A, and W_t, W_r, W_e, and W_n are a set of weights. The role of the weights is to allow the overall cost of a transition to modified

according to what aspect of performance is deemed important by the demonstrator. For example, if the time weight W_t is set to a value greater than the other weights, then the overall cost C_k will be determined more by the event's cost in the time performance area. If we adopt a greater W_t value for all transitions in $\mathcal{A}_D$, then a search through $\mathcal{A}_D$ for the least cost path will result in a path with low execution time. In this way, our event-path selection framework allows robot performance to be tuned according to what is required. Note that by increasing the value of the other weights in the set, paths can be selected that cause the robot to perform reliably, with low in control effort, or to copy the demonstrator.

5.4 Results

5.4.1 Paths Selected by Framework

With the cost C_k for each arc in $\mathcal{A}_D$ determined, it was a straight-forward matter to find an optimal event path between the start and goal states in the task. The well known "double-sweep" algorithm was used for this purpose (see [32] for details), and we show in Figure 5.1 the event paths selected for the demonstration set D_1 to D_6 of the spindle-assembly task. The figure shows four selected paths in total. In the first column, it shows the path selected when the time weight was emphasised. We have labelled this path the "time path". In the second and third columns, the figure presents the paths selected when the reliability weight, and the number-of-demonstrations and control-effort weights were emphasised. We have labelled these paths the "reliability/number-of-demonstrations path" and "control-effort path". Finally, in the fourth column of the figure, we show the path selected by our framework when an even emphasis was given to all weights. We have labelled this path the "even path". Note that we provide on the left hand side of the figure an "event count" column to indicate the number of events existing in each path.

Recall from Table 5.1 the main execution strategies used by the demonstrator to complete the spindle-assembly task. Two interesting and important questions regarding the paths selected by our framework are: what strategies were adopted in each of the paths,

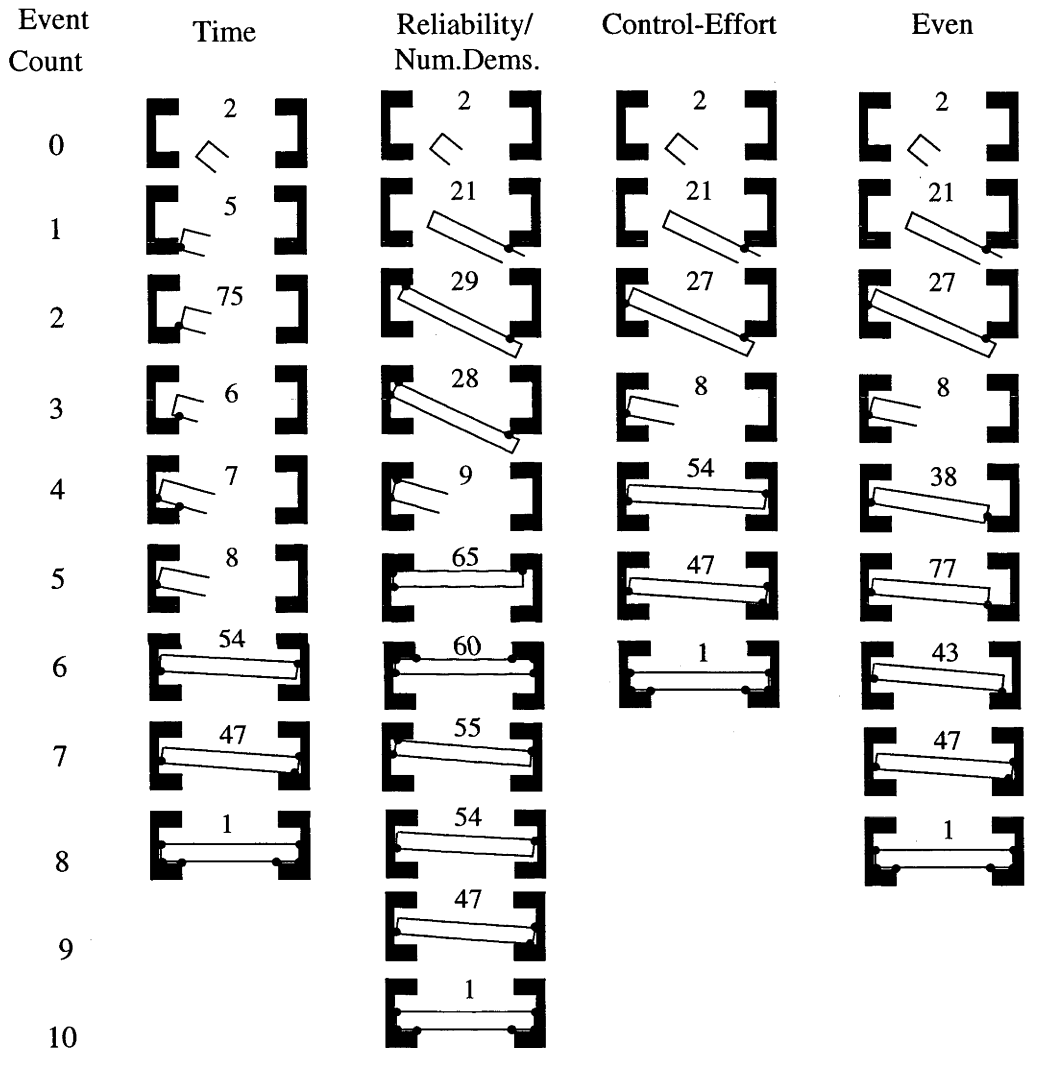


Figure 5.1: Event paths selected by our framework

and does the adoption of these strategies make sense given the weight that was emphasised to produce the path?

The time-path has used the direct-body and direct-head assembly strategies. The reason is because both these methods use an essentially "straight-line", direct approach to inserting both the spindle head and body. Our use of path length as the basis for the metric in this performance area has caused these strategies to be selected. Other strategies that were demonstrated (ie. the guided, push-forward, creep-over, and delayed strategies) all have longer path lengths in comparison, and hence were not selected for the time path.

In contrast to the time path, the reliability path contains the guided strategy for spindle head insertion, and the push-forward strategy for spindle-body insertion. These strategies were selected because of the guided motion each path provides. Our assertion was that the robot can traverse a state more reliably when its motion is more guided. Then, for spindle-head insertion, our framework has selected the 2-21-29-28 approach, as opposed to either the direct-head or delayed assembly approaches. It has chosen the guided approach over the direct head approach because of the guided motion allowed by state 21 for spindle head insertion. Much of the distance traveled occurs in state 21, where a dof of 3 exists. This is in contrast to the direct-head approach where most of the "distance" traveled occurs in state 2 (the no contact state), where no guiding occurs. The guided approach to spindle head insertion was selected over the delayed approach because of the shorter path total length it encodes.

The push-forward approach to spindle body insertion was selected for the reliability path. It is easy to see why this approach was selected over the direct-body approach, ie. it contains states that are on average result in a lesser degree of spindle freedom. To see why the push-forward approach was selected over the creep-over approach, recall that spindle-body insertion involves pivoting the spindle about a point in contact with the left support so that it can be passed around the front lip of the rebate in the right support. The push-forward approach was demonstrated using state 9 (spindle dof of 2) to achieve this pivoting motion, while the creep-forward approach was demonstrated with state 8

(spindle dof of 1) for the same motion. That is, the push-forward approach was selected because state 9 encodes a greater constraint and guides motion more than in state 8.

The control-effort path uses the guided approach for spindle-head insertion and the direct-body approach for spindle-body insertion. Understanding why these strategies were selected is straightforward. Recall that our metric in this performance area was based on the number of events in an event path. Studying the automaton $\mathcal{A}_D$ in Figure 2.7 reveals that the control-effort path is the path between the start and goal states in the task with the least number of events.

The path selected in the number-of-demonstrations performance are was the same as the reliability path. It was selected because of the frequency with which the strategies in this path were demonstrated. The guided approach to spindle head insertion was demonstrated two out of six times (D_3 and D_6), while the push-forward approach to spindle-body insertion was demonstrated three out of six times (D_2, D_3 and D_6). In addition, many of the events in these strategies were demonstrated in isolation elsewhere in the demonstration set. For example, (2-21) was demonstrated three times, (54-47) four times, and (47-1) six times.

The even path uses the guided approach to spindle-head insertion, and the creep-over approach to spindle-body insertion. This selection of strategies represents a balanced approach to the assembly. The guided approach is not as optimal in the time area as the direct-head method, however it is not that much longer, and is more reliable. The creep-over approach to spindle-body insertion is also a balanced selection. It is not as fast as the direct-body approach but is more reliable. It is not as reliable as the push-forward approach (because , as note previously, state 8 is used for pivoting rather than state 9), but it is shorter in distance (and therefor faster) than that method. The length of the path is not as short as the control-effort path, but is shorter than the reliability path. This even path also contains events that were used often by the demonstrator, eg. (2,21), (27,8), and (47,1).

We have seen what strategies were adopted in the paths selected by our framework. Of

interest also are strategies that weren't selected in any path, ie. we want to be sure that non-selected strategies were left out for the correct reasons. The delayed insertion strategy was the only strategy not selected at some point by our method. The main reasons for its absence from selected paths are that it is quite a long path in both time (ie. distance to be moved) and control effort (number of transitions), and that this strategy was only demonstrated once (in demonstration D_5).

5.4.2 Robot Execution of Selected Paths

We have presented the paths predicted by our framework to be performed best by the robot in each performance area. Our aim in this section is to validate our approach by having the robot execute selected paths. We want to show how paths predicted to be performed well are actually performed well in reality. However, prior to presenting the details of the experimental results, we first outline the experimental set up that was used.

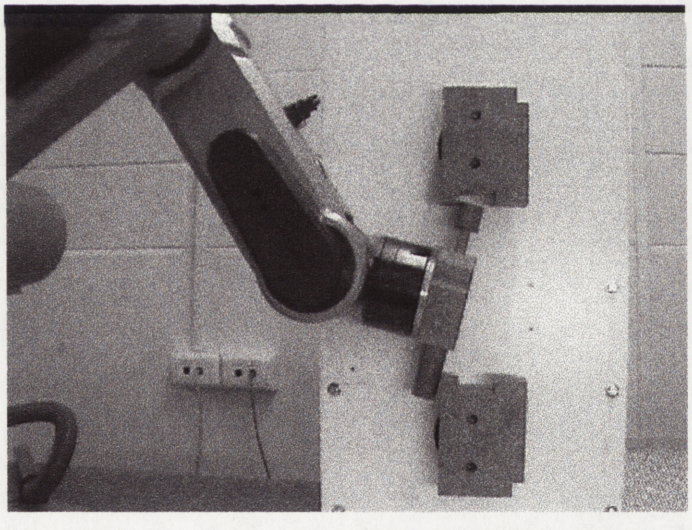


Figure 5.2: The Scorbot Eshed robot on which experiments were conducted

In order to have the robot execute selected paths, each element in the robotic system needed to be implemented. Recall from Figure 2.3, the components in the robotic system (ie. the manipulator, CC, DEC, EPP, and PM). A functional EPP is provided by our work

in this chapter. We present the robot manipulator used in experiments in Figure 5.2, and details regarding the continuous controller used in this thesis were discussed in Chapter 2. Work in the previous chapter derived a functional DEC that was used for experiments here. The component for which we have not yet had need is the Process Monitor. A functional PM was required to be implemented for the experiments we present in this chapter.

Our assertion in this thesis has always been that, for PbD in service robotic environments, a PM should be trained by demonstration, rather than be derived from apriori-known, geometric, task information. Recall our discussion in Chapter 2 on the subject. Then an appropriate method for deriving a PM for our experiments here was by using the method presented by Hovland and McCarragher [36]. Recall that their work derived a functional PM for a complicated, asymmetric, assembly task using a multi-layer perceptron. For our experiments in this chapter we made use of the MLP process monitor implementation coded by these authors in the Automated Systems Laboratory at the Australian National University. Details of the method can be found in [36] or [34], however an overview of the method is as follows.

The MLP consists of nodes in three layers, with nodes in each layer being fully connected to nodes in neighboring layers. Into the input layer is fed position data from the polhemus, and force data from the force sensors (but converted first into the frequency domain). Each node in the output layer corresponds to one of the events to be process monitored in the task. During the training phase, sample demonstrations of an event are provided. The force and position data from these samples are input into the input layer, while the node in the output layer corresponding to the event being demonstrated is set to a value of 1. All other nodes in the output layer are set to a value of zero. A back-propagation method is then used to set values of a set of weights, where each weight in the set corresponds to one of the connections between nodes in the MLP [3]. During the operation phase, position and force data input into the input layer of the MLP result in

[3]For a detailed description of the training method, see for example [101].

Path Name	Weight Emphasised	State Sequence	Successful Executions	Execution Time (secs)	Computation Time (secs)
Time	W_t	2-5-75-6-7-8-54-47-1	6/10	34.3	1.04
Reliab./N.Dems.	W_r, W_n	2-21-29-28-9-65-60-55-54-47-1	10/10	52.7	1.29
Control Effort	W_e	2-21-27-8-54-47-1	7/10	39.1	0.81
Even	equal	2-21-27-8-38-77-43-47-1	8/10	42.0	1.12
Delayed	n/a	2-21-30-11-66-61-56-49--78-79-48-47-1	8/10	1:09.6	1.51

Table 5.2: Results of implementing the paths selected by our framework on the robot

each output node being set to a certain value. The selection of the event that occurred is then achieved by simply identifying the output node with the highest value. Note that for our experiments here, we constructed a MLP with output nodes only for events existing in selected paths, ie. we did not train-up the PM to recognise every possible transition in the spindle-assembly task.

With a PM implemented, our experiments could be conducted. Ten attempts at each selected path were performed by the robotic system. Table 5.2 shows the results of these experiments. In the first column it shows the name of each path, while in the second column, it shows the weight emphasised to produce the path. The third column in the table shows the state sequence encoded by each path, and the fourth, fifth, and sixth columns show for each path, the time taken for execution, the proportion of successful executions achieved, and the computation time required for control purposes. In addition to paths selected by our framework, we also include in the last row in Table 5.2, results for robot execution of the delayed strategy. Recall how this strategy was identified by our framework as corresponding to less-optimal robot performance of the task. We had the robot execute this path to confirm that it really did encode less optimal performance.

In the time performance area the time path was confirmed as the most optimum mode of execution. Times for execution in our experiments ranged from 34.3 seconds for the time path, to 1 minute and 9.6 seconds for the delayed path. The reason for the faster

execution time for the time path was the more direct route taken by this path compared to others, ie. a very direct, and therefore fast, method of insertion is enabled by using the direct-head and direct-body insertion approaches. At the other end of the spectrum, the delayed path and the reliability path resulted in slow execution times. The reason was because both these paths meant the spindle had to be moved a greater distance compared to the time path. For example, the reliability path uses the guided approach to spindle head insertion and the push-forward approach to spindle-body insertion. Compared to the direct-head and direct-body approaches used by the time path, these strategies required a longer distance for the spindle to be moved, and therefore a greater execution time.

The reliability of the robot for the different paths ranged from six successful executions in ten attempts for the time path, to ten from ten attempts for the reliability path. Failures in experiments were caused mainly by the CC. Flexibility in the spindle and robot arm, along with backlash in the robots joints, and the quite fine tolerances required in some areas of the task, caused the CC to sometimes not achieve an event in a path. The time path was the least reliability path in the experiments. For this path, three of the four occasions where failure occurred was caused the assembly process accidently moving from state 8 into state 9, ie. the spindle head accidently made contact with the side of the rebate in the left support. On the other occasion where failure occurred, the process moved into state 4 from state 2, ie. the spindle head clipped the front edge of the rebate in the left support when on its way to state 5. The secret of the high reliability of the reliability path turned out to be the guided motion encoded by the guided and push-forward execution strategies. Failure in the time path was caused by incorrect positioning of the spindle relative to the supports. The guided approach to spindle-head insertion used in the reliability path was more reliable because of the constraint provided in state 21. This constraint reduced the possibility of flexibility and backlash in the robot from causing incorrect positioning of the spindle relative to the supports. State 21 was not the only state in which guided motion contributed to the reliability of the path. The presence of state 9 (instead of state 8), and of states 60 and 65 in the path also contributed in

this regard. Recall the need to pivot the spindle about a point in contact with the left support when inserting the spindle body into the rebate of the right support. The use of state 9 instead of state 8 during this pivoting motion meant that the pivot point was more constrained, ie. the spindle head did not accidently slip into another state (as occurred in the time path). States 65 and 60 also assisted in making the reliability path reliable because they provided a very constrained way to insert the spindle body into the right support. Spindle-body insertion using these states see the back-edge of the rebate in the right support used as a guide for the insertion. In summary, our results suggest that a valid approach to predicting the reliability of an event is the extent to which motion in the prior state is guided.

Column six in the Table 5.2 shows the computation time required for control purposes for each path. Processing in experiments was conducted on a Motorola 68040 based VME board. The computation time shown for each path was calculated as the sum of the computation times required be each of the PM, DEC and CC functionalities. Recall that a set of valid $\mathbf{u}(t)$ by the DEC is achieved ahead of time, ie prior to experiments. Computation for selecting the correct $\mathbf{u}(t)$ to be used for each event needed negligible computation time. The PM and CC were the main contributors to the computation times shown in the table. Notice the relationship between the number of events in each path and the computation time required, ie. the longer the path, the longer generally the greater computation required. We found that each event required about the same computation time to process monitor, so the more events in a path, the greater computation required by the PM. We also found that paths with more states required greater computation by the CC. These findings suggest that the number of events in a path is a valid metric for predicting control effort.

We have discussed the results of experiments in the time, reliability, and control-effort performance areas. The number-of-demonstrations performance area is different to other areas in that we do not need experimental results to validate the path selected. That is, we saw in Section 5.4.1 how this path *did* copy the demonstrator. The reason

for experiments in this performance area is to validate our assertion that copying the demonstrator leads to more optimal robot performance of the task. Experiments suggest that our assertion is a valid one. We have seen that the number-of-demonstrations path included the guided approach to spindle-head insertion and the push-forward approach to spindle-body insertion. One of the reasons identified by the human for using these strategies was their reliability. We have noted the reasons for the reliability of these strategies above, ie. they allow assembly motion to be "guided". However, the human identified another, additional reason why these paths encode reliable execution. Note how events (2-21) and (9,65) both provide a reliable way to make contact between the spindle and the right support. For example, event (2,21) can be reliably achieved because the large, flat, contact area provided by the spindle's side, ie. the required contact can be achieved, even if some uncertainty exists about the spindle's position relative to the support. For the same reason, event (9,65) also encodes a reliable method for making contact between the spindle and right support. Reliable robot execution of the number-of-demonstrations path has resulted in part from the inclusion of events (2,21) and (9,65) in the path. That is, in this case, copying the demonstrator has led to more optimal robot performance of the task. Note that reliability of execution for the reason that a large contact area exists was not modeled by our reliability metric. It would be very difficult to achieve a set of metrics that modeled every aspect of task performance identifiable by the intelligent human system. This fact gives strength to the inclusion of the number-of-demonstrations performance area our framework. It can capture for the robot, aspects of task performance identifiable by the human that are not modeled directly by metrics in other performance areas.

The paths discussed so far have resulted in excellent performance in one area, at the detriment to performance in the others. Recall the role of the even path as providing a balanced mode of execution. Then experimental results suggest that a balanced mode of execution *is* provided by the even path selected by our framework. This path was performed with better than average reliability, ie. it was successfully completed on 8 out

of ten occasions attempted. It was performed with an execution time of 42.0 seconds, which was around the average. Finally, it required less than average control effort, with a CPU computation time of 1.12 seconds. Such a path is appropriate if good all-round performance of the task is desired.

Recall that the delayed strategy for spindle-assembly task execution was not one selected by our framework. An interesting question then is whether the sub-optimal performance predicted by our framework for this path actually results in reality. The experimental results for this path shown in Table 5.2 suggest that it does. This path resulted in worse performance on average of the task compared to others. It took the longest time of any path for the robot to execute, the control effort required to execute the path is high due to the large number of transitions it involves, and the reliability of the path is around the average. The fact that this path was not selected by our framework again suggests the validity of our measures for predicting robot performance.

5.5 Conclusion

Our premise at the outset of this chapter was that a human can demonstrate a range of task-level, execution strategies. We stated our aim was to present a method that could select for the robot the most optimal strategies demonstrated. Then, according to our aim, the method presented in this chapter has been successful. We saw that a number of different strategies for the spindle-assembly task were demonstrated. Our method could select strategies that saw the robot perform optimally in a number of different performance areas. It could select strategies that resulted in task performance with low execution time, high reliability, low control effort, or that copied the demonstrator. In addition, it was able to select a strategy that saw the robot take a balanced approach to task execution. Finally, it was able to avoid selecting demonstrated strategies that resulted in less optimal performance of the task. Experimental result confirmed the validity of the selected strategies in each case. Where a strategy was selected for a particular performance area,

experiments showed that the robot performed with the expected characteristics. Where a balanced approach to task execution was desired, the robot performed the task well in each performance area. Where our method predicted poor performance, experiments confirmed that poor performance resulted. Our method in this chapter has provided an important, new step in the process of noise removal in PbD; one not seen before in the literature to date. That is, it has provided a method for identifying and removing sub-optimality in the demonstration at the task level.

Chapter 6

Conclusions

6.1 Introduction

This chapter brings the conclusion of the thesis. Our presentation here divides into two parts. First, we review in Section 6.2 the major results and conclusions of work in the thesis. Our focus is on identifying how the work presented has helped solve the problem of demonstration sub-optimality in PbD. The second part of this chapter identifies possible areas for further research. We do not presume that work presented provides a complete solution, nor one where improvements cannot be made. Section 6.3 discusses and highlights areas where further research is required.

6.2 Major Results and Conclusions

The content of this thesis divided nicely into three areas: (i) determining task-specific, geometric information, (ii) deriving noise-free, low-level, control commands, and (iii) selecting a task-level strategy. The major results and conclusions in each area can be summarised as follows.

(i) This work uses regression analysis on demonstration data to estimate the geometric properties of a task. It provides a valuable step for removing demonstration sub-optimality

in PbD, since task specific, geometric information allows sensible decisions to be made about whether a demonstrated action contains noise. The method proposed uses configuration space as the means for representing the geometric properties of a task. The method proved a valid, alternative way to the usual, geometric-model-based methods for determining C-space. It had the advantage of requiring only a set of demonstrations of the task, and so was well suited to applications where a geometric model was not available, eg. for PbD in domestic type environments. A key finding for the method was that the accuracy of its representation for a particular region in C-space was found to depend on how often that region was visited in the demonstration. For frequently visited regions, an accurate representation of the C-space topology resulted. Where a region was less-frequently visited, a less accurate representation prevailed. We identified this fact as a weakness of the method compared to geometric-model-based methods for determining C-space. Our approach in this thesis was to circumvent the use of regions where a less accurate representation existed by deriving commands for the robot that visited regions well-visited in the demonstration.

(ii) This work divides into two parts. The first part proposed a path planning technique to determine noise-free paths in partially known configuration spaces. The proposed method was used to derive noise-free position control commands. The method has the advantage over others in the literature that: (a) it can derive paths containing undemonstrated points, so significant noise removal is possible, (b) it does not assume all demonstrated paths pass between the same start and end points, (c) it can derive paths lying outside an envelope formed by demonstrated paths, and (d) it can be applied to find paths in a C-space of any dimension. A limitation of the method is that a non-valid path can be derived if parameters in the method are not set appropriately. The second part of work in this area proposed force control-command synthesis. The proposed method determines the direction of force control as orthogonal to our current position on a C-surface. It determines a noise-free magnitude for force control based on smoothing and spline fitting. The major advantages of the method are that (a) it is able to remove the friction component from demonstrated

force commands, (b) it is able to remove high frequency noise included by force sensors in demonstrated force commands, and finally (c) it provides a valid way to produce force commands for points in a position control command that were undemonstrated.

(iii) This work proposes a method for selecting task-level execution strategies. It is essentially the process of removing sub-optimality in the demonstration at the task level. No other work of this type has been presented in the literature to date. Given this fact, the key finding for the work is that removing demonstration sub-optimality at the task level is a valid and important step in PbD. Results in this thesis showed that robot performance of a task can be improved by actively selecting more optimal demonstrated strategies over those that are less optimal. Two other major findings for the work were made. First, that the discrete event formalism of HDS skill modeling provides a powerful and valid description of execution strategy at the task level. Second, that by allowing the demonstrator to select as important one of four possible execution characteristics, flexibility into the PbD programming approach can be achieved. Experiments in this thesis showed that the demonstrator could tailor how the robot performed a task, depending on what was desired.

6.3 Further Research

A number of areas of further research for work in this thesis exist. They are:

6.3.1 Regression analysis using force data

The regression analysis of Chapter 3 determined unknown parameters for a C-surface equation based only on demonstrated position data. An augmented approach would be to also use in the regression analysis, demonstrated force data. Force data should lie in the sub-space orthogonal to a C-surface, however, it is well known that noise and friction will corrupt demonstrated force data. Noise, assuming it is white, will be identified and removed by the regression process. An estimate of the friction-free demonstrated force

can be achieved by (i) identifying a demonstrated "velocity" using sequential position data points on a C-surface, and (ii) removing the force component lying parallel to this velocity. We have noted that our method for deriving C-space can provide less-accurate estimates for regions of C-space where little demonstration data was available. Including force data in the regression process would improve estimates for these regions.

6.3.2 Process monitor assumptions

Work in Chapter 3 made the assumption that a PM could return information about (i) whether an event resulted in a gain or loss of constraint on the manipulated object in the task, and (ii) the type of constraint that was gained or lost. We identified at the time some methods for determining this type of information. For example, we noted that (i) may be identified by using reasoning on contact forces and spindle velocities. Alternatively, we suggested an approach based on training coupled with "learning" methods (eg. neural networks), as adopted by [114, 35] for identifying when an event in a task occurs. The method we presented in Chapter 3 was based around the assumption that this type of information could be determined automatically by the PM. Research into methods for achieving this functionality in the PM is required.

6.3.3 Optimising paths across states

Synthesis of a position control command in Chapter 3 took, as the start and end points of the command, demonstrated points on the boundary of a C-surface. Noise removal for a demonstrated path was achieved by removing noise in a piece-wise fashion from the segments in each state in the path. The assumption made was that noise in the demonstration will exist only within each state, and does not exist "across" states. An extension of the approach presented in this thesis would be to allow undemonstrated start and end points. Then, if the point used in the demonstration for a transition between two states is inefficient, the inefficiency will be identified and removed. Of course, the undemonstrated start and end points would have to be selected with care, ie. (i) so they

do not lie on the non-obstacle-free side of unknown boundaries, and (ii) so they will be connected to points in likely-free regions and/or interior segments during the connectivity-graph construction phase of the process. Solutions for ensuring (i) and (ii) would form the major part of further work in this area.

6.3.4 Setting parameters for position control command synthesis

Our method for position control command synthesis of Chapter 3 was based around a set of parameters. We identified how successful command synthesis depended on appropriate values being selected for these parameters. We also noted that, in a fully implemented PbD system, parameter selection must be achieved automatically. Section 4.3.7 in Chapter 3 presented a number of simple measures used to set parameter values for the spindle assembly task. Experiments showed the validity of these measures, however, further research into this area of the work is required. The research should identify two facts: whether the measures in Section 4.3.7 are applicable to a wider range of tasks than the spindle assembly task, and if not, what measures are appropriate for a more general range of tasks.

6.3.5 Discrete event sequence as task-level description

Work in Chapter 5 used a single discrete event as its basis for a task-level description of execution-strategy. That is, we predicted how the robot would perform each individual event, assigned an appropriate cost for that event in the automaton, and searched the automaton for a minimum cost path. An alternate basis for describing execution strategy at the task level may be as a *sequence* of events. For example, work presented in [47], investigated the idea of using the complete sequence of events encoded by a demonstration as the basis for describing execution strategy at the task level. That is, each demonstration was taken as a separate demonstrated strategy, and, in the same way as for work in Chapter 5, a prediction of robot performance for each was made in order to select the most optimal. Work in Chapter 5, and work in [47], represent the two extremes in the number of events

to take in a sequence. Preliminary work in this area has identified that a different event path will result depending on what length of sequence is adopted. Then a possible focus for further work is to investigate (i) the relationship between the desired path selected and the sequence length adopted, and (ii) whether there is an "optimal" sequence length, ie. one that results in the most optimal desired event path.

Bibliography

[1] I.Imam abd J.E.Davis. Robot simulation and off-line programming - an integrated cae-cad approach. In B.Ravani, editor, *CAD Based Programming for Sensory Robots*, volume 50 of *NATO ASI Series F: Computer and Systems Sciences*, pages 189–201. Springer-Verlag, 1988.

[2] G.Messina A.Capizzi and G.Tricorni. Robot programming languages standardization in manufacturing environment. In *Proceedings of the IEEE/RSJ International Conference on Inelligent Robots and Systems*, pages 488–492, July 1993.

[3] A.Danthine and M.Geradin, editors. *Advanced Software in Robotics.* Elsevier Science Publishers B.V. (North Holland), 1984.

[4] A.Gollu and P.Varaiya. Hybrid dynamic systems. In *Proc. of the 28th IEEE Conference on Decision and Control*, pages 2708–2712, Tampla, Florida, December 1989.

[5] A.Liegeois. Simulation as a programming aid. In U.Rembold and K.Hormann, editors, *Languages for Sensor Based Control in Robotics*, volume 29 of *NATO ASI Series F: Computer and Systems Sciences*, pages p331–341. Springer-Verlag, 1987.

[6] A.P.Ambler and D.F.Corner. RAPT1 *users manual.* Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, 1982.

[7] Christopher G. Atkeson and Stefan Schaal. Learning tasks from a single demonstration. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 1706–1712, May 1997.

[8] Australian Robot Association. *Proceedings of the International Conference on Field and Service Robotics*, Canberra, Australia, December 1997.

[9] B.Hannaford and P.Lee. Hidden markov model analysis of force/torque information in telemanipulation. *The International Journal of Robotics Research*, 10(5):528–539, October 1991.

[10] B.J.McCarragher. *A discrete event dynamic systems approach to robotic assembly.* PhD thesis, MIT Department of Mechanical Engineering, July 1992.

[11] B.J.McCarragher. Force sensing from human demonstration using a hybrid dynamical model and qualitative reasoning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 557–563, San Diego, CA, USA, May 1994.

[12] B.J.McCarragher and H.Asada. The discrete event control of robotic assembly tasks. *ASME Journal of Dynamic Systems, Measurement and Control*, 117(3):384–393, September 1995.

[13] C.C.Geschke B.Shimano and C.Spalding. VAL-II: a robot programming language and control system. In *SME Robots VIII Conf.*, pages 20:103–20:119, Detroit, MI, June 1984.

[14] M. Skubic S.P. Castrianni and R.A. Volz. Identifying contact formations from force signals. In *Proceedings of the 1997 International Conference on Neural Networks*, pages 1623–1628, 1997.

[15] C.Blume and W.Jacob. *PASRO: PASCAL for robots.* Springer Verlag, 1985.

[16] C.Crangle. Conversational interfaces to robots. *Robotica*, 15:117–127, 1997.

[17] J.Duffy C.D.Crane III and M.Locke. Off-line programming and path generation for robot manipulators. In B.Ravani, editor, *CAD Based Programming for Sensory*

Robots, volume 50 of *NATO ASI Series F: Computer and Systems Sciences*, pages 425–432. Springer-Verlag, 1988.

[18] C.Thorpe. Mixed traffic and automated highways. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Grenoble, France, September 1997.

[19] D.A.Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, Spring 1991.

[20] D.A.Pomerleau. Progress in neural network-based vision for autonomous robot driving. In *Proceedings of the Intelligent Vehicles '92 Symposium*, pages 391–396, Pittsburgh, PA, USA, June 1992.

[21] D.C.Lay. *Linear Algebra and its Applications.* Addison Wesley, 1994.

[22] D.Lees and L.Leifer. A graphical programming language for robots in lightly structured environments. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 648–653, Atlanta, GA, USA, May 1993.

[23] D.R..Myers. An approach to automated programming of industrial robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3109–3113, Detroit, Michigan, USA, April 1999.

[24] K.Kapellos D.Simon and B.Espiau. Control laws, tasks and proceedures with orccad: application to the control of an underwater arm. In *Proceedings of the 6th IARP Workshop on Underwater Robotics*, Toulon, France, March 1996.

[25] K.Kapellos D.Simon, B.Espiau and R.Pissard-Gibollet. Orccad: software engineering for real-time robotics, a technical insight. *Robotica*, 15(1):111–115, January-February 1997.

[26] D.Wenrui and M.Kampker. Pin-a pc-based robot simulation and offline programming system using macro programming techniques. In *IECON'99. Conference Pro-*

ceedings. 25th Annual Conference of the IEEE Industrial Electronics Society, pages 442–446, San Jose, CA, USA, November 1999.

[27] D.Wenrui and M.Kampker. User oriented integration of sensor operations in a offline programming system for welding robots. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation*, pages 1563–1567, San Francisco, CA, USA, April 2000.

[28] E.C.Koenig. Parallel processing considerations for interactive man-robot systems. *Systems Analysis Modeling Simulation*, 16(1):1–8, 1994.

[29] E.Marchand. Visp: A software environment for eye-in-hand visual servoing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3224–3229, Detroit, Michigan, USA, April 1999.

[30] E.Mazer. LM-GEO: Geometric programming of assembly robots. In A.Danthine and M.Geradin, editors, *Advanced Software in Robotics*, pages 99–110. Elsevier Science Publishers B.V. (North Holland), 1984.

[31] E.Mazer and J.F.Miribel. Manuel de reference lm. Technical report, Artificial Intelligence and Robotic Group laboratorie, IMAG, October 1982.

[32] E.Minieka. *Optimization Algorithms for Networks and Graphs.* Marcel Dekker Inc.

[33] A.W.M.Van Den Enden and N.A.M Verhoeckx. *Discrete-Time Signal Processing, An Introduction.* Prentice Hall International, 1989.

[34] G.E.Hovland. *Control of sensory perception for discrete event systems.* PhD thesis, Australian National University, 1997.

[35] P.Sikka G.E.Hovland and B.J.McCarragher. Skill aquisition from human demonstration using a hidden markov model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2706–2711, Minneapolis, Minnesota, USA, April 1996.

[36] Geir.E.Hovland and B.J.McCarragher. Combining force and position measurements for the modeling of robotic assembly. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'97), Grenoble, France*, pages 655–660, September 1997.

[37] Hanqiu Sun. Xiaobu Yuan. Baciu G. Yunqing Gu. Direct virtual-hand interface in robot assembly programming. *Journal of Visual Languages & Computing*, 10(1):55–68, February 1999.

[38] H.Anton. *Elementary Linear Algebra 5e.* Wiley, 1987.

[39] H.Asada. Teaching and learning of compliance using neural nets: Representation and generalization to nonlinear compliance. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 1237–1244, May 1990.

[40] H.Asada and H.Izumi. Automatic program generation from teaching data for the hybrid control of robots. In *IEEE Transactions on Robotics and Automation*, pages 163–173, 1989.

[41] H.Ogata and T.Takahashi. A geometric approach to task understanding for assembly operations. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 695–700, Atlanta, GA, USA, May 1993.

[42] H.Ogata and T.Takahashi. A geometric approach to task understanding and playback: compact and robust task description for complex environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 848–854, San Diego, CA, USA, May 1994.

[43] Geir E. Hovland and Brenan J. McCarragher. Hidden markov models as a process monitor in robotic assembly. *International Journal of Robotics Research*, 146(17):266–267, October 1997.

[44] S.Van Huffel and J.Vandewalle. *The Total Least Squares Problem: Computational Aspects and Analysis.* Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, 1991.

[45] I.H.Witten. Pbd systems: when will they ever learn? In *Workshop: Programming by Demonstration vs Learning from Examples; International Conference on Machine Learning*, California, USA, July 1995.

[46] Fraunhofer IPA. Vr4 - software tool for dynamic and real-time oriented virtual environments. Company Brochure, Stuttgart, Germany, 1996.

[47] J.Chen and B.J.McCarragher. Robot programming by demonstration, selecting optimal event paths. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 518–523, April 1998.

[48] J.E.Lloyd and D.K.Pai. Extracting robotic part-mating programs from operator interaction with a simulated environment. In A.Casals and A.T.de Almeida, editors, *Experimental Robotics V. The Fifth International Symposium*, pages 675–686. Springer-Verlag, Berlin, Germany, 1998.

[49] D.K.Pai J.E.Lloyd, J.S.Beis and D.G.Lowe. Programming contact tasks using a reality-based virtual environment integrated with vision. *IEEE Transactions on Robotics and Automation*, 15(3):423–434, June 1999.

[50] J.F.Engelberger. *Robotics in Service.* Mit Press, Cambridge, Massachusetts, 1989.

[51] D.Lees L.Leifer H.F.M.Van der Loos I.Perkash J.Hammel, K.Hall and R.Crigler. Clinical evaluation of a desktop robotic assistant. *Journal of Rehabilitation Research and Development*, 26(3):1–16, Summer 1989.

[52] B.Espiau K.Kapellos R.Pissard-Gibollet D.Simon J.J.Borrelly, E.Coste-Manire and N.Turro. The orccad architecture. *International Journal of Robotics Research, Spe-*

cial Issue on Integrated Architecture for Robot Control and Programming, 17(4):338–359, April 1998.

[53] J.J.Craig. *Introduction to robotics: mechanics and control.* Addison-Wesley, second edition edition, 1989.

[54] J.L.Dallaway. Human-computer interaction with robotic workstations. In *Proceedings of the RESNA Annual Conference*, pages 339–341, 1986.

[55] R.D.Jackson J.L.Dallaway and R.M.Mahoney. The user interface for interactive robotic workstations. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1682–1686, 1994.

[56] R.D.Jackson J.L.Dallaway, R.M.Mahoney and R.G.Gosine. An interactive robot control environment for rehabilitation applications. *Robotica*, 11(6):541–552, November 1993.

[57] C.J.Nachtsheim J.Neter, M.H.Kutner and W.Wasserman. *Applied Linear Regression Models.* IRWIN, 3rd edition, 1996.

[58] J.Patrick. *Training: Research and Practice.* Academic Press, San Diego, CA, 1992.

[59] J.P.Merlet. Programming tools for force-feedback command of robots. In U.Rembold and K.Hormann, editors, *Languages for Sensor Based Control in Robotics*, volume 29 of *NATO ASI Series F: Computer and Systems Sciences*, pages 69–81. Springer-Verlag, 1987.

[60] JR3 Inc, 22 Harter Ave, Woodland, CA, 95776. *JR3: DSP-based force sensor recievers, software and installation manual*, April 1994.

[61] H.Kimura J.Takamatsu and K.Ikeuchi. Classifying contact states for recognizing human assembly tasks. In *Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 177–182, August 1999.

[62] Y.Xu J.Yang and C.S.Chen. Hidden markov model to skill learning and its applications to telerobotics. *IEEE Transactions on Robotics and Automation*, 10(5):1248–1255, October 1994.

[63] M. Kaiser and R. Dillman. Building elementary skills from human demonstration. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 2700–2705, April 1996.

[64] K.Ikeuchi, M.Kawade, and T.Suehiro. Toward assembly plan from observation, task recognition with planar, curved and mechanical contacts. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2294–2301, 1993.

[65] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[66] J.Latombe L.E.Kavraki, P.Svestka and M.H.Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, August 1996.

[67] L.I. Lieberman and M.A. Wesley. Autopass: an automatic programming system for computer controlled mechanical assembly. *IBM Journal of Research and Development*, 21(4):321–333, 1977.

[68] C.Crangle L.Liang and L.Leifer. A computational model for a robotic arm instructed by natural language. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 451–456, Los Angeles. CA, USA, Nov 1990.

[69] L.Sciavicco and B.Siciliano. *Modeling and control of robot manipulators.* McGraw-Hill, 1996.

[70] A.Birk H.Kitano M.Asada, R.D'Andrea and M.Veloso. Robotics in edutainment. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 795–800, San Francisco, California, USA, May 2000.

[71] M.Carter. Using telerobots for recreation. In *Proceedings of the International Conference on Field and Service Robotics*, pages 173–178, Canberra, Australia, December 1997.

[72] Brenan J. McCarragher and Haruhiko Asada. The discrete event modeling and trajectory planning of robotic assembly tasks. *Journal of Dynamic Systems, Measurements and Control*, 117(3):394–400, October 1995.

[73] M.H.Raibert and J.J.Craig. Hybrid position/force control of manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102/127:126–133, June 1981.

[74] H.Friedrich M.Kaiser and R.Dillmann. Obtaining good performance from a bad teacher. In *Workshop: Programming by Demonstration vs Learning from Examples; International Conference on Machine Learning*, California, USA, July 1995.

[75] S.P.Castrianni M.Skubic and R.A.Volz. Indentifying contact formations from force signals: A comparison of fuzzy and neural network classifiers. In *"Proceedings of the IEEE International Conference on Neural Networks*, volume 3, pages 1623–1628, June 1997.

[76] Y.Nakamura M.Uechi, H.Ogata and M.Mizukawa. A component achitecture for customizing robot-teaching systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, California, USA, May 2000.

[77] N.Delson and H.West. Robot programming by human demonstration: The use of human inconsistency in improving 3d robot trajectories. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1248–1255, September 1994.

[78] N.Delson and H.West. Robot programming by human demonstration: the use of human variation in identifying obstacle free trajectories. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 564–570, May 1994.

[79] N.Delson and H.West. Robot programming by human demonstration: Adaptation and inconsistency in constrained motion. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, 1996.

[80] N.Hogan. How humans adapt to kinematic constraints. In *Proceedings of the 7th Yale Workshop on Adaptive Learning Systems*, May 1992.

[81] N.M.Amato and Y.Wu. A randomized roadmap method for path and manipulation planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 113–120, 1996.

[82] L.Priolo P.Bison, E.Pagello and S.Ziviani. Simulation tools as a programming aid for robot programming. In U.Rembold and K.Hormann, editors, *Languages for Sensor Based Control in Robotics*, volume 29 of *NATO ASI Series F: Computer and Systems Sciences*, pages p343–357. Springer-Verlag, 1987.

[83] K.Ali P.Fiorini and H.Seraji. Health care robotics: A progress report. In *Proceedings of the IEEE International Confenmce on Robotics and Automation*, pages 1271–1276, Albuquerque, New Mexico, USA, April 1997.

[84] Polhemus Inc. *3SPACE FASTRAK Users Manual Revision F*, 1993.

[85] P.Sikka and B.J.McCarragher. Learning to recognize discrete state transitions in assembly. In *Proceedings of the Australian Robot Association's National Conference on Robots for Australian Industries*, Melbourne, July 1995.

[86] P.Varaiya. Smart cars on smart roads: Problems of control. *IEEE Transactions on Automatic Control*, 38(2):195–207, 1993.

[87] P.V.O'Neil. *Advanced Engineering Mathematics.* Wadsworth Publishing Company, second edition, 1987.

[88] P.V.Whalen. *Teaching accomodation task skills: from human demonstration to robot control via artificial neural networks.* PhD thesis, Air Force Insitute of Technology, Wright-Patterson Air Force Base, 1995.

[89] Joris De Schutter Qi Wang. Towards real-time robot programming by human demonstration for 6d force controlled actions. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 2256–2261, April 1998.

[90] Wim Witvrouw Qi Wang, Joris De Schutter and Sean Graves. Derivation of compliant motion programs based on human demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2616–2621, April 1996.

[91] Q.Wang and J.De Schutter. Force controlled robot programming by human demonstration. Technical Report 95R64, Katholieke Universiteit Leuven, Belgium, August 1995.

[92] S.Fleury M.Ghallab R.Alami, R.Chatila and F.Ingrand. An achitecture for autonomy. *The International Journal of Robotics Reseach*, 17(4):315–337, April 1998.

[93] M.Kaiser R.Dillmann and A.Ude. Aquisition of elementary robot skills from human demonstration. In *Proceedings of the International Symposium on Intelligent Robotic Systems*, Pisa, Italy, 1995.

[94] R.D.Schraft. Mechatronics and robotics for service applications. *IEEE Robotics and Automation Magazine*, 1(4):31–35, December 1994.

[95] R.H.Taylor. Synthesis of manipulator control programs from task-level specifications. Memo AIM 228, AI Lab, Stanford, July 1976.

[96] P.D.Summers R.H.Taylor and J.M.Meyer. Aml: A manufacturing language. *International Journal of Robotics Research*, 1(3):19–41, 1982.

[97] R.J.Popplestone and A.P.Ambler. A language for specifying robot manipulations. In A.Pugh, editor, *Robotic Technology*, pages 125–141. Peter Peregrinus, London, England, 1983.

[98] R.Koeppe and G.Hirzinger. Sensorimotor compliant motion from geometric perception. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 805–811, Kyongju, South Korea, October 1999.

[99] A.Breidenbach R.Koeppe and G.Hirzinger. Skill representation and aquisition of compliant motions using a teach device. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 897–904, Osaka, Japan, November 1996.

[100] R.Navon and A.Retik. Programming construction robots using virtual reality techniques. *Automation in Construction*, 5(5):393–406, February 1997.

[101] R.P.Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, April 1987.

[102] R.W.Brockett. Hybrid models for motion control systems. In H.L.Trentelman and J.C.Willems, editors, *Essays on Control: Perspectives in the Theory and Its Applications*, chapter 2, pages 29–5. Birkhauser, Boston, MA, 1993.

[103] G.Pardo-Castellote S.A.Schneider, V.W.Chen and H.W.Wang. Controlshell: a software architecture for complex electromachanical systems. *The International Journal of Robotics Reseach*, 17(4):315–337, April 1998.

[104] S.B.Huffman and J.E.Laird. Learning from highly flexible tutorial instruction. In *Proceeding of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, USA, July 1994.

[105] S.B.Huffman and J.E.Laird. Flexibly instructable agents. *Journal of Artificial Intelligence Research*, 3:271–324, Jun-Dec 1995.

[106] S.B.Kang and K.Ikeuchi. Determination of motion breakpoints in a task sequence from human hand motion. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 551–556, San Diego, CA, USA, May 1994.

[107] S.Bonner and K.G.Shin. A comparative study of robot languages. *IEEE Computer*, 15(12):82–96, 1982.

[108] J.De Schutter and P.Simkens. Cad-based verification and refinement of high level compliant motion primitives. In B.Ravani, editor, *CAD Based Programming for Sensory Robots*, volume 50 of *NATO ASI Series F: Computer and Systems Sciences*, pages 203–222. Springer-Verlag, 1988.

[109] H.Bruyninckx S.Dutre and J.De Schutter. Indentification and monitoring based on energy. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 1333–1338, Minneapolis, Minnesota, USA, April 1996.

[110] S.Hirai. Identification of contact states based on a geometric model for manipulative operations. *Advanced Robotics: The International Journal of the Robotics Society of Japan*, 8(2):139–155, 1994.

[111] H.I.Connacher S.Jayaram and K.W.Lyons. Virtual assembly using virtual reality techniques. *Computer Aided Design*, 29(8):575–584, 1997.

[112] S.K.Tso and K.P.Liu. Demonstrated trajectory selection by hidden markov model. In *Proceedings of the International Conference on Robotics and Automation*, pages 2713–2718, Albuquerque, New Mexico, April 1997.

[113] Marjorie Skubic and Richard A. Volz. Identifying contact formations from sensory patterns and its applicability to robot programming by demonstration. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 458–464, 1996.

[114] Marjorie Skubic and Richard A. Volz. Learning force based assembly skills from human demonstration for execution in unstructured enviroments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1281–1288, May 1998.

[115] S.Lee and S.Shimoji. Machine acquisition of skills by neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 781–788, New York, NY, USA, July 1991.

[116] S.Liu and H.Asada. Transfer of human skills to robots: Learning from human demonstrations for building an adaptive control system. In *Proceedings of the 1992 American Control Conference*, pages 2607–2612, 1992.

[117] S.Liu and H.Asada. Teaching and learning of deburring robots using neural networks. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 339–345, Los Alamitos, CA, USA, May 1993.

[118] C.Crangle S.Michalowski and L.Liang. Experimental study of a natural-language interface to an instructable robotic aid for the severely disabled. In *Proceedings of the 10th Annual Conference on Rehabilitation Technology*, pages 19–23, San Jose, CA, USA, Jun 1987.

[119] S.Mujtaba and R.Goldman. Al user's manual. Technical Report STAN-CS-81-889, Stanford Artificial Intelligence Laboratory, 1982.

[120] B.Blanchard S.Zegloul and M.Ayrault. Smar: A robot modeling and simulation system. *Robotica*, 15(1):63–73, Jan-Feb 1997.

[121] T.Arai. A robot language system with a colour graphic simulator. In A.Danthine and M.Geradin, editors, *Advanced Software in Robotics*, pages 215–226. Elsevier Science Publishers B.V. (North Holland), 1984.

[122] T.Arai and H.Yago. A graphical robot language developed in japan. *Robotica*, 15(1):99–103, Jan-Feb 1997.

[123] M.H.Lee T.G.Williams, J.J.Rowland and M.J.Neal. Teaching by example in food assembly by robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, California, USA, May 2000.

[124] F.Schwarz T.Horsch and H.Tolle. Motion planning for many degrees of freedom: Random reflections at c-space obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3318–3323, May 1994.

[125] T.Kesavadas and H.Subramanium. Flexible virtual tools for programming robotic finishing operations. *The Industrial Robot*, 25(4):268–275, 1998.

[126] T.Lozano-Perez. The design of a mechanical assembly system. *Unpublished M.Sc. thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA*, 1976.

[127] T.Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computing*, C32:108–120, February 1983.

[128] T.Takahashi. Time normalisation and analysis method in robot programming from human demonstration data. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 37–42, Minneapolis, Minnesota, USA, April 1996.

[129] T.Takahashi and H.Ogata. Robotic assembly operation based on task-level teaching in virtual reality. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1083–1088, 1992.

[130] U.Rembold and K.Hormann, editors. *Languages for Sensor-Based Control in Robotics*, volume 29 of *NATO ASI Series F: Computer and Systems Sciences.* Springer-Verlag, 1986.

[131] W.A.Fuller. *Measurement Error Models.* Probability and Mathematical Statistics. Wiley, 1987.

[132] Wind River Systems, 1010 Atlantic Ave, Alameda, CA 94501-1147. *VxWorks reference manual 5.1*, 1993.

[133] J.G.Neugebauer W.M.Strommer and T.Flaig. Transputer-based virtual reality workstation as implemented for industrial robot control. In *Informatique '93. 2nd International Conference on Interface to Real and Virtual Worlds. Proceedings and Exhibition Catalogue*, pages 137–146, Montpellier, France, March 1993.

[134] W.O.Troxell and J.A.Davis. Task-achieving modules in robot programming. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1259–1261, July 1992.

[135] M.C.Nechyba Yangsheng Xu. Human skill transfer: neural networks as learners and teachers. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 314–319, Pittsburgh, PA, USA, August 1995.

[136] M.C.Nechyba Yangsheng Xu. On the fidelity of skill models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, USA, April 1996.

[137] M.C.Nechyba Yangsheng Xu. Human control strategy: abstraction, verification, and replication. *IEEE Control Systems Magazine*, 17(5):48–61, October 1997.

[138] Y.Xu and J.Yang. Towards human-robot coordination: skill modeling and transferring via hidden markov model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1906–1911, Nagoya, Aichi, Japan, May 1995.

[139] K.Arakawa Y.Yanagihara, T.Kakizaki and A.Umeno. Task world reality for human and robot system- a multimodal teaching advisor and its implementation. In *Pro-*

ceedings of IEEE International Workshop on Robot and Human Communication, pages 38–43, Tsukuba, Japan, Nov 1996.

[140] Jiong Zhang and Yangsheng Xu. Modeling human strategy in controlling light source. In *Proceedings 1999 IEEE International Conference on Robotics and Automation*, volume 4, pages 3140–3145, Detroit, MI, USA, May 1999.

[141] M.Beitler-S.Chen D.Chester Z.Kazi, M.Salganicoff and R.Foulds. Multimodal user supervised interface and intelligent control for a rehabilitation robot. In *Proceedings of IJCAI Workshop on Developing AI Applications for the Disabled*, pages 46–58, Montreal, Canada, 1995.

[142] M.Salganicoff-S.Chen D.Chester Z.Kazi, M.Beitler and R.Foulds. Multimodally controlled intelligent assistive robot. In *Proceedings of 16th annual RESNA Conference*, pages 348–350, Salt Lake City, Utah, 1996.