



UNIVERSITÀ DEGLI STUDI DI PARMA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

PARMA(I) - VIALE DELLE SCIENZE
TEL. 0521-905800 • FAX 0521-905758

Dottorato di Ricerca in Tecnologie dell'Informazione
XVIII Ciclo

Jacopo Aleotti

ROBOT PROGRAMMING BY DEMONSTRATION
IN VIRTUAL REALITY

DISSERTAZIONE PRESENTATA PER IL CONSEGUIMENTO
DEL TITOLO DI DOTTORE DI RICERCA

GENNAIO 2006

© Jacopo Aleotti, 2006

Contents

Introduction	1
1 Advanced robot programming	5
1.1 Human-robot interaction	5
1.2 Robot programming by demonstration	7
1.3 Related work	10
1.4 Virtual reality	12
2 System architecture	17
2.1 Virtual reality devices	17
2.1.1 Immersion CyberGlove	17
2.1.2 Polhemus FasTrak	20
2.2 VR programming	22
2.2.1 Kinematics modeling	22
2.2.2 VRML	24
2.2.3 OpenGL	26
2.2.4 Virtual Hand Toolkit	27
2.3 Software architecture	33
2.3.1 The robot library	36
2.3.2 Communication infrastructure	38

3	One shot learning of assembly tasks	43
3.1	Assembly tasks with fixed base manipulation	44
3.1.1	Demonstration interface	44
3.1.2	Task recognition	45
3.1.3	Task generation	46
3.1.4	Task simulation	47
3.1.5	Task execution	48
3.1.6	Experiments	49
3.2	Assembly tasks with mobile manipulation	52
4	Trajectory Reconstruction and Stochastic Approximation	59
4.1	Introduction to trajectory learning	60
4.2	Related work	61
4.3	The trajectory learning technique	64
4.3.1	Trajectory clustering	65
4.3.2	HMM-based trajectory evaluation	66
4.3.3	Trajectory approximation	69
4.4	Experiments	72
5	Grasp Recognition for Pregrasp Planning by Demonstration	83
5.1	Introduction to robot grasping	84
5.2	Related work	88
5.3	Grasp recognition in virtual reality	89
5.3.1	Grasp classification	89
5.3.2	Experiments	94
5.4	Pregrasp planning by demonstration	97
5.4.1	Grasp mapping	97
5.4.2	Pregrasp planning	100
6	Evaluation of Virtual Fixtures	105
6.1	Virtual Fixtures	105

CONTENTS

iii

6.2	Related work	107
6.3	Methods and protocols	109
6.3.1	Feedback type	109
6.3.2	Peg-in-hole task	110
6.3.3	Index of difficulty and Fitts' law	114
6.3.4	Subject protocol	116
6.4	Experimental results	117
6.4.1	Relative effectiveness of virtual fixtures	117
6.4.2	Combining multiple virtual fixtures	124
6.4.3	Users' remarks	126
6.5	Predicting human performance	127
6.6	Discussion	129
	Conclusions	131
	Appendices	133
	A Parametric Curves	135
A.1	OpenGL NURBS Interface	138
A.2	Nurbs++ library	139
	B Learning of Hidden Markov Models	141
B.1	Forward-Backward algorithm	142
B.2	Viterbi algorithm	143
B.3	Baum-Welch algorithm	144
B.4	Multiple observation sequences	146
B.5	Continuous observation densities in HMMs	147
	Bibliography	150

Introduction

Manually programming a robot is a complex and tedious job. It would be much more convenient to just show the robot what to do, and for the robot to learn and imitate the required behavior automatically. This type of learning is often known as "Programming by Demonstration", hereafter called PbD.

Robot programming by demonstration has emerged as a new method to reduce the efforts required in traditional robot programming. Reducing the necessary knowledge and skills of the programmer is obviously an essential requirement for the development of practical robot systems. It is known, indeed, that the market of service robotics is growing rapidly and that robots in the near future will be mainly employed at home for the support of disabled and elderly people. The acceptance of such systems will depend on several conditions. An important factor will be the cost of maintenance but, most of all, the costs arising for the daily adaptation to the environment. If each change in the routines and abilities will require special trained people, with detailed knowledge of robot programming, this will certainly lead away potential users.

The robot programming by demonstration paradigm is therefore achieving a great interest in robotics research. The main challenges that must be solved for the development of PbD systems are the implementation of natural user interfaces for human robot interaction, and the investigation of effective machine learning strategies for robot imitation.

Natural user interfaces are essential to simplify the workload of the user. Tradi-

tional ways for robot instruction must be replaced by advanced interaction schemes and multimodal communication infrastructures between the user and the robot. Multimodal interfaces allow different modes of expression such as verbal, gestural and even tactile. Moreover, the design of simple user interfaces requires the use of non intrusive sensor devices. Passive sensors, such as cameras, are therefore preferable. However, high precision tracking of the human body can be required, especially for performing assembly tasks. To this purpose the use of equipment like gloves, pose sensors, and even more general wearable devices is beneficial for advanced PbD systems.

Learning by imitation is one of the most important mechanisms, in both biological and artificial agents, for transferring knowledge and skills. Several fundamental problems arise in the context of imitation learning. The very first problem is the choice of an adequate strategy for imitation. Imitation strategies vary depending on the level of granularity and the level of cognition. Moreover, the more the demonstrator and imitator embodiments are different, the more the learning problem becomes complex. To solve the problem of mapping between dissimilar embodiments, imitation metrics must be defined. The basic idea is that a computer program analyzes the actions of a demonstration and reproduce them on a robotics platform exploiting some a priori knowledge about the task.

The most general way to put into practice the PbD concept is by letting the user demonstrate the task in the real world, while taxing the system with the requirement to understand and replicate it. However a recurrent problem, in PbD, as well as in any area of robotics, is perception. Indeed, the level of perception in complex or partially observable environments is often poor and inadequate to build a model of the environment. To circumvent this problem, PbD systems have been applied in highly engineered demonstration environments, which usually constrain the number and type of objects involved in the task. An alternative strategy to robot programming by demonstration in a real workspace is to transfer the execution of the demonstration of a task in a virtual environment. Performing the demonstration in a virtual environment provides some functional advantages which can decrease the time and fatigue

required for demonstration and improve overall safety. The focus of this thesis is the investigation of virtual reality technologies applied to PbD.

Contributions of the Thesis

The main contribution of this thesis is the development of a robot PbD system for assembly tasks where the demonstration phase is carried out in a virtual environment. The system has a modular architecture and exploits advanced devices for virtual reality such as a data glove and an electromagnetic tracker. It comprises different functional modules for managing all the components that have been developed. The system targets basic dextrous assembly tasks which can be performed by novice users without any prior knowledge about robot programming. The underlying complexity of the PbD architecture is made transparent to the human operator by means of a simple and natural user interface. More specifically, the most important contributions of the thesis are the following:

- the development of a user friendly multimodal interface for PbD exploiting advanced graphical, haptic, and communication software libraries;
- the proposal of a one-shot learning strategy for acquiring manipulation tasks in virtual reality for both fixed base robot manipulators and mobile manipulators;
- the development of a reusable library for different robotics components (robot arms, mobile robots and robot hands);
- the proposal of an algorithm for trajectory reconstruction and stochastic approximation of the operator's motions;
- the investigation of the grasp recognition problem in virtual reality and the realization of a pregrasp trajectory planner based on human demonstration;
- the evaluation of the benefits provided by the use of virtual fixtures for user assistance.

Organization of the Thesis

The organization of the thesis follows the subdivision of the system into his different functional modules. Chapter 1 provides an introduction to the problem of advanced robot programming. In particular, the chapter illustrates the fundamentals concepts of human-robot interaction, robot programming by demonstration and virtual reality.

The system architecture is presented in chapter 2 focusing on the adopted virtual reality devices and on the main software components of the system. In particular, the software tools which enable the virtual reality system (for demonstration and simulation) are described in detail together with the communication infrastructure.

The core of the developed PbD system is the one-shot learning strategy for automatic acquisition of the demonstrations provided by the user. This module, which is described in chapter 3, is a task level procedure for analyzing and segmenting the demonstrations.

Chapter 4 presents the proposed approach for trajectory learning. The method allows the system to fit datasets containing one or multiple example trajectories. It exploits a trajectory clustering algorithm, a Hidden Markov Model-based stochastic procedure for evaluation of the consistency of the demonstrated paths, and a trajectory reconstruction algorithm for curve approximation.

Chapter 5 describes the grasp recognition module and the pregrasp trajectory generator for the PbD system.

Chapter 6 addresses the problem of evaluating a set of virtual fixtures (implemented using visual, auditory, and tactile sensory feedback) in the demonstration environment, focusing on a peg-in-hole task. A final chapter summarizes this dissertation. The thesis also includes two appendices. Appendix A introduces the basic concepts about parametric curves and, in particular, it provides the formal definition of NURBS (Non-Uniform Rational B-Spline) curves and illustrates their properties. Appendix B contains a survey of Hidden Markov Models. In particular, discrete and continuous HMM are introduced together with the learning algorithms for both single and multiple observation sequences.

Chapter 1

Advanced robot programming

This chapter introduces the main topics of this thesis. A classification of robot programming systems based on different levels of human-robot interaction is first proposed. Afterward, the robot programming by demonstration paradigm is discussed in detail. An overview of the literature about robot programming by demonstration is then presented. The chapter closes with an brief introduction to virtual reality and to robot programming in virtual environments.

1.1 Human-robot interaction

Human-robot interaction (HRI) is a subset of the field of human-computer interaction (HCI). HCI has been defined by the Curriculum Development Group of the Association of Computing Machinery (ACM), Special Interest Group on Computer-Human Interaction (SIGCHI) as "a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them". HRI is a particular form of HCI where a human interacts with a robot.

HRI is an emerging research topic aimed at simplifying robot programming especially in the context of service robotics, where end users with little or no specific

expertise might be required to program robot tasks. The personal robotics market is growing rapidly; it serves the consumer market in many diverse segments such as home automation and domestic service robots (*e.g.* robotic vacuum cleaners and home security robots), education and entertainment robots. Effective human-robot interaction is an essential requirement for such kind of applications.

An important aspect of HRI is multimodality. A user interface for a HRI system is multimodal if multiple input-output sensor modalities are available to the user. Usually, multimodal interfaces provide benefits in terms of interaction, *i.e.* they are easy to learn, to use, and more natural if compared to traditional robot interfaces. For example, multimodal interfaces can speed up the execution of a task and reduce the number of failures.

Robotic systems, based on human-robot interaction, can be further classified according to different levels of autonomy and type of interaction. The level of autonomy measures the amount of human intervention, while the type of interaction refers to the level of shared interaction among humans and robots. The most general type of interaction occurs when a group of humans interacts with a group of robots. The focus of this thesis is to the simplest case of interaction that occurs when a single human interacts with a single robot.

Traditional robot-level textual programming systems do not require any form of interaction. Robot programs are hand-coded and then loaded into the robots afterwards. Automatic learning is a fundamental component for such systems. However, the development of fully autonomous robot systems is still a challenging task, due to the difficulty of learning from a self exploration of the environment and to the complexity of modeling unstructured and unknown environments. Hence, a promising solution to improve and speed up learning capabilities of robot systems is through HRI.

A primitive form of interaction, especially in industrial applications, is robot guiding. In such systems the robot is manually guided through a sequence of operations which are then stored and replicated indefinitely. Teleoperation is a direct form of robot programming, where a human operator remotely controls a robot plat-

form by means of a master input device [1, 2, 3]. The basic structure of a teleoperation system requires a constant interaction between the user and the robot. More advanced schemes exist, for example the shared control paradigm, which allows a greater degree of autonomy.

Automatic programming or task-level programming is another strategy for robot programming which requires interaction. The end user of an automatic programming based system specifies high level actions to be performed by the robot by means of advanced programming interfaces which may include gestures and speech recognition.

Robot programming by demonstration is the most intuitive method for robot programming and it will be described in detail in the following section. In a robot programming by demonstration system the user interacts with the robot by showing the correct way in which a task must be performed while the robots learns how to imitate the given demonstration.

1.2 Robot programming by demonstration

As discussed in section 1.1 the current trend towards service robotics requires the development of simple programming techniques. Programming by Demonstration (PbD) has emerged as one of the most promising solutions for effective programming of robot tasks [4, 5].

The PbD tenet is to make robots acquire their behaviors by providing to the system a demonstration of how to solve a certain task, along with some initial knowledge. A PbD interface then automatically interprets the user's demonstration, thus eliminating the need for alternative, explicit programming techniques. PbD interfaces commonly include haptic devices for gesture recognition, such as joysticks or more advanced instrumentation which may include exoskeletons, gloves and wearable devices, locomotion interfaces and full body tracking systems.

Providing a demonstration of a task to be reproduced by others is an effective means of communication and knowledge transfer between people. However, while

PbD for computer programming has achieved some success [6], teaching tasks involving motion of physical systems, possibly in dynamic environments, directly addresses the well-known difficulties of embodied and situated systems [7]. Hence, PbD is retrieving a great interest in the robotics community.

There are a number of ways in which PbD systems can be classified. An important aspect to distinguish PbD system is the imitation mechanism. The imitation mechanism can be pursued at different levels of granularity. At each level of granularity corresponds a level of cognition. In particular, the more the level of granularity grows the more the level of cognition decreases. Figure 1.1 shows such classification, which highlights four imitation strategies. The higher level of granularity is achieved by an imitation strategy aimed at learning basic primitives of motions. Primitives are also often defined as basic skills [8, 9, 10]. Skill learning is challenging because it is generally nonlinear, time-variant and non-deterministic. The skill learning process can be generalizable and decomposable.

Trajectory learning is at the second level of granularity. There are several reasons why human trajectory imitation is beneficial to Human-Robot Interaction. First, learning drastically simplifies robot programming, since demonstrating a trajectory is much simpler than programming it off-line or having a planning system discover one automatically. Indeed, human trajectories often encode a wealth of potentially useful information, hard to specify otherwise. Such information includes collision free paths, regions of space to be avoided, and preferential directions to approach objects. Qualitatively different paths of the hand can also emerge in multiple demonstrations of the same task, thereby providing multiple solutions to choose among during task execution.

A second beneficial effect of trajectory learning is that human trajectories provide implicit information about the available free space, and hence help in assessing the required tightness of fitting [11]. Speed recommendations could also be extracted from human demonstrations, highlighting difficult segments to be executed at low speed along with other less critical motions amenable to execution at higher speed. Furthermore, zero velocity points allow segmentation of different parts of the task.

A third implication of human motion imitation concerns safety in HRI. Humans are known to effect motions with peculiar characteristics [12], and hence the actual prediction of a biological motion is essential for true mimicry of human behavior. Learning human trajectories is thus a step towards improving robot behavior predictability (and hence user safety) when robots operate in human-inhabited environments. In summary, there are a number of reasons why reproducing a human trajectory can be beneficial to a robot system and can simplify its deployment in a service application. Conveying all this information via a conventional programming interface would be very difficult if not impossible.

At higher level of cognition imitation can be achieved with task level learning approaches [4, 13, 14, 15]. An example of task level learning is one-shot learning, which stands for immediate learning from a single demonstration. In particular, one shot learning can be applied to recognize sequences of assembly operations by observing changes in the state configuration of the objects in the environment. Finally, implicit imitation, i.e learning by following, has the lowest level of granularity.

Another possible way to classify PbD systems depends on the demonstration environment. The most straightforward way to put into practice the PbD concept is by letting the user demonstrate the task in the real world, while taxing the system with the requirement to understand and replicate it. Recent examples of PbD systems involving demonstration in the real world are [14] and [5]. This is also the most general approach to programming by demonstration, but the complexity of the underlying recognition and interpretation techniques strongly constrains its applicability. To circumvent this problem, a PbD system might require to restrict the objects and actions involved in the task to be demonstrated to a predefined set, or set up a highly engineered demonstration environment. However, if objects and actions occurring in the task are constrained in number and type, the same *a priori* knowledge can be transferred into a virtual environment. The focus of this thesis is the investigation of virtual reality technologies applied to PbD.

Performing the demonstration in a virtual environment provides some functional advantages which can decrease the time and fatigue required for demonstration and

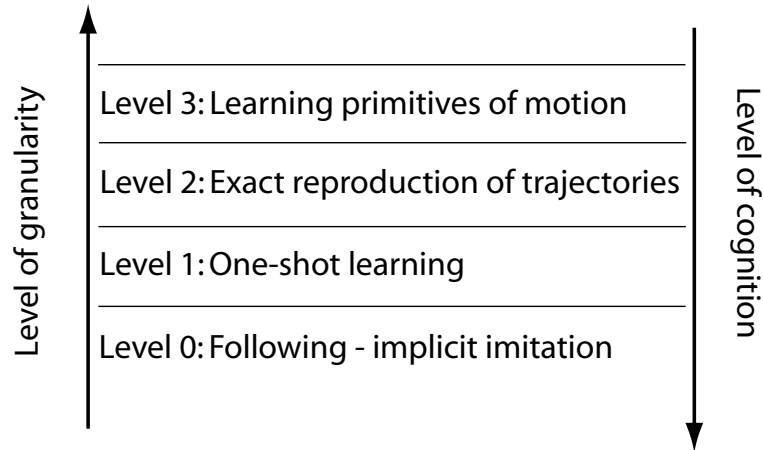


Figure 1.1: Imitation mechanisms.

improve overall safety by preventing execution of incorrectly learned tasks. For example human hand and grasped object positions do not have to be estimated using error-prone sensors like cameras. Moreover the virtual environment can be augmented with operator aids such as graphical or other synthetic fixtures [16], and force feedback. Of course, these functional advantages should be weighed against the very drawback of a virtual environment, namely its need for advance explicit encoding of a priori knowledge about the task, which restricts the applicability of the approach. Previous works evaluating PbD in virtual environments include [17, 18, 19], which will be examined in detail in the next section.

1.3 Related work

This section reviews some of the most important contributions to robot programming by demonstration. More specific references will be provided in the following chapters.

Takahashi et al. [17, 18] were the first authors to propose the paradigm of robot programming by demonstration in virtual reality. The proposed system comprised a

VPL DataGlove and was aimed at performing basic manipulation tasks. Assembly operations were recognized at the task level and interpreted using task-dependent information. Experiments showed that beginner users were able to program a sequence of routines without any prior skill of robot programming. Additional contributions are the work of Lloyd et al. [19], who proposed a demonstration system for executing contact tasks in a virtual environment, and the work of Kawasaki et al. [20]. In the latter, the authors presented a teaching system for multi-fingered robots in virtual reality. The operator used a force-feedback glove to manipulate objects. Actions were segmented and represented as primitive motions. The performance of the system were tested in a simulated environment for pick-and-place tasks by a 5-fingered robot.

In their seminal work [4], Ikeuchi and Suehiro proposed the Assembly Plan from Observation (APO) system. The APO system observes assembly tasks performed in a real environment by an operator, recognizing polyhedral objects and relations among them. Assembly relations are defined by face contacts between manipulated objects and environmental objects. Further research, by the same authors, has focused on the extraction and recognition of essential interactions between multiple demonstrations of the same task [21, 22, 14]. Essential interactions are identified by extracting attention points between input data collected from data gloves and a vision system. The analysis of the demonstrations consists of two phases. In the first phase actions are segmented and attention points are extracted. In the second phase the system closely examines human demonstrations around attention points to identify attribute values. The advantage of this approach is that slightly different demonstrations of the same task can be combined and generalized.

Ehrenmann et al. [23, 24] proposed a complete multisensor system for PbD. Data glove values are processed by neural networks for grasp recognition and visual tracking of the human hand is done by active contours. The system has been tested both in a simulated environment and in a real workspace on a 7 degrees of freedom robot manipulator with a Barrett hand. In [25] Zöllner and Dillmann improved the system to handle two hand manipulation tasks using multiple probabilistic hypothesis.

Hovland et al. [26] proposed a system for skill acquisition using Hidden Markov Models. Assembly skills are represented by a discrete event controller whose states correspond to the states of the Hidden Markov Model. The output of the controller provides the commands for the robot.

Amit and Matarić [27] presented an architecture for learning movements sequences for humanoid robots. The architecture is based on neuroscience concepts such as mirror neurons. The proposed imitation learning strategy was developed on multiple levels of movements abstractions.

Billard et al. [28] focused on the problem of finding optimal imitation strategies. The imitation process is modeled as a hierarchical optimization process. General metrics were proposed to optimize task reproduction.

Drumwright et al. [9] presented a unified methodology for learning motor primitives for humanoid robots. A primitive model for a behavior, for example a tennis forehand stroke, was constructed by interpolating examples of joint angle trajectories.

Schaal et al. [29] proposed a real-time statistical learning algorithm for humanoid robots. The method is based on the probabilistic locally weighted learning technique. The algorithm approximate nonlinear functions by means of piecewise linear models. The method was successfully applied for different applications such as learning full-body inverse kinematics and dynamics.

1.4 Virtual reality

Virtual reality (VR) can be defined as a high-end user-computer interface that involves real-time simulation and interactions through multiple sensorial channels (Burdea et al. [30]). Sensorial communication between the user and the VR system can be achieved through vision, sound, touch, smell and taste. According to Burdea there are three main features that characterize a virtual reality system. These features are also known as the three I's of virtual reality. The first "I" refers to interaction. The synthetic world must respond to the user's input by modifying the virtual world in a dynamic way. The second "I" is immersion. Immersion is the feeling of being present

into the computer generated virtual world. Immersion-Interaction is a twin element and denotes the VR users' capability to interact with the simulated world scenarios through all human sensory and input and output channels. The third "I" of virtual reality is imagination. Imagination has some artistic undertone and it essentially relates to the human imaginative inventiveness to find valuable applications for VR technology.

The first workstation for virtual reality applications was released in the early 1960's and it was named the "Sensorama Simulator". The setup included stereo sound, integrated with the full 3-D camera views. The viewer could ride a motorcycle while sensing the wind, simulated by a fan. Around the 1970' researchers realized the possibility of using computer-generated images instead of analog images taken by cameras. In the same period NASA began research on using this technologies for space flight, and later, moon landings. The pivotal convergence of technologies that have made Virtual Reality possible have come about in the last fifteen years. The last ten years have seen advancements in areas that are absolutely crucial to the VR paradigm. These include the Liquid Crystal Display (LCD), high performance image generation systems, along with tracking and sensory glove systems.

Figure 1.2 shows the main components of a classical virtual reality system. The main element is the VR engine. Interactions between the user and the VR engine are mediated by input/output devices which read user's input and generate feedback simulation results. Joysticks or trackballs are used in simple tasks, while sensing gloves are used for haptic interactions. Typically, advanced feedback from a VR engine is generated by stereo head-mounted displays (HMD's) or large-volume displays such as the CAVE [31]. Simpler applications make use of standard monitor displays. Such systems, like the one presented in this thesis, belong the category of "Fish Tank VR" [32]. In a Fish Tank VR system the virtual 3D scene is obtained by coupling head position with respect to a monitor to the 3D displayed image so that the correct perspective view is obtained. In the VR system proposed in this thesis the virtual environment is not directly coupled with the orientation of the head of the user. The user can change the position of the virtual camera by means of a standard computer

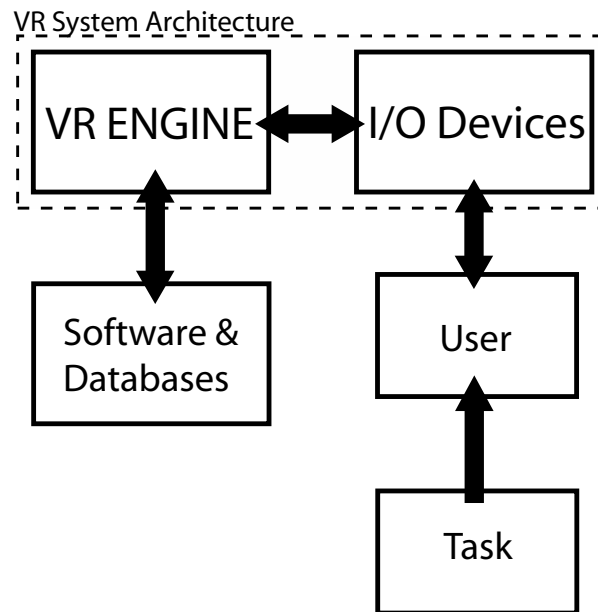


Figure 1.2: Components of a VR system.

mouse. The VR engine responds to user's commands by changing the state and the view of the virtual world. The synthetic world is modeled off-line using dedicated software libraries and databases. The models have to be rendered in real time. Another computational load for the VR engine is related to object dynamics, collision detection and contact force computation for interaction between the virtual objects.

Two important aspects that must be also taken into account while developing a VR system are the screen update rate and the lag in position sensing. To create a correct effect of illusion and to achieve good performance it is important to keep the refresh rate high (more than twenty updates per second are usually required). Ware et al. [33] investigated the influence of lag and frame rate on reaching tasks in Fish Tank virtual reality. They found that the lag in head tracking is less important than the lag in hand tracking and that low frame rates cause a degradation in performance.

VR technologies have been adopted in many fields. Traditional VR applications

range from medical applications [34], education, arts, to entertainment and manufacturing [35]. VR is bringing large benefits to the medical community, despite limitations such as the lack of standards and the cost of the devices required for advanced systems. These advantages involve medical training, diagnosis, endoscopic examinations, open surgery and minimally invasive surgery and rehabilitation. There are also numerous emerging fields where VR technologies have been successfully applied, such as robotics for robot programming (which is the topic of this thesis), robot teleoperation and supervisory control, but also virtual prototyping, assembly verification, ergonomic analysis and training of personnel.

Augmented reality (AR) [36, 37, 38] is another growing field of research which deals with the combination of real world and computer generated data. Usually VR environments are very simplistic and systems that can create more realistic environments are very expensive. AR is a possible solution that presents to the user a combination of the real scene, viewed by the user, and a virtual scene, generated by the computer, that augments the scene with additional information. Overlaying the shape of a pedestrian on a car display is an example of useful AR.

Augmented reality systems can be classified into two types: computationally context-free and computationally context-aware. In context-free systems the user sees-through head mounted display with the computer image projected in front of him. The context-aware systems overlay graphics or other media onto a real image by sensing the context in which it finds itself.

Chapter 2

System architecture

This chapter introduces the developed programming by demonstration system. Section 2.1 describes the virtual reality devices used in the thesis. Section 2.2 focuses on the main software packages and libraries used in the design and the development of the system. In particular, the adopted computer graphics packages are introduced together with the Virtual Hand Toolkit library. Section 2.3 provides the details about the PbD software architecture and the communication infrastructure.

2.1 Virtual reality devices

This section introduces the two VR devices used in this thesis. The first is the CyberGlove by Immersion Corporation [39], the second is the Polhemus Fastrak [40].

2.1.1 Immersion CyberGlove

The CyberGlove is a tactile feedback instrumented glove with 18 resistive sensors for joint-angle measurements. The glove comprises

1. two bend sensors on each finger;
2. four abduction sensors;

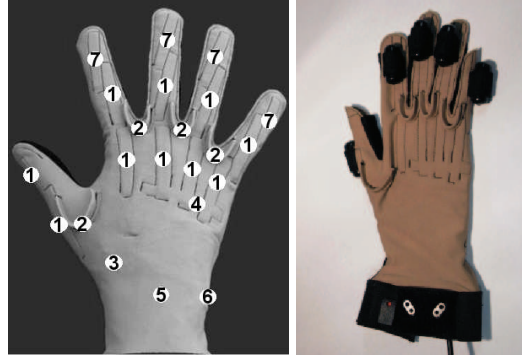


Figure 2.1: CyberGlove sensors (left image) and CyberTouch device (right image).

3. one sensor for thumb crossover;
4. one sensor for palm arch;
5. one sensor for wrist flexion;
6. one sensor for wrist abduction.

The resolution of the resistive sensors is 0.5° and the accuracy is 1° . The complete technical specifications of the glove are shown in table 2.1. The angle of the distal joints of each of the four fingers (pinkie, index, middle, ring) are estimated via software by coupling the distal joints with the proximal joints as the device does not provide individual bend measurements for these four degrees of freedom. A 22-sensor model glove exists with three flexion sensors per finger.

Figure 2.1 shows on the left side a 22-sensor model CyberGlove and the position of the sensors. The number of each sensor corresponds to the previous enumeration. Each finger, as explained above, has an additional distal flexion sensor indicated with number 7. An accurate analysis of the accuracy of the CyberGlove can be found in [41].

The glove is connected to an interface unit called CGIU (*CyberGlove Instrumentation Unit*) which contains the electronics for sensor reading. The CGIU is connected to the host pc through a RS232 serial line.

Parameter	Value
Resolution	0.5 degrees
Sensor Repeatability	1 degree
Sensor linearity	0.6%
Sensor Data Rate	150 records/sec
Interface	RS-232 (115.2 kbaud)

Table 2.1: CyberGlove technical specifications.

The CyberGlove uses linear bend sensors. The sensors incorporate thin electrical strain gauges placed on an elastic nylon blend material. Joint angles are measured indirectly by a change of resistance in pairs of strain gauges. Whenever a finger is under motion one of the strain gauges is under compression, while the other is under tension. The change in resistance produces a change in voltage on a Wheatstone bridge. The glove comprises as many Wheatstone bridge circuits as sensors. The differential voltages are then demultiplexed, amplified and finally digitized by A/D converter in the CGUI. The glove data are then sent to the host computer over the serial line.

The communication interface between the glove and the software is handled by a configuration utility called *Device Configuration Utility (DCU)* and by a *Device Manager (DM)*. Both the DCU and the DM are part of the Virtual Hand Toolkit developed by Immersion, which will be described in the following sections. The DCU provides an intuitive java-based interface for configuring, calibrating and testing Immersion products as well as Polhemus trackers. The DM is a smart driver providing a network transparent mechanism for interfacing the devices.

Calibration is required to adjust the gains and offsets of the raw values read from the glove which result in the Device Manager returning joint angles instead of raw sensor values. The calibration routine is quick and it is generally required for each user to improve the precision of the sensor readings as human hand sizes and range of motion vary greatly.

The actual glove used in this thesis is a CyberTouch device. The CyberTouch is a CyberGlove instrumented glove with 6 vibrotactile actuators. One vibrotactile actuator is located on the back of each finger, and one additional actuator is located in the palm. Each actuator can be individually programmed to vary the strength of vibrations. The vibration amplitude of each vibrotactile stimulator can be programmed up to $1.2N$, while the vibrational frequency can vary from 0Hz to 125Hz. Each actuator consists of a plastic capsule housing a DC electrical motor. The motor shaft has an off-centered mass, which produces vibrations when rotated. By changing the speed of rotation it is possible to change the vibration frequency. Vibrations are felt by the skin mechanoreceptors. Whenever a signal is received by the actuator, the driver unit applies currents using D/A converters and operational amplifiers. In this way the feedback loop is closed. Usually, vibrations are generated as the operator interacts with objects in a virtual environment as will be shown in chapter 6.

2.1.2 Polhemus FasTrak

The Polhemus FasTrak is a 3D motion tracking device. The FasTrak is a six degrees of freedom electro-magnetic sensor that computes the position and the orientation of a small receiver as it moves through space with respect to a fixed transmitter box. The receiver is mounted on the wrist of the glove. In the standard operating mode the receiver must be located within $76cm$ from the transmitter. Within the standard range the static accuracy of the Fastrak is $0.08cm$ RMS for receiver position and 0.15° RMS for receiver orientation. The resolution is $0.04cm$ and 0.025° . The complete specifications of the Fastrak are shown in table 2.2.

The Fastrak uses a digital signal processing (DSP) architecture and includes a *System Electronics Unit* (SEU). A digital to analog converter and amplifier allows the excitation of three transmitter antennas (located inside the transmitter box) with sinusoidal currents. The AC magnetic fields induce voltages in three orthogonal receiver coils, which are sent to coil-specific, low-noise differential amplifiers. The output from these amplifiers is multiplexed with calibration signals to three parallel analog to digital converters. Each transmitter antenna is in turn excited with a driv-

Parameter	Value
Latency	4 ms
Update Rate	120 update/sec
Static Accuracy	0.08cm RMS, 0.15° RMS
Resolution	0.04cm, 0.025°
Interface	RS-232 (115.2 kbaud)

Table 2.2: Fastrak technical specifications.



Figure 2.2: The system devices with electronics (left image) and the operative setup (right image).

ing signal. Each excitation produces a pattern of three linearly independent vectors, one for each receiver coil. Nine measurements for each measurement cycle are collected. The SEU takes these nine signal measurements each cycle and calculates the six position and orientation values.

Figure 2.2 shows the CyberTouch and the Fastrak devices, including the CGIU, the transmitter, the receiver and the SEU. The same figure shows the operative setup of the system where an operator, wearing the glove, is performing a manipulation task in a virtual environment displayed on the screen of the host computer.

2.2 VR programming

In the previous section the input devices used in the PbD system have been presented. These devices enable simulation in virtual reality. Additional key aspects for the development of a VR system are the modeling of the virtual world and the kinematic mapping of the input devices to the simulation scene. These tasks are usually solved by VR toolkits and computer graphics libraries. This section first introduces the kinematics modeling problems for simulation, then the Virtual Reality Modeling Language (VRML) is briefly introduced as well as the OpenGL graphics library. Finally the Virtual Hand Toolkit (VHT) by Immersion is described in detail.

2.2.1 Kinematics modeling

The kinematics modeling problem determines the location of the 3D objects with respect to a world system of coordinates as well as their motion in the virtual world. Object kinematics is governed by parent-child hierarchical relations, with the motion of a parent object affecting that of its child. Another aspect of kinematics modeling is the way the world is view through a virtual camera and it will be discussed in section 2.2.3. Homogeneous transformation matrices are used to express object translations, rotations and scaling. A homogeneous transformation matrix is given by the generic equation

$$\mathbf{T}_{A \leftarrow B} = \begin{bmatrix} \mathbf{R}_{3 \times 3} & \mathbf{P}_{3 \times 1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.1)$$

where $\mathbf{R}_{3 \times 3}$ is the rotation submatrix expressing the orientation of system of coordinates B with respect to system of coordinates A , and $\mathbf{P}_{3 \times 1}$ is the vector expressing the position of the origin of system B with respect to the origin of A . Homogeneous transformation matrices offer computational advantages as they can treat both rotations and translations in a uniform way. They can be compounded and they are easily invertible.

One of the objects seen in the virtual environment is a virtual hand that the op-

erator can control in real time by moving his own hand while wearing the glove. The position and the orientation of the hand in space is computed exploiting the information collected by the 3D tracker attached to the glove. The tracker receiver's position relative to the source (transmitter) is given by the time-dependent homogeneous transformation matrix $\mathbf{T}_{transmitter \leftarrow receiver}(t)$. Assuming that the transmitter is fixed, then its position in the virtual reality world system of coordinates (W) is given by the fixed matrix $\mathbf{T}_{W \leftarrow transmitter}$. Thus the virtual hand can be moved in the virtual world by multiplying its vertices with a compound matrix

$$V_i^W(t) = \mathbf{T}_{W \leftarrow transmitter} \mathbf{T}_{transmitter \leftarrow receiver}(t) V_i^{hand} \quad (2.2)$$

where V_i^{hand} are the vertex coordinates in the hand system of coordinates, and $i = 1, \dots, n$.

If the virtual world contains other objects that can be grasped by the virtual hand, once one object is grasped its position does not change with respect to the receiver, and thus $\mathbf{T}_{receiver \leftarrow object}$ is invariant until the object is released. Therefore the object's position in the virtual world is given by

$$\mathbf{T}_{W \leftarrow object}(t) = \mathbf{T}_{W \leftarrow transmitter} \mathbf{T}_{transmitter \leftarrow receiver}(t) \mathbf{T}_{receiver \leftarrow object} \quad (2.3)$$

So far it has been assumed that the object in the virtual environment are monolithic, meaning that, for example, no object hierarchies were defined to describe complex shapes such as the hand. By means of a proper object hierarchy the virtual hand can be decomposed into different subgroups and the fingers can be moved independently by the glove readings. Object hierarchies define groups of objects which move together as a whole, but whose parts can also move independently. A hierarchy implies multiple levels of virtual objects. The higher level objects are called the parent objects and the lower level is formed of children objects. Motion of a parent object is replicated by that of all its children. However, a child object can move without affecting the parent object's position. Graphically the hierarchy can be represented as a tree graph. Its nodes are object segments and its branches represent relationships. The

power of hierarchical structures stems from their support by most graphics libraries such as VRML or the VHT.

The human hand model can be seen as a hierarchy of a palm parent node with five children representing the fingers. Each finger is further decomposed into a hierarchy of substructures representing phalanges. For example, it is thus possible to determine the position of the fingertip of the index finger in the world system of coordinates

$$\begin{aligned} \mathbf{T}_{W \leftarrow fingertip}(t) = & \mathbf{T}_{W \leftarrow transmitter} \mathbf{T}_{transmitter \leftarrow receiver}(t) \\ & \cdot \mathbf{T}_{receiver \leftarrow 1}(t) \mathbf{T}_{1 \leftarrow 2}(t) \mathbf{T}_{2 \leftarrow 3}(t) \mathbf{T}_{3 \leftarrow fingertip} \end{aligned} \quad (2.4)$$

where $\mathbf{T}_{i \leftarrow j}(t)$ represents the homogeneous matrix between subsequent nodes of the finger such as the first finger link, the interphalangeal link and the distal link. To complete the discussion if a virtual camera is introduced in the system, the global transformation matrix must be introduced. It expresses the world system of coordinates with respect to the camera system of coordinate (*camera*)

$$\mathbf{T}_{camera \leftarrow fingertip}(t) = \mathbf{T}_{camera \leftarrow W}(t) \mathbf{T}_{W \leftarrow fingertip}(t) \quad (2.5)$$

2.2.2 VRML

Virtual environments are created using geometric modeling tools such as graphical editors or 3D digitizers. Editors allow to export object models into different graphics file formats which can be imported into virtual reality systems. In particular, VHT supports the Virtual Reality Modeling Language (VRML) file format. A VRML parser loads the input data files and builds the corresponding virtual environment as described in section 2.2.4. VRML is a scene description language which is human readable. VRML has been developed to be used over the internet and parsed by a plug-in for a web browser. The language has numerous capabilities such as built in geometric primitives including face sets and solids, and editing features for lights and materials. VRML provides also advanced capabilities for stand-alone applications, which have not been used in this thesis, such as spatialised sound, viewpoints and navigation methods, event handling and routing. The VRML scene graph is composed of a hierarchy of nodes and routes. Shapes are the building blocks of a VRML

world. Primitive shapes can be defined for simple 3D objects such as boxes, cones, cylinders and spheres. A **Shape** node builds a shape with the following syntax

```
Shape {
    appearance Appearance {
        material Material {
            diffuseColor ...
            emissiveColor ...
            transparency ...
        }
    }
    Geometry ...
}
```

the **Appearance** is used to specify object's color, smoothness of its surface and shininess. The **Geometry** node is used to specify the geometrical structure of the object. The geometry can be specified in terms of one of the previously mentioned primitive nodes as well as with more complex geometries defined as the indexed face set. A **Transform** node is a grouping node which creates a coordinate system that is positioned, rotated and scaled with respect to the parent coordinate system. Shapes built in the new coordinate system are positioned, rotated, and scaled along with it. The **children** field of a **Transform** node includes a list of one or more nodes with the following syntax

```
Transform {
    translation ...
    rotation ...
    scale ...
    children [
        Shape { ... }
        ...
        Transform { ... }
        ...
    ]
}
```

2.2.3 OpenGL

The OpenGL (Open Graphics Library) graphics system is a software interface to graphics hardware. OpenGL is designed as a hardware independent interface which does not provide high level commands for describing three dimensional objects. OpenGL provides a set of basic primitives for computer graphics applications, such as points, lines and polygons. The OpenGL interface can be combined and integrated with a virtual reality toolkit such as VHT which provides a wrapper around OpenGL. The virtual environment built upon the high level toolkit is fully compatible with the OpenGL interface. The OpenGL interface is a state machine which can be set to various states. The current color is an example of a state variable. Other state variables control additional properties such as the current viewing and projection transformations, line and polygon patterns, drawing modes, position and characteristics of lights and material properties (which can be specified in the input VRML files as explained in the previous section).

One of the most important features of the OpenGL interface is the ability to control the viewing transformations of the virtual camera. The inverse of the matrix $T_{camera \leftarrow W}(t)$ in equation 2.5 is the view transformation. It represents the position and the orientation of the camera system of coordinates in the fixed world system of coordinates. The first stage in the OpenGL rendering pipeline is mapping of the virtual objects to camera coordinates. This is followed by lighting, perspective projection, clipping and screen mapping. The graphics pipeline processes only the subspace actually seen by the camera. This subspace is represented as a volume of space called frustum, which is a portion of a pyramid aligned with the camera system of coordinates. The frustum pyramid is defined by a near and a far rectangle. The tip of the frustum is at the origin of the camera system of coordinates, also called the center of projection. Lines from the center of projection to the 3D objects vertices intersect the viewing plane to form the object's perspective projection.

2.2.4 Virtual Hand Toolkit

The Virtual Hand Toolkit [42] is the application development component of the VirtualHand Suite 2000 by Immersion Corporation, which also includes the Device Configuration Utility and the Device Manager. A virtual reality toolkit is by definition "an extendable library of object-oriented functions designed for VR specifications. Simulated objects are parts of classes and inherit their default attributes, thus simplifying the programming task." [30]. Although VHT is not an open-source library it offers to the developers the possibility of extending the basic classes and write applications modules using the same simulation kernel. The toolkit offers high level functionalities and low level translators for the specific I/O devices.

The core of the VHT is a multithreaded architecture backed with event-based synchronization. The toolkit is divided into a set of functional components as shown in figure 2.3. VHT is built upon two main libraries `VHTDevice` and `VHTCore` and a set of support libraries such as the COSMO/Optimizer engine for display and model import, and the collision detection engines. The `VHTDevice` library provides support for the I/O devices and exceptions, while the `VHTCore` library provides support for handling scene graphs and, in particular, for the use of the virtual hand. It also provides a wrapper for the collision detection libraries and for the model import library (COSMO).

There are three basic steps that must be followed when using the VHT as the main haptic simulation loop.

1. Connecting to a physical device (for example CyberGlove and Polhemus tracker).
2. Associating the devices to a virtual hand (`vhtHumanHand` instance).
3. Creating the support environment (with the `vhtSimulation` and `vhtEngine` classes).

The `vhtEngine` is the central container for the VHT runtime context. A single instance of the `vhtEngine` is required for a VR application. The `vhtHumanHand` can be registered for automatic management and the `vhtEngine` takes care of fetching the

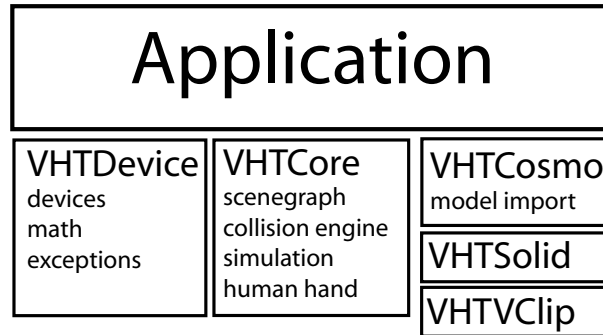


Figure 2.3: VHT library components.

latest data from the devices. VHT also requires the definition of a simulation logic for the application by means of a user defined subclass of the `vhtSimulation` class. One of the typical purposes of the simulation class is to specify the behavior of the objects when a collision event is detected (for example a collision between the virtual hand and the object to be grasped).

2.2.4.1 Device layer

VHT uses a client-server architecture where the user's application acts as the client. The physical devices reside on the server side, which is handled by the Device Manager. The physical device can reside on the local machine or on any other machine on the network. The class `vhtIOConn` describes a single device connection. An application that uses both the glove and the tracker defines two instances of `vhtIOConn` referring to predefined entries in the device registry, maintained by the DCU. To access a default device defined in the registry, the application code must use the `vhtIOConn::getDefault` method. The physical devices are abstracted with the proxy concept, implemented in VHT by the `vhtDevice` abstract base class. Each concrete class connected to a Device Manager must have a corresponding class that extends `vhtDevice`. The concrete classes defines all the methods needed to perform I/O. Any

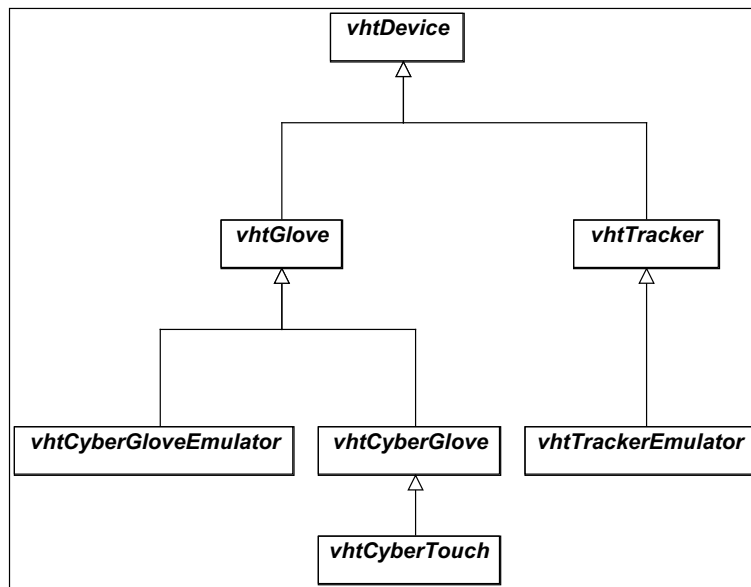


Figure 2.4: VHT device classes.

`vhtDevice` subclass must implement a constructor method that takes a `vhtIOConn` object as an argument. This constructor initiates a connection and performs any necessary handshake. Once a device's proxy has been constructed data can be acquired from the hardware. Device subclasses implement two basic data collection methods, `getData` and `update`. The first method is used to return the most recently updated data from the device, while the second method polls the hardware for the most recent data and stores it internally.

Figure 2.4 shows the class diagram of the main VHT device classes. In particular, the `vhtCyberGlove` class is the device's proxy for the Cyberglove. The `vhtGlove` and `vhtTracker` classes provide emulator classes which can be used for connecting non-existent devices or to perform advanced custom functions such as mappings from one data source to another and coordinate transformations. An emulator class behaves in exactly the same manner as the class itself, but has no device connection. The `vht-`

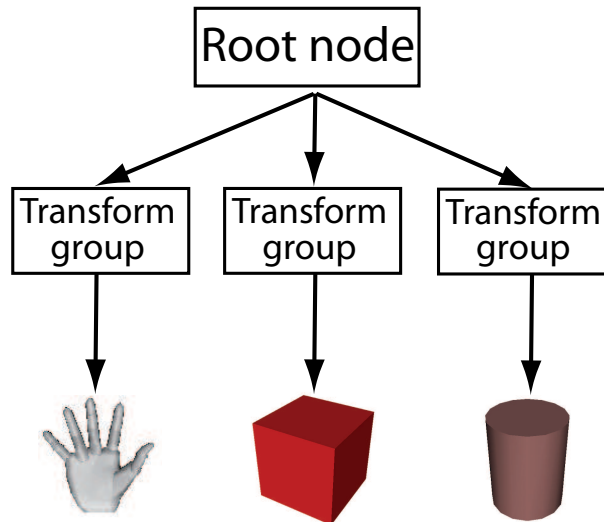


Figure 2.5: Example of Haptic Scene Graph.

`CyberTouch` class inherits from the `vhtCyberGlove` class. This class adds one overloaded method for setting the vibration amplitude of the vibrotactile actuators.

2.2.4.2 Scene graphs

VHT supports scene graphs to deal in a formal way with geometrical information describing the virtual environment. VHT relies on a mapping mechanism between different scene graphs to deal with both haptic and visual information. A haptic scene graph is a directed tree structure without cycles containing information about the geometry of the objects, coordinate transformations and grouping. Instances of the class `vhtTransformGroup` allow to store homogeneous transformation matrices. Each object has a proper coordinate transformation from the global system of coordinates that defines the basis of its local coordinate frame. The scene graph can be constructed manually or by importing external models, such as VRML files, through a model parser. To allow objects in the environment to be moved the scene graphs nodes that define transformations must be properly updated once per frame.

VHT manages three scene graphs. The Haptic Scene Graph (HSG), the Visual Scene Graph (VSG) and the Neutral Scene Graph (NSG). The HSG is primarily a collision data structure that describes the state of the shapes concerning the position, the orientation and the geometry. The VSG manages the scene graph which is built by the model parser when external nodes are imported in the simulation environment. The NSG acts as a mapping between the VSG and the HSG. It keeps the two trees synchronized during the simulation by copying the transformation matrices from the HSG to the VSG to maintain visual consistency. Figure 2.5 shows an example of an HSG. The graph contains a root node, the virtual hand and two graspable objects, a box and a cylinder. The virtual hand is in its turn a complex subtree that will be described in the following section. Each object node has a transform group parent node which can be modified to move the object in the environment.

2.2.4.3 Human hand class

VHT provides the support for the simulation of a virtual human hand. The `vhtHumanHand` class manages all data updates, kinematics calculations, graphical updates and it can draw itself in any OpenGL context. A human hand instance can be constructed with the references to both a glove and a tracker objects. Once an application has instantiated a `vhtHumanHand`, the object can be used to control and update the device's proxies. The human hand class includes a complete kinematic model of a human hand. This model provides a mapping from the actual devices into a hierarchy of homogeneous transformations representing the kinematic chain of each finger. The graph representing the hand can be manipulated as well as any other haptic scene graph. Each hand defined in the application must be registered with the `vhtEngine`. The hand haptic graph is divided into six subgraphs, one for each finger and one for the palm. Each finger is managed by the `vhtHumanFinger` class. Instances of this class contain pointers to each of the three phalanges namely the metacarpal, the proximal and the distal.

VHT provides a mechanism to allow objects in the scene to be grasped by a virtual hand. A grasp state manager encapsulates a grasping algorithm. The grasping

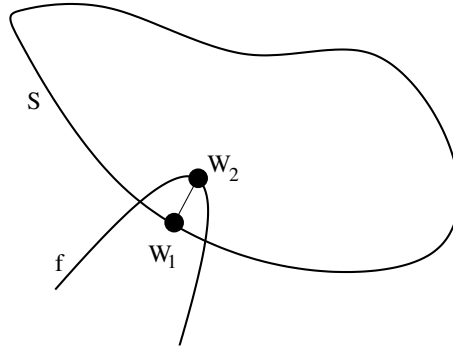


Figure 2.6: Example of a virtual finger colliding with an object.

algorithms exploits collision detection information between the hand and the objects. If the contacts provide sufficient friction, the object is attached to the virtual hand.

2.2.4.4 Collision detection

The VHT collision detection system consists of a set of optimized algorithms and techniques. The collision detection process can be divided into two steps a wide mode and a local mode. In wide mode, the collision detection algorithms globally cut all possible collision pairs to a smaller set of probable collision pairs in a conservative manner. In local mode, two shapes are compared at the actual geometry level to determine detailed contact information. Two implementations of the GJK algorithm are provided: VCLIP and SOLID. The VCLIP algorithm was used in the experimental evaluation of the system proposed in this thesis.

A collision framework is a pair of objects, a `vhtCollisionEngine` and a `vhtCollisionFactory`. The collision engine performs hierarchical culling of all geometry pairs by operating on collision data structures provided by its collision factory (*e.g.* VCLIP collision factory). The collision detection engine provides a list of pairs of colliding objects. The list of pairs are shapes in the scene graph that are within a collision epsilon distance of each other. In particular, the algorithm returns for each pair of objects the minimum translation distance, *i.e.* the smallest distance that one object must

be translated so that the two shapes collide. For non-colliding objects the minimum translation distance equals their smallest distance, but for penetrating objects it equals the deepest penetration depth. VHT provides for each pair of colliding objects two witness points and the contact normal. Figure 2.6 shows a schematic representation of a virtual finger f penetrating an object S in the environment, highlighting the two witness points W_1 and W_2 and the minimum translation distance.

2.3 Software architecture

The PbD system described hereafter handles basic manipulation operations in a 3D “block world” allowing the simulation of manipulation tasks both with a fixed base robot manipulator or with a mobile manipulator as will be shown in chapter 3. The core of the system targets task-level program acquisition. The virtual scene simulates the actual workspace and displays the relevant assembly components. The system recognizes, from the user’s hand movements, a sequence of high level actions and translates them into a sequence of commands for a robot manipulator. The recognized task is then performed in a simulated environment for validation. Finally, if the operator agrees with the simulation, the task is executed in the real environment (currently only for fixed base manipulation tasks) referring to actual object locations in the workspace. A library of simple assembly operations has been developed. It allows to pick and place objects on a working plane, to stack objects, and to perform peg-in-hole tasks.

Figure 2.7 shows the main components of the PbD testbed. The system has a client server structure. The client comprises the operator’s demonstration interface and two modules, a task planner and a task performer. The demonstration interface accepts the user’s commands which are generated by the input devices and produces a multimodal feedback for the operator. The multimodal feedback consists of different sensorial channels, the first is the visual feedback of the virtual environment used for task demonstration and task simulation, the second is a vibrotactile feedback generated by the actuators of the CyberTouch. Auditory feedback is also exploited in

the system and its use will be investigated in chapter 6. Algorithm 1 shows the procedure that manages the collisions between the virtual objects in the demonstration environment. The procedure first resets the grasp state manager, then computes all the collision pairs between virtual objects. Subsequently, for each entry the colliding objects are retrieved. If the system detects a collision between the virtual hand and an object, the algorithm checks if the grasping conditions are met, and in case constrains the objects to the hand. Alternatively, if a collision between two objects is detected, *i.e.* on object being grasped and a static object, the algorithm notifies the user about the collision with a programmable feedback (visual, vibrotactile or auditory).

The Task Planner is a software component which analyzes and interprets the demonstration performed by the user, while the Task Performer is responsible of generating the simulation of the task and the commands for the execution of the task in the real workspace. The server application is used to control the robot manipulator in the real workspace. The communication between the client and the server is handled by an high level infrastructure based on CORBA, which will be discussed in detail in section 2.3.2.

Figure 2.8 shows a partial class diagram of the PbD system highlighting the main classes. Two classes have been developed to manage the simulation environments used for task demonstration and task simulation. An instance of the `UserVirtualDemo` class manages the demonstration environment while an instance of the `UserSimulation` class manages the robot simulation of the task. The two classes both derive from the `vhtSimulation` class. In particular, the main functionalities of the `UserVirtualDemo` class include the activation and the communication with the `TaskPlanner`, which analyzes the task performed by the user in the virtual environment, the management of the collision detection factory and the generation of the vibrotactile feedback. Feedback is activated if any collision pair is detected between a graspable object in the environment and the virtual hand.

The `UserSimulation` class manages the simulation of the task in a virtual environment which comprises the 3D model of a robot arm or a more complex mobile platform in the case of mobile manipulation tasks. At each frame the new position of

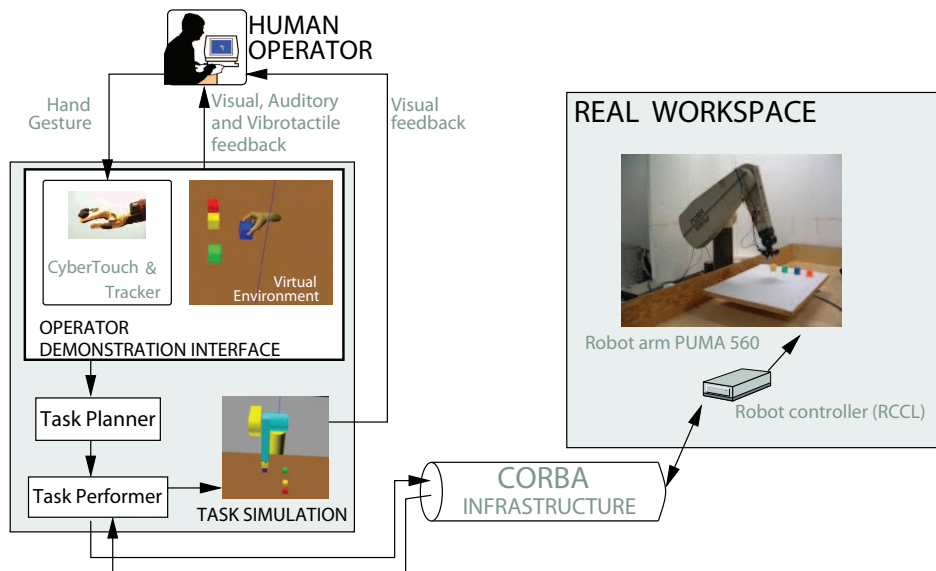


Figure 2.7: The system architecture.

```

1: Reset grasp state manager
2: pointList= collisioncheck()
3: if pointList.NumEntries > 0 then
4:   for all collision pairs do
5:     get the colliding objects
6:     if it is a hand-object collision then
7:       get contact normal
8:       get witness point
9:       generate feedback to the user
10:    if grasping conditions are met then
11:      constrain grasped objects
12:    end if
13:  end if
14:  if it is a object-object collision then
15:    generate feedback to the user
16:  end if
17: end for
18: end if

```

Algorithm 1: Algorithm for managing collisions between virtual objects.

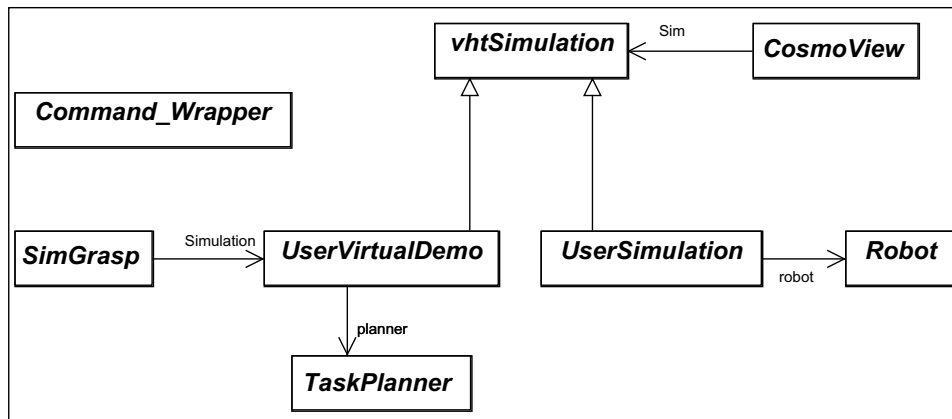


Figure 2.8: Partial class diagram of the PbD system.

the robot is computed according to the task and to the robot kinematic constraints. The simulation phase of the task does not require the activation of the collision detection factory. The **UserSimulation** is associated with the **Robot** class which is the main abstract base class that has been defined to build the internal representation of the simulated robot. The **Robot** class has different subclasses that define a hierarchy of robotic components such as mobile robots, robot manipulators and robot hands. The complete description of the developed robot library is provided in section 2.3.1. More details about the behavior of the task simulation will be discussed in chapter 3. The **SimGrasp** and **CosmoView** classes are associated to the two simulation classes and are used to manage the visual scene graphs.

2.3.1 The robot library

A robot library has been implemented to support a combination of different mobile platforms. Different models of mobile robots have been created, along with different robot arms and robot end-effectors as shown in figure 2.9. The system allows the simulation of a Nomad200 mobile robot. Three robot manipulators have been included, a six degrees of freedom Puma 560 robot arm, a Manus arm and a Spider arm. Finally

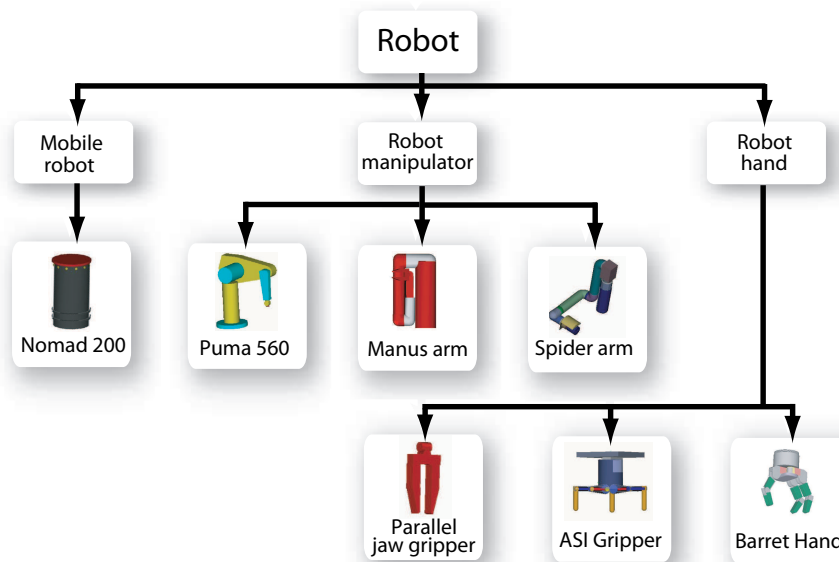


Figure 2.9: The robot library.

three robotic grippers are supported, a simple parallel yaw gripper, the Barrett Hand and the ASI gripper. The Manus arm and the Nomad200 will be described in detail in chapter 3, while the Barrett Hand will be introduced in chapter 5. The Spider arm has been designed to operate for space missions for the EUROPA project ("External Use of RObotics for Payloads Automation"). The robot is accommodated on a Pallet Adapter to carry out servicing operations of scientific payloads. It has seven degrees of freedom with a length of $1.8m$ and a mass of $60Kg$.

The ASI gripper has three degrees of freedom, and is particularly suited for no-gravity manipulation tasks. Three articulated fingers are present and contacts with the grasped object can occur along three intersecting lines equally spaced of 120° . Each articulated finger has a distal phalanx, that gets in touch with the object, and two intermediate phalanxes. As the fingers move independently, the grasping configuration may be any triangle with vertices on the approach trajectory segments. This feature

allows to grasp irregular objects, even if not positioned in a central configuration with respect to the workspace of the gripper itself. In the actual gripper each finger is also equipped with a position sensor, a proximity sensor and a miniaturized force/torque sensor.

Kinematically and geometrically correct models of the robots have been developed exploiting VRML and the RRG Kinematics library [43]. The developed robot library allows an easy reconfiguration of the simulated platform by means of an object-oriented software architecture which provides an abstraction hierarchy of the different components. The base class `Robot` shown in figure 2.8 and 2.9 provides a general interface for the subclasses defining the models and the behavior of the different robotic components. A complete mobile robotic platform can be defined by three instances of the defined classes, *i.e.* a robot gripper can be linked to a robot arm, and a robot arm can be linked to a mobile robot. The three components can be controlled independently yielding to a hierarchical representation of the robot. The motion of a component is automatically conveyed to its children, *e.g.* the translation of the mobile base is conveyed to the robot arm and to the gripper.

2.3.2 Communication infrastructure

The communication architecture between the client and the server has been built on top of a software framework for distributed telerobotic systems exploiting advanced CORBA features [44]. The framework allows development of portable multithreaded client-server applications. This section provides a brief introduction to CORBA and to the cited framework along with some details regarding the exploitation of the framework for the PbD system.

Distribution imposes several challenges on application development. User implemented application components tend to be heterogeneous, especially in the robotics domain. Being implemented in different programming languages, application components are targeted for different hardware and operating system platforms. Moreover, if application components are distributed over a network, concurrency control issues must be solved and potential failures, such as temporarily unreachable components,

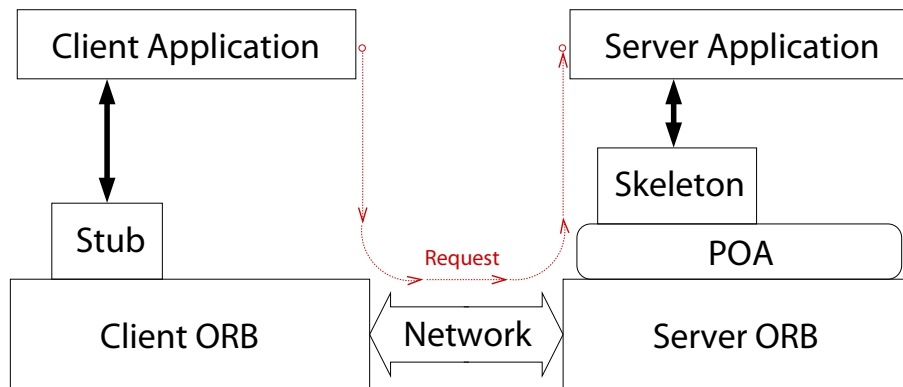


Figure 2.10: CORBA invocation of a remote method.

must be addressed.

CORBA (Common Object Request Broker Architecture) [45] is a standard middleware for heterogeneous and distributed application integration. CORBA was developed by the Object Management Group (OMG) [46], a large non-profit consortium that includes major software vendors (Sun, DEC, IBM, Apple, Hewlett-Packard, etc.) as well as end users. CORBA allows the interconnection of objects and applications, regardless of the computer language and regardless of the geographical location of the computers involved in the communication. CORBA enables integration of distributed and heterogeneous applications and reusability and portability of application components, on the basis of object oriented technology. The core component of CORBA is an Object Request Broker (ORB). It enables client objects to request operation executions from server objects. Client and server objects need not be implemented in the same programming language; they can be running on different types of operating systems and on different types of hardware platforms.

To achieve language and platform independence, an interface to the objects is written in a specification language called CORBA IDL. CORBA IDL is a module interconnection language with constructs for all concepts of the common object model. Application builders use this IDL to define export interfaces of objects. Operation execution requests can be defined statically or dynamically. For the purpose of the

```
module CTS { ...
  interface Manipulator {
    boolean set_manipulator_speed(in SpeedType speed);
    Position get_gripper_position();
    boolean move_gripper(in Position pos);
    JConfigType get_joints_configuration();
    boolean move_joint_to(in short joint, in float angle);
    boolean move_joint_by(in short joint, in float angle);
    boolean park_robot();
    boolean get_gripper_openness();
    boolean open_gripper();
    boolean close_gripper();
    boolean halt();
    boolean restore();
  };
  ... }
```

Listing 2.1: IDL interface for the robot manipulator.

PbD system static invocation has been exploited. Client objects request operation execution through stubs as local procedure calls. Client stubs for static requests, which are programming language dependent, are generated by an IDL compiler. Figure 2.10 shows a detailed scheme of a CORBA static invocation procedure.

The client entity performs application tasks by obtaining object references to servants and invoking operations on the servants objects. Servants can be local or remote, relative to the client. The client is unaware of how the CORBA object is implemented. IDL stubs and skeletons efficiently marshal and demarshal operation parameters respectively. Stubs provide a strongly typed, static invocation interface (SII) that marshal application data into a common packet level representation. Conversely, skeletons demarshal the packet level data back into typed data that is meaningful to the server application. The Portable Object Adapter (POA) associates servants with the ORB and demultiplexes incoming requests to servants. The ORB Core delivers client requests to the Portable Object Adapter and returns responses (if any) to clients.

The framework developed in [44] allows to remotely control a robot manipulator

using an IDL interface, shown in listing 2.1, and high level C++ procedure calls. It has been chosen because it can be easily interfaced with the C++ architecture governing the PbD system. Moreover, the framework has been adopted because, in the developed setup, client and server run on heterogeneous operating systems (MS Windows 2000 for the client, and Solaris 8 for the server as described in chapter 3).

At the Client side a **CommandWrapper** object, shown in figure 2.8 contains all the references to CORBA objects and interacts with the server. The client can search for components (CORBA objects such as the robot manipulator) looking for a Naming Service, that locates requested objects based on their name and returns the reference to the remote object stored under that name. The CORBA interface allows the client to initialize the robot speed, to move the arm both in the cartesian and joint space, to get the configuration of the robot, and to open and to close the gripper.

Chapter 3

One shot learning of assembly tasks

This chapter focuses on the experimental evaluation of the PbD system presented in chapter 2. The recognition module targets one shot learning of assembly tasks and the imitation strategy is aimed at recognizing a sequence of hand-object actions such as pick-and-place without reproducing the exact movements of the hand. The main objective is to investigate the benefits of a virtual demonstration environment. Overcoming some difficulties of real world demonstrations, a virtual environment can improve the effectiveness of the instruction phase. Moreover, the user can also supervise and validate the learned task by means of a simulation module, thereby reducing errors in the generation process.

Some experiments involving the whole set of system components demonstrate the viability and effectiveness of the approach. The capabilities of the system have been tested for both fixed base manipulation tasks and mobile manipulation tasks.

3.1 Assembly tasks with fixed base manipulation

This section describes the proposed PbD system for assembly tasks with a fixed base manipulator. The architecture of the system follows the canonical structure of the “teaching by showing” method, which consists of three phases. The first phase is task presentation, where the user wearing the dataglove executes the intended task in a virtual environment. In the second phase the system analyzes the task and extracts a sequence of high-level operations, taken from a set of rules defined in advance. In the final stage the synthesized task is mapped into basic operations and executed, first in a 3D simulated environment and then by the robotic platform.

The PbD system goes through four different reference frames that must be correctly matched with appropriate homogeneous transforms. The first one is the reference frame relative to the Polhemus tracker. This frame must be mapped into the second reference frame that describes the virtual demonstration environment. The third and fourth frames are attached to the simulation and the real workspaces respectively.

The modular implementation of the architecture allows easy replacement of individual modules, thus improving reusability and flexibility of the application.

3.1.1 Demonstration interface

As described in chapter 2, the demonstration interface includes an 18-sensor Cyber-Touch and a six degree of freedom Polhemus tracker. Operator’s gestures are directly mapped to an anthropomorphic 3D model of the hand in the simulated environment. The developed demonstration setup, the virtual workspace is built upon the *Virtual Hand Toolkit (VHT)* provided by Immersion Corp. As discussed in chapter 2 to grant a dynamic interaction between the virtual hand and the objects in the scene, VHT allows objects to be grasped. A collision detection algorithm (V-Clip) generates collision information between the hand and the objects, including the surface normal at the collision point. A grasp state is achieved if the contact normals provide sufficient friction; otherwise, if the grasp condition for a grasped object is no longer satisfied, the object is released. At least two fingers, one of which has to be the thumb, must be

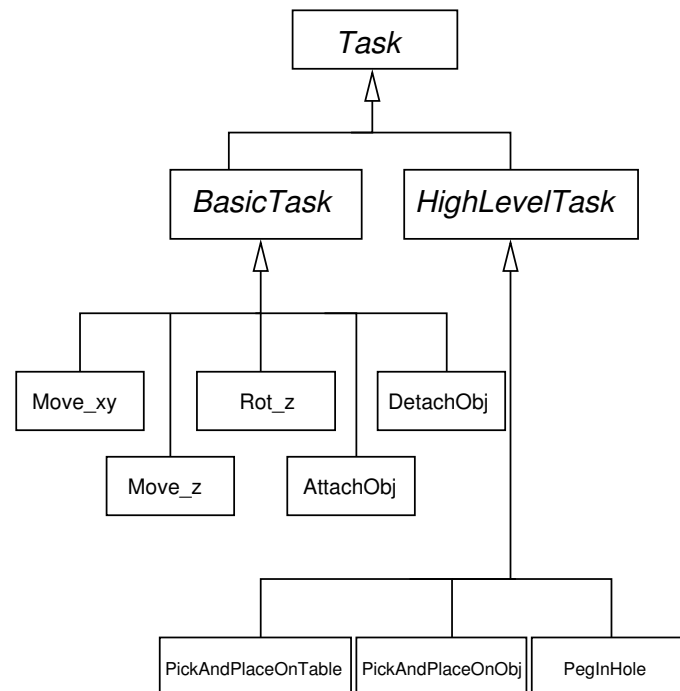


Figure 3.1: Task hierarchy.

in contact with the object to be grasped. The user interface also provides a vibratory feedback using CyberTouch actuators. Vibrations convey proximity information that helps the operator to grasp the virtual objects.

3.1.2 Task recognition

The task planner introduced in chapter 2 analyzes the demonstration provided by the human operator and segments it into a sequence of high-level primitives that should describe the user actions. To segment the human action in high-level operations, an algorithm based on changes in the grasping state has been implemented: a new operation is generated whenever a grasped object is released. The effect of the operation is determined by evaluating the achieved object configuration in the workspace.

Three high-level tasks have been identified as basic blocks to describe assembly operations. The first one is used to pick objects and place them onto a support plane (*PickAndPlaceOnTable*), the second one is used to pile objects (*PickAndPlaceOnObj*), and the last one is used to put small objects in a hole of a container on the working plane (*PegInHole*).

The three high-level tasks have been implemented in C++ as subclasses of a *HighLevelTask* abstract class (Figure 3.1). Information about the recognized high-level task is passed to the constructor when the *HighLevelTask* class is instantiated. In detail, *PickAndPlaceOnTable* constructor requires a reference to the grasped object and the release position on the table; *PickAndPlaceOnObj* constructor requires a reference to both the grasped object and the object on which it must be deployed; finally, *PegInHole* constructor requires a reference to both the grasped object and the container.

3.1.3 Task generation

A set of *BasicTasks* have been implemented for the basic movement of the real robot. The available concrete classes (Figure 3.1) include basic straight movements of the end effector, such as translations in free space or translations constrained to the XY plane, parallel to the workspace table, or towards the z axis. One additional class provides the implementation for the rotation of the end effector around the z axis. Moreover, two classes describe the basic operations to pick up and to release objects by simply closing and opening the on-off gripper of the manipulator.

The high level tasks identified in the previous phase are then decomposed in a sequence of *BasicTasks* objects describing their behavior. Table 3.1 describes how the three proposed high level tasks are decomposed as three sequences of ten basic tasks. The difference is that in the first case the object has to be released on the table, whereas, in the second one, it must be released on the top of a pile at z_f height above the table plane. Since the available manipulator has no force sensor, z_f is computed in the virtual demonstration environment based on contact relations. For the peg-in-hole task the grasped object must be released after its initial insertion in the hole.

In table 3.1 *TOP*, *TABLE* and *DOWN* are predefined constants that define the current environment.

Table 3.1: High level tasks decomposition.

	P&POnTab	P&POnObj	PegInHole
1	Move_xy(x_i, y_i)	Move_xy(x_i, y_i)	Move_xy(x_i, y_i)
2	Move_z(z_i)	Move_z(z_i)	Move_z(z_i)
3	Rot_z(θ_i)	Rot_z(θ_i)	Rot_z(θ_i)
4	AttachObj	AttachObj	AttachObj
5	Move_z(TOP)	Move_z(TOP)	Move_z(TOP)
6	Move_xy(x_f, y_f)	Move_xy(x_f, y_f)	Move_xy(x_f, y_f)
7	Move_z(TABLE)	Move_z(z_f)	Move_z(DOWN)
8	DetachObj	DetachObj	DetachObj
9	Move_z(TOP)	Move_z(TOP)	Move_z(TOP)
10	Rot_z(θ_f)	Rot_z(θ_f)	Rot_z(θ_f)

Each concrete class of the task tree provides two methods to perform the operation, one in the simulated environment and one in the real workspace. Once the entire task has been planned, the task performer manages the execution process in both the simulated and real workspaces.

3.1.4 Task simulation

After the recognition phase, the system displays to the human operator a graphical simulation of the generated task. This simulation improves safety since the user can check the correctness of the interpreted task. If the user is not satisfied after the simulation, the task can be discarded without execution in the real environment.

The simulation is non-interactive and takes place in a virtual environment exploiting the same scene graph used for workspace representation in the demonstration phase. The only difference is that the virtual hand node in the HSG is replaced by a VRML model of the robot manipulator currently simulated. In the following examples the Puma 560 robot arm has been simulated. The VRML model of the Puma

560 comprises about 200 vertices and 300 triangles.

The simulated robot has the ability to perform all the operations described in the previous section. The movement of the VRML model is obtained applying an inverse kinematics algorithm for the specific robot and is updated at every frame. To this purpose, the PbD system exploits the RRG Kinematix library from the Robotics Research Group at the University of Texas ([43]). The library can be used for the computation of forward and inverse kinematics (using the pseudo inverse method) of any serial chain manipulator. All information about the robot is included in a Denavit-Hartenberg parameter file supplied to the algorithm.

In the simulation, picking and releasing operations are achieved by attaching and detaching the nodes of the HSG representing the objects to the last link of the VRML model of the manipulator.

The initial location of the objects in the simulation environment can be different from their position in the demonstration environment as the systems recognizes and generates the high level assembly operations by assigning labels to the objects being manipulated. The exact location of the objects is resolved at simulation time and the initial configuration of the objects in the real workspace must match the configuration of the workspace in the simulated environment. Further work will focus on the development of a robust vision system for the recognition of the objects in the real workspace.

3.1.5 Task execution

The execution in the real workspace is obtained through the C++ framework [44] based on the TAO ACE ORB [47] introduced in chapter 2. In particular, The TAO ORB is implemented on top of ACE. This is the lower level middleware that implements the core concurrency and distribution patterns for communication software. ACE provides the reusable C++ wrapper framework components that support the quality of service requirements of high-performance, real-time applications.

The PbD system builds a client-server CORBA connection using a Fast Ethernet switch. The client side runs on MS Windows 2000, whereas the server controlling the

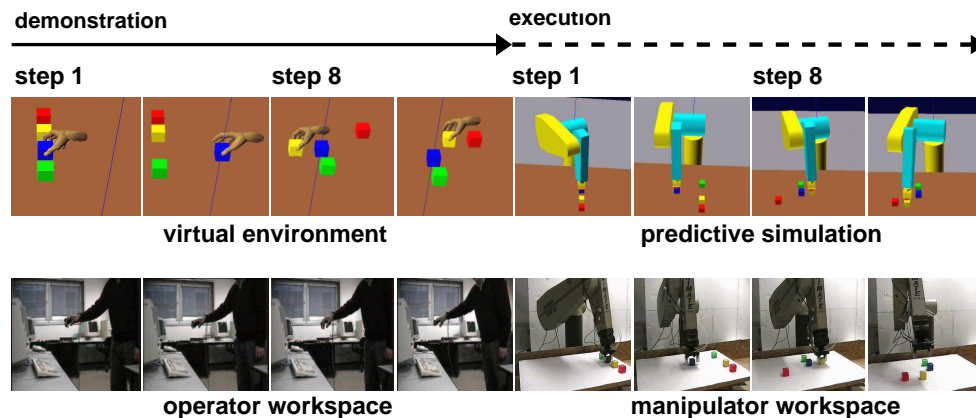


Figure 3.2: PbD of an assembly task in a block world.

real manipulator runs on Solaris 8. The methods of the concrete classes in the task list invoke blocking remote calls of the servant manipulator object which transforms them into manipulator commands, exploiting the Robot Control C Library (RCCL) [48].

3.1.6 Experiments

The implementation of the virtual environment for assembly tasks in the “block world” consists of a single plane and a set of 3D colored blocks on it reproducing qualitatively the real workspace configuration.

The capabilities of the PbD system have been evaluated in assembly experiments comprising pick and place operations on the workspace plane, stacking of objects and peg-in-hole operations. Three of experiments are described in the following.

The first experiment (Figure 3.2) consists of a sequence of eight pick and place actions on the working plane. The workspace contains four colored boxes of the same dimensions. The images on the left of figure 3.2 are snapshots of the virtual environment and the operator workspace for the demonstration phase; the images on the right show the simulation and the real workspaces.

In the second experiment (Figure 3.3) the workspace contains three objects: two

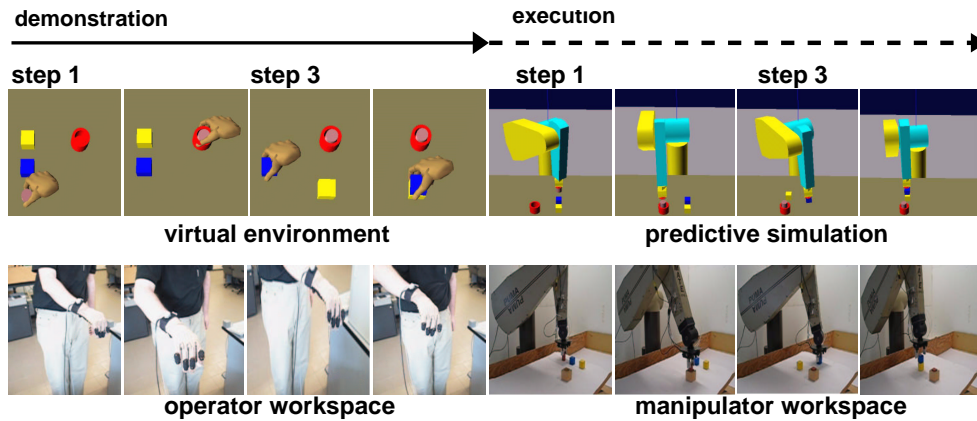


Figure 3.3: PbD of Peg-in-hole and piling of boxes tasks.

colored boxes and a cylinder. The user demonstration consists of a sequence of three steps. The user first picks up the cylinder and puts it in the container, then puts the yellow box on a different position on the table, finally he grasps the blue box and stacks it on top of the yellow one. While performing the demonstration, the user can dynamically adjust the point of view of the virtual scene. This feature, typical of demonstration in virtual environments, can help in picking partially occluded objects, releasing them on the plane or on other boxes, and inserting them in the container.

Figure 3.4 shows a close-up image of the peg entering the hole. It should be mentioned that since the robot has no force sensor, the task is essentially a pick-and-place tasks with tight accuracy requirements and no contact is intended between the peg and the hole.

In the third experiment the user demonstrates a stacking tasks of three boxes and a peg in hole insertion. The initial configuration of the objects in the demonstration and simulation environments are different. Initially the blue box is placed on the working plane with a certain orientation with respect to the z axis, then the remaining boxes are stacked onto it. As a stacking operation is recognized, for example when the yellow box is released nearly on the top of the blue box, the system automatically displays the yellow box on the top of the blue box with the proper orientation and

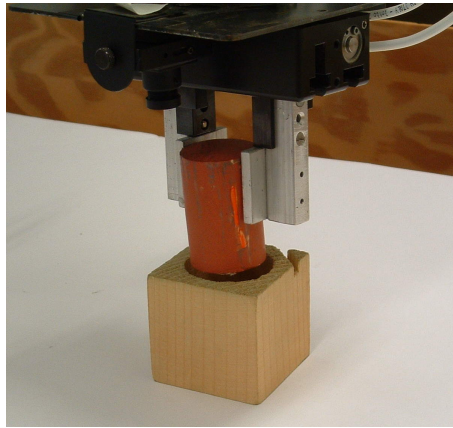


Figure 3.4: Peg entering the hole.

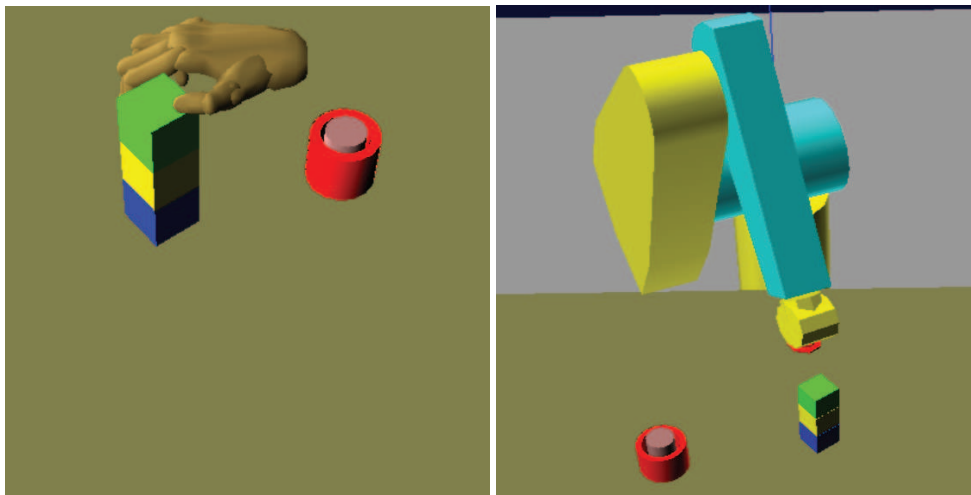


Figure 3.5: Demonstration environment and task simulation for a stacking task.

vertical alignment. This is a kind of visual virtual fixtures which is introduced in the virtual environment to assist the user in the assembly operation. Figure 3.5 shows the demonstration environment and the corresponding simulation.

3.2 Assembly tasks with mobile manipulation

Mobile platforms are commonly used today for a wide range of applications such as automated warehouses and cleaning. In most applications the interaction with the environment has been engineered to make the task well-defined and easier. However, little progress has been made toward practical manipulation systems capable of functioning autonomously. For complex tasks that are more general, especially in unstructured environments, robotic platforms require human interaction. Besides simplifying complex manipulation tasks, human robot interaction for mobile manipulation is fundamental for assisting handicapped and elderly people in domestic environments.

The use of virtual environments for mobile robot programming has been mainly investigated for teleoperation tasks by [49, 50, 51]. Virtual reality has also been used to simulate social robotic behavior of mobile autonomous robots by [52]. Task learning and interactive teaching have been proposed for mobile assistants by [53, 54]. A particular attention is often dedicated, in the cited papers, to novel interfaces for human robot interaction such as multimodal programming.

The prototype system described in the following is an advanced user interface for programming manipulation tasks by demonstration. The mobile platform consists of a mobile base carrying a robot manipulator on its top. As described in chapter 2 the simulation environment of the system supports the Nomad200 mobile robot, by Nomadics Technologies Inc., and several serial robot manipulators.

The Nomad 200 is a three-wheel synchronous drive, non-holonomic base. The three wheels are controlled by two motors, one driving their translation and the other controlling their steering. A third motor controls the rotation of the turret on top of the base housing sensors and onboard computers. Twenty bumpers (tactile sensors) are arranged in two offset rings around the base. Around the turret, two rings of sonar and

active infrared sensors (16 sensors each) provide distance and proximity information. With an 80 cm height and a 50 cm diameter, the Nomad 200 can accept a payload of about 200 kg, which allows mounting a manipulator on top of it.

The Manus robot arm was used in the proposed experiment so as to simulate the real mobile manipulation platform available at the Robotics and Intelligent Machines Laboratory (RimLab) of the University of Parma. The Manus arm is a lightweight 6 d.o.f. manipulator (about 15 kg, with all the motors located in the base link), by ExactDynamics [55], designed to be mounted as a wheelchair add-on for disabled persons. The maximum payload is 2 kg. The arm has been designed to operate in human environments and with disabled persons. Therefore it incorporates safety features to prevent serious damage to the humans such as a limitation of the maximum motor power, speed of arm motion, and gripping force. Kinematically and geometrically correct models of the arm and the base have been developed exploiting VRML and the RRG Kinematics library as described in section 3.1.4. The VRML model of the Manus arm has been linked to the VRML node representing the mobile base. Therefore in the complete model the mobile base acts as the father of the robot arm and the translational and rotational motion of the mobile robot are conveyed to the arm. The VRML model of the Nomad 200 has about 200 triangles while the Manus arm comprises about 1300 triangles.

As the real mobile platform is intended to move with the robot arm positioned in a very compact folded posture, which minimizes the chances of unwanted collisions with the surrounding environment, the simulation environment includes the automatic procedures for the fold-out and for the fold-in of the robot arm.

Currently, the functionalities of the systems have been tested in simulation. For the actual application of the system to a concrete scenario more investigation is required to include in the system autonomous capabilities for obstacle detection and accurate object recognition.

The demonstration interface consists of two components. The first is a two dimensional graphical interface showing a representation of the workspace from the top. The user exploits this interface to select via-points he wants the mobile robot to

visit in its navigation. The via-points are introduced with the mouse. At the end of the selection process the system generates a curve that interpolates the specified via-points. The generated curves are Non-Uniform Rational B-Splines (NURBS) and the interpolation algorithm exploits the NURBS++ library [56] that will be introduced in chapter 4 and in appendix A. The user is asked to select also the region of interest for the manipulation task by clicking near the object to be manipulated. This information is used to compute a suitable final position and orientation of the mobile base relative to the objects.

- 1: Trajectory specification to approach the object
 - 2: Task demonstration in virtual environment
 - 3: Trajectory specification for release
- Task simulation:
- 4: Mobile base navigation towards the object
 - 5: Fold-out of the robot arm
 - 6: Robot arm motion towards the object
 - 7: Object grasping
 - 8: Fold-in of the robot arm
 - 9: Mobile base navigation towards release region
 - 10: Fold-out of the robot arm
 - 11: Robot arm motion towards the goal
 - 12: Release of the grasped object.

Algorithm 2: Demonstration and simulation sequence of a mobile manipulation task.

The second component of the demonstration interface is a virtual environment used for the demonstration of the manipulation task. The behavior of the virtual environment is identical to the one presented in section 3.1.1. The system after the generation of the interpolated NURBS evaluates the curve and transforms the coordinates of the evaluated points into the reference frame of the simulation environment. The evaluation process extracts a set of equidistant point from the curve which are then linearly interpolated at each step of the simulation. The mobile base moves and rotates at constant speed. The complete sequence of steps required for the demonstration (and the corresponding simulation) of a manipulation task in the current prototype are resumed in Algorithm 2.

In the proposed experiment the mobile manipulator performs a manipulation tasks in an indoor environment representing the ground floor of the Department of Computer Engineering of the University of Parma. It starts in a room and is programmed to perform a manipulation task in a different room. Figure 3.6 shows the interface used for the specification of the trajectory of the mobile robot. In particular, the image on the left shows the generated trajectory for the approach phase of the mobile robot to the table containing the objects to be manipulated, while the figure on the right shows the trajectory followed by the mobile base to reach the zone of the environment for the release of the grasped object.

Figure 3.7 show a sequence of images taken from the demonstration phase of the manipulation tasks. The user grasps with the virtual hand a blue cube and releases it on a different table. Figure 3.8 shows a sequence of images taken from the complete simulation of the task.

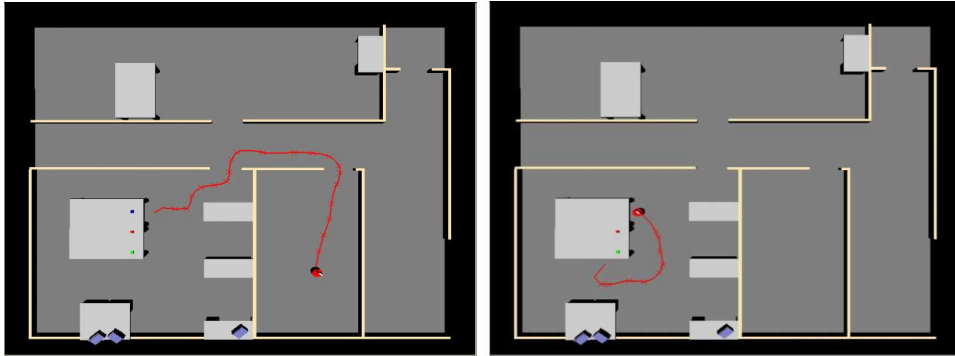


Figure 3.6: Trajectory specification for a mobile manipulation task.

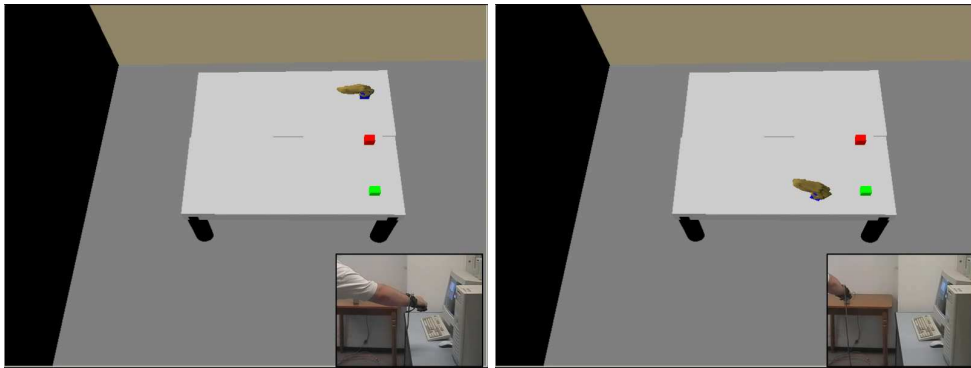


Figure 3.7: Demonstration environment in virtual reality for a mobile manipulation task.

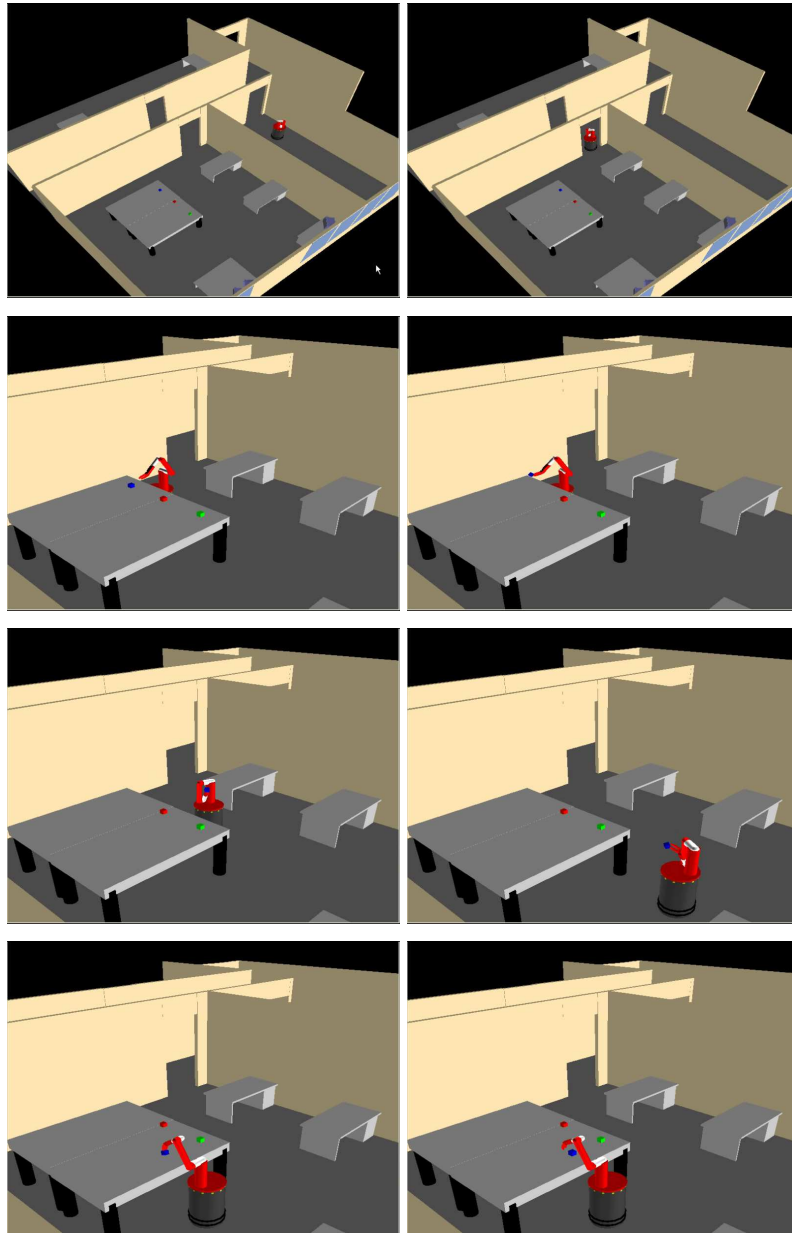


Figure 3.8: Simulation environment for a mobile manipulation task.

Chapter 4

Trajectory Reconstruction and Stochastic Approximation

This chapter describes the trajectory learning component of the PbD system. Trajectory learning is a fundamental aspect in robot Programming by Demonstration, where often the very purpose of the demonstration is to teach complex manipulation patterns. However, human demonstrations are inevitably noisy and inconsistent. The proposed approach is suitable for learning from both individual and multiple user demonstrations. In case of multiple demonstrations, the approach clusters recorded object trajectories into qualitatively different sets, and recovers a smooth robot trajectory overcoming sensor noise and human motion inconsistency problems. Trajectory selection exploits a Hidden Markov Model–based stochastic procedure for trajectory evaluation. Eventually, the desired robot end-effector trajectory corresponding to the human demonstrations is reconstructed as a NURBS curve by means of a best-fit data smoothing algorithm. The resulting curve is also suitable for interactive, local modification in a simulation environment which is an integral part of the PbD system.

The rest of the chapter is organized as follows. Section 4.1 introduces the trajectory learning problem. Section 4.2 reviews related research in PbD involving motion learning with HMMs and trajectory generation. Section 4.3 describes the structure of

the system and the algorithms for trajectory clustering, selection and approximation. Section 4.4 shows some experimental results that demonstrate the potential of the proposed approach. In particular, some experiments involving object transportation while avoiding obstacles in the workspace are presented.

4.1 Introduction to trajectory learning

In chapter 3 the capabilities of the PbD platform for one shot learning of basic assembly tasks have been presented. The imitation strategy was aimed at recognizing a sequence of hand-object actions such as pick-and-place, stacking, and peg-in-hole, without reproducing the exact movements of the hand. The task recognized was then performed in a simulated environment for validation and, possibly, in a real workspace exploiting basic straight line movements of the end effector. This approach, however, is not feasible in presence of obstacles in the workspace. In this chapter a multi-step trajectory learning technique is presented, the system generates paths by proper imitation and filtering of user trajectories. The method allows the system to fit datasets containing one or multiple example trajectories. As stated in [11], data smoothing yields several advantages for characterizing human motion, since it removes noise from tracking sensors and it reduces jitter and unwanted movements.

To deal with the case of multiple user demonstrations, a distance-based geometric procedure is proposed which clusters hand trajectories into sets representing alternative feasible solutions for the same manipulation task. Trajectory clustering is required, since qualitatively different paths of the hand can emerge in multiple demonstrations of the same task. Without clustering, the approximation of the whole dataset with a single trajectory would lead to the possibility of collisions of the trajectory with the obstacles in the workspace and, in general, to a loss of information.

In addition to the clustering procedure, a method to fit clusters containing one or multiple trajectories is presented. An automatic selection procedure identifies the most consistent trajectories within each cluster. An algorithm is then applied to fit them with a single path. Trajectory selection is needed to filter out inconsistent mo-

tions, possibly arising due to jitter or involuntary movements. The proposed trajectory synthesis technique is based on a combination of Hidden Markov Models (HMMs) for trajectory selection and Non-Uniform Rational B-Splines (NURBS) for trajectory approximation.

While the focus of this chapter is on the exploitation of multiple user demonstrations, the proposed approach can also deal with a single user demonstration. In case of a single demonstration, the approach filters out high-frequency trajectory components, typically due to noise, arm tremor, and involuntary movements, and generates a trajectory fitting the provided demonstration with a controlled degree of approximation. The synthesized NURBS-based trajectory has a high-level, compact representation, satisfies mathematical continuity constraints, and can be manipulated with local changes in an interactive graphical environment. When multiple demonstrations are provided, in addition to supporting the above features, the approach takes advantage from the implicit information conveyed by demonstrations about the free space available to the intended motion. Moreover, it identifies the qualitatively different trajectories adopted by the user.

4.2 Related work

Trajectory learning has been the focus of extensive research, especially in the last few years.

Ude and Dillmann [57] proposed a PbD system where a stereo vision system is used to measure the trajectory of the objects being manipulated by the user. Smoothing vector splines are utilized to reconstruct single trajectory either in Cartesian or in joint coordinates.

Billard and Schaal [58] presented a biologically inspired method for human imitation. The model was implemented on a dynamic simulation of a humanoid robot. The algorithm is based on a hierarchy of artificial neural networks.

Riley et al. [59] focused on motion synthesis of dance movements for a humanoid robot. The approach includes collection of example human movements, handling of

marker occlusions, extraction of motion parameters, and trajectory generation. In [60] the same authors presented a method for the formulation of joint trajectories based on B-spline basis functions.

In [61] a method to enable real-time full body imitation of a humanoid robot is presented. The robot uses a 3D vision system to perceive the movements of the human teacher, and then estimates the teacher's body posture through an inverse kinematics algorithm.

Lee [62] described a spline smoother for finding a best-fit trajectory from multiple examples of a given physical motion that can simultaneously fit position and velocity information. The system was tested for handwriting motions.

The early work of Delson and West [11] provided the conceptual basis for the exploitation of multiple human demonstrations in PbD systems. They investigated a method for generating a robot trajectory from multiple demonstrations for both 2D and 3D tasks. The range of human inconsistency is used to define an obstacle-free region. The generated robot trajectory is guaranteed to avoid obstacles and is shorter than any of the demonstrations.

The works [63, 64] are among the early ones exploiting HMMs for skill learning and acquisition. In [63], Yang et al. described a system for skill learning with application to telerobotics where human skills are represented as parametric models using HMMs. The best action sequence can be selected from all previously measured actions using the most likely performance criterion. Experiments were carried out involving Cartesian space, joint space and velocity domain. In [64], the authors presented a similar method for human intent recognition and skill learning of gestures, like digits drawn with a mouse, in both isolated and continuous cases. An important distinction from the work presented in this thesis is that the approaches presented in [63, 64] are based on a static classification of human gestures that does not support clustering.

Tso and Liu [65] focused on the evaluation of behavior consistency and trajectory selection with a stochastic approach based on HMMs. In their work, the most consistent trajectory among multiple human demonstrations is selected as the best one. A

similar method has been proposed by Yu et al. [66] for finding the best demonstrated curve in a robotic therapy experiment for persons with disabilities. The trajectory evaluation and selection approach (which was also investigated in [63, 64]) is unable to filter out noise and involuntary movements in the examples. The solution proposed in this thesis is more robust because it is based on a method for trajectory approximation after selection.

The work of Pook and Ballard [67] presented results for the recognition and segmentation of manipulation primitives from a teleoperated task. K -nearest pattern vectors from sensor feedback determine potential classifications and a Hidden Markov Model provides task context for the final segmentation. The illustrative task is the picking up of a plastic egg with a tool.

Bluethmann et al. [68] proposed another system for learning new dexterous skills by humanoid robots via teleoperated demonstration. The system was experimented at the NASA Johnson Space Centre on the Robonaut humanoid.

Lieberman and Breazeal [69] discussed an approach for learning new motor skills through multiple demonstrations while preserving accuracy and style of the movement. The approach blends between two solutions for the approximation of the motion. The first solution uses radial basis functions to interpolate demonstrations in absolute world coordinate space. The second solution uses radial basis functions to interpolate end effector paths in a coordinate frame relative to the object to be grasped.

Inamura et al. [70] proposed a framework called the "mimesis model" based on Hidden Markov Models. The framework allows abstraction of motion patterns for a humanoid robot and symbol representation, generation of self-motions from the symbol representation and recognition of motions.

Calinon and Billard [71] proposed a HMM-based system for gesture recognition and reproduction. The model extracts key-states as inflexion points in joint-angles trajectories. The system was tested on a humanoid robot that recognizes and reproduces stylized letters.

Drumwright et al. [9] introduced a method for performing variations on a given demonstrated behavior with the aid of an interpolation mechanism. The method al-

allows recognition of free-space movements for humanoid robots. This approach can achieve a more accurate interpolation of the examples compared to the solution presented in this chapter, which is based on a global approximation algorithm. However, the proposed method has fewer constraints, since the approach in [9] requires an appropriate manual segmentation of the example trajectories into time-frames.

Finally, the work of Ogawara et al. [72] is the only one, that investigates the problem of automatically clustering manipulation tasks demonstrated by the user. An important distinction with respect to the approach proposed in this thesis lies in the choice of the clustering algorithm, as will be discussed in the following section.

None of the works discussed above exploits NURBS curves for trajectory approximation. The advantages brought by NURBS curve fitting will be shown later in sections 4.3.3 and 4.4.

4.3 The trajectory learning technique

The trajectory learning subsystem described in this chapter is conceived as an integral part of the PbD system presented in chapter 3. Figure 4.1 provides a schematic representation of the subsystem, highlighting the role of trajectory learning into the overall architecture.

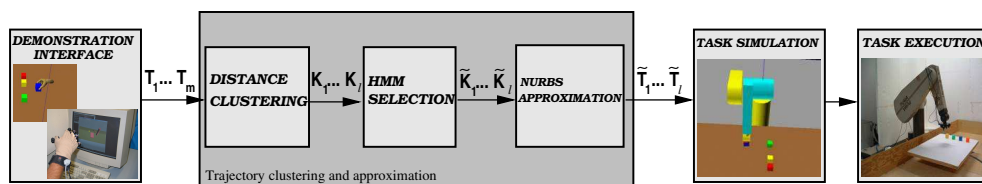


Figure 4.1: PbD system with trajectory learning.

The complete reference task consists of several demonstrations of the same operation, each one divided into three phases, namely grasping, transportation, and releasing of an object. For every operation the system tracks the (x, y, z) coordinates of the grasped object. After each demonstration, the manipulated object is brought back to

its original position. The resulting dataset consists of m trajectories $T_1 \dots T_m$. Each T_i can be written as $T_i = \{(x_{i,n}, y_{i,n}, z_{i,n})\}$, where i is the trial index and n is the index of the sample within the trial.

The sampling rate of the tracker was set to $20ms$ for the experiments reported in this chapter. To reduce memory requirements and speed-up the later processing stages, a distance filtering procedure was also applied to the data recorded by the tracker. The procedure consisted in discarding those samples whose distance to an accepted sample was below a given programmable threshold.

After the demonstration phase, the *trajectory clustering and approximation* module takes as input the initial dataset $T_1 \dots T_m$ which is then clustered into l groups that correspond to macro-movements representing qualitatively different executions of the manipulation task. Eventually, this module generates as output a transformed set of trajectories $\tilde{T}_1 \dots \tilde{T}_l$ that best fit the identified clusters.

After the clustering and approximation phase, the recognized task is performed in a simulated environment for validation. The simulation shows the 3D model of a robot arm, reproducing the current experimental setup, and the set of the generated trajectories \tilde{T} . The user can select the desired trajectory that will be followed by the end-effector. The end effector is programmed to follow the generated curve by applying an inverse kinematics algorithm.

4.3.1 Trajectory clustering

A distance-based clustering algorithm has been implemented to merge the set of trajectories T into a set of clusters $K_1 \dots K_l$. The initial dataset is first preprocessed to equalize all the trajectories to the same duration N , where N is the number of samples of the shortest trajectory. Data reduction is achieved by randomly undersampling the trajectories whose initial size is larger than N .

A hierarchical clustering algorithm is then applied to the preprocessed dataset. An m by m distance matrix D is built by assigning each trajectory to an initial cluster and letting the distances between trajectories T_i and T_j equal the normalized sum of

distances, which is given as

$$d_{i,j} = \frac{1}{N} \sum_{n=1}^N [(x_{i,n} - x_{j,n})^2 + (y_{i,n} - y_{j,n})^2 + (z_{i,n} - z_{j,n})^2]^{\frac{1}{2}} \quad (4.1)$$

The average-link clustering method has been chosen. At each iteration the closest pair of clusters $\operatorname{argmin}_{i,j} d_{i,j}$ is selected, with their associated sets denoted as K_i and K_j . The two clusters are merged into a single cluster, therefore reducing by one the size of D . The combined cluster of index k replaces the higher-numbered merged cluster (*i.e.* $k = \max(i, j)$). Distances $d_{k,s}, d_{s,k} \forall s$ are determined according to a merge average-link heuristic:

$$d_{k,s} = \frac{(|K_i| \times d_{i,s}) + (|K_j| \times d_{j,s})}{|K_i| + |K_j|}, \forall s \quad (4.2)$$

and similarly for $d_{s,k}$, where $|K_i|$ is the cardinality of the cluster K_i .

The hierarchical clustering ends when the distance between the two closest clusters is above a constant threshold that was empirically determined.

The proposed clustering algorithm is deterministic but does not cope with uneven trajectory duration; moreover, the equalization process can lead to some loss of information. Alternative clustering strategies, such as HMM-based clustering [72], do not require equalization. However, in a preliminary investigation, these stochastic methods have proven less robust and computationally more expensive.

4.3.2 HMM-based trajectory evaluation

HMMs are a general stochastic tool for modelling time series data in partially observable systems. A HMM is a graph of connected states whose transitions are hidden. The states emit observation symbols that can be measured to uncover the underlying process of state changes. As reported in section 4.2, HMMs have been successfully applied in the context of skill learning and gesture recognition, where the human actions are the measurable signals and the human mental states are the unobservable

underlying stochastic processes.

In this section an automatic approach is described, for evaluating the most consistent trajectories within each cluster. The algorithm consists of two phases. In the first phase the equalized dataset of positions is preprocessed again to convert it into a finite number of discrete observable symbols suitable for training discrete HMMs. In the second phase a HMM is built and trained for each cluster. Finally, for each group a stochastic evaluation procedure is applied to find out the most consistent trajectories.

4.3.2.1 Preprocessing

Trajectories are transformed from the time domain to the frequency domain by applying the short-time Fourier transform (*STFT*) to their three cartesian components [63, 64]. This transform efficiently preserves information from the original signal and emphasizes local frequency properties. The STFT of a signal $x(t)$ is the Fourier transform of the signal multiplied by a shifted analysis window $\gamma^*(t' - t)$ defined as

$$STFT_x^\gamma(t, f) = \int_{t'} [x(t')\gamma^*(t' - t)]e^{-j2\pi ft'} dt' \quad (4.3)$$

The Hamming window was used with a width of 20 points and with 50% overlap to prevent loss of information.

After the frequency analysis, the spectra are quantized to a finite codebook of L prototype vectors using the *k-means* algorithm. Prototypes vectors are called code-words and correspond to discrete symbols. Thus, vector quantization maps the frequency spectra of a trajectory to a sequence of symbols that are used as observable entities to train a HMM.

The *k-means* algorithm is an unsupervised learning technique for partitioning N data points into K disjoint subsets S_j containing N_j data points so as to minimize the sum-of-squares criterion:

$$J = \sum_{j=1}^K \sum_{n \in S_j} |x_n - \mu_j|^2 \quad (4.4)$$

where x_n is a vector representing the n th data point and μ_j is the geometric centroid of the data points in S_j . The *k-means* algorithm (Algorithm 3) consists of a re-estimation procedure as follows. Initially, K data points are selected at random as initial cluster centers. In step 2, every point is assigned to the cluster whose centroid is closest to that point. For step 3, the centroid is recomputed for each set. Steps 2 and 3 are repeated until a stopping criterion is met, *i.e.*, when there is no further change in the assignment of the data points.

- 1: Select K data-points at random, as initial cluster centers.
- 2: Assign all data points to their nearest cluster center.
- 3: Compute the mean within each cluster, and let these be the new cluster centers.
- 4: Repeat from 2.

Algorithm 3: K-means algorithm.

4.3.2.2 HMM trajectory selection

A HMM [73] is defined by a set of states S , including an initial state S_I and a final state S_F ; a transition probability matrix, $A = \{a_{ij}\}$, where a_{ij} is the transition probability from state i to state j ; a set of state probability density models $B = \{b_j(\mathbf{O})\}$, where $b_j(\mathbf{O})$ is the probability distribution in state j of the 3-dimensional vector of observation symbols \mathbf{O} . A HMM can be expressed compactly as $\lambda = (A, B, \pi)$, where π is the initial state distribution.

For the representation of the probability density functions $b_j(\mathbf{O})$ a general solution has been adopted that is a finite mixture of gaussians of the form

$$b_j(\mathbf{O}) = \sum_{m=1}^M c_{jm} \mathfrak{N}[\mathbf{O}, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}], \quad 1 \leq j \leq N \quad (4.5)$$

where c_{jm} is the mixture coefficient for the m th mixture in state j and \mathfrak{N} is a gaussian density function with mean vector $\boldsymbol{\mu}_{jm}$ and covariance matrix \mathbf{U}_{jm} for the m th

mixture component in state j . \mathfrak{N} has the following form:

$$\mathfrak{N}[\mathbf{O}, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}] = \frac{1}{(2\pi)^{n/2} |\mathbf{U}_{jm}|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{O} - \boldsymbol{\mu}_{jm})^t \mathbf{U}_{jm}^{-1} (\mathbf{O} - \boldsymbol{\mu}_{jm}) \right] \quad (4.6)$$

where $|\mathbf{U}_{jm}|$ is the determinant of the covariance matrix \mathbf{U}_{jm} .

A 3-dimensional HMM is created for each cluster to deal with the multidimensional dataset. The Baum-Welch algorithm is used for training, and the topology of the HMMs is fixed to a Q state left-right model (Bakis model). An example of a five state Bakis HMM is shown in figure 4.2.

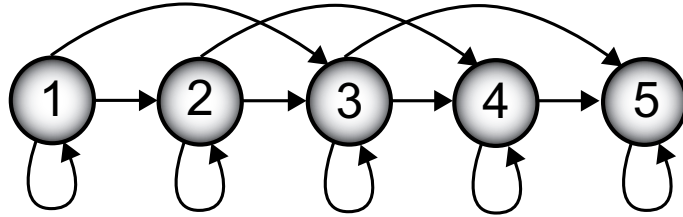


Figure 4.2: Example of a left-right HMM.

After the training phase an evaluation procedure is started within each cluster. The initial set of clusters $K_1 \dots K_l$ is transformed into a restricted set $\tilde{K}_1 \dots \tilde{K}_l$ that has the same dimension but such that each cluster contains only the n most consistent trajectories. The selection process scores each trajectory using as evaluation metric the probability $p(\mathbf{O} \mid \lambda_i)$ of output of the observation sequence \mathbf{O} given the model λ_i for the i th cluster. This probability can be computed with the Forward-Backward algorithm. The selection procedure within a cluster is skipped if the number of trajectories is lower than n . An overview of discrete and continuous Hidden Markov Models is provided in appendix B.

4.3.3 Trajectory approximation

NURBS curves are the mathematical tool exploited in the PbD system for trajectory approximation. NURBS curves have become popular in the CAD/CAM community

as standard primitives for high-end 3D modelling applications and simulation environments since they provide a flexible way to represent both standard analytic and free-form curves; moreover, their evaluation is fast and they can be easily manipulated. NURBS have proven the best parametric curves also for path planning and 3D curve approximation [74].

After the evaluation procedure, given the set of clusters $\widetilde{K}_1 \dots \widetilde{K}_l$, the system computes an approximating trajectory within each cluster. For each cluster K_i the data points, that belong to the selected trajectories, are first transformed into the reference frame of the simulation environment and then sorted into a single list of points $Q_s = \{(x_s, y_s, z_s)\}$ according to their euclidean distance to the starting point Q_0 of the movement ($\|Q_1 - Q_0\| \leq \|Q_2 - Q_0\| \leq \dots \leq \|Q_f - Q_0\|$). The starting point is known, given the configuration of the objects in the environment. Finally, an approximation algorithm is applied to the sorted data set to find the best fitting NURBS curve. It should be mentioned that this approach is not suitable for all types of trajectories.

A NURBS [74] is a vector-valued piecewise rational polynomial function of the form

$$C(u) = \frac{\sum_{i=0}^n w_i P_i N_{i,p}(u)}{\sum_{i=0}^n w_i N_{i,p}(u)} \quad a \leq u \leq b \quad (4.7)$$

where the w_i are scalars called weights, the P_i are the control points, and the $N_{i,p}(u)$ are the p th degree B-spline basis functions defined recursively as

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u \leq u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \quad (4.8)$$

where u_i are real numbers called knots that act as breakpoints, forming a knot vector $U = u_0 \dots, u_t$ with $u_i \leq u_{i+1}$ for $i = 0, \dots, t$.

Two approximation algorithms have been investigated [75]: the first one is based on a least squares method, and the second one is based on a global approximation algorithm with a specified error bound. The first algorithm finds the NURBS curve C satisfying $Q_0 = C(0)$ and $Q_f = C(1)$; the remaining points Q_s are approximated in the least squares sense, minimizing the following expression with respect to $C(u_s)$:

$$\sum_{s=1}^{f-1} |Q_s - C(u_s)|^2 \quad (4.9)$$

The global approximation algorithm exploits a knot removal procedure to change the geometry of the NURBS. The algorithm eliminates as many knots as it can, as long as the curve stays within a certain error bound.

The PbD system exploits the NURBS++ library [56] and the HMM Matlab Toolbox [76]. This toolbox supports inference and learning for HMMs with discrete outputs, Gaussian outputs, or mixtures of Gaussians output. The Gaussians can be full, diagonal, or spherical (isotropic). The covariance matrices used in the experiments have a diagonal structure. Both the trajectory clustering algorithm and the trajectory selection algorithm have been developed in Matlab. The Matlab compiler has been exploited to automatically convert the algorithms into a self-contained application which has been linked to the main PbD system. Details about the NURBS++ library are provided in appendix A. The proposed method for stochastic trajectory learning and approximation is summarized in algorithm 4.

- 1: Demonstration phase: the user provides m trajectories $T_1 \dots T_m$.
- 2: Equalization of all the trajectories.
- 3: STFT of the trajectories.
- 4: K-means quantization of spectra to a finite codebook of L prototype vectors.
- 5: A 3-dimensional HMM λ_i is created for each cluster.
- 6: HMM training within each cluster.
- 7: Trajectory scoring ($p(\mathbf{O} | \lambda_i)$).
- 8: Clusters $\tilde{K}_1 \dots \tilde{K}_l$ are created containing only the n best scoring trajectories.
- 9: NURBS trajectory approximation within each cluster.
- 10: A transformed set of trajectories $\tilde{T}_1 \dots \tilde{T}_l$ best fitting the identified clusters is generated.

Algorithm 4: Stochastic trajectory learning and approximation algorithm.

4.4 Experiments

The trajectory learning capabilities of the PbD system have been evaluated in manipulation experiments consisting of pick-and-place operations on a small box (shown in yellow or clear shading in figure 4.3) in a workspace comprising a working plane and two static obstacles (a box and a cylinder).

Five experiments are described in the following. The first two experiments tested the viability of the two NURBS approximation algorithms, whereas the remaining experiments tested the trajectory clustering, selection and editing features.

Experiment 1 (Figure 4.3) consisted of a single demonstration. The user was re-

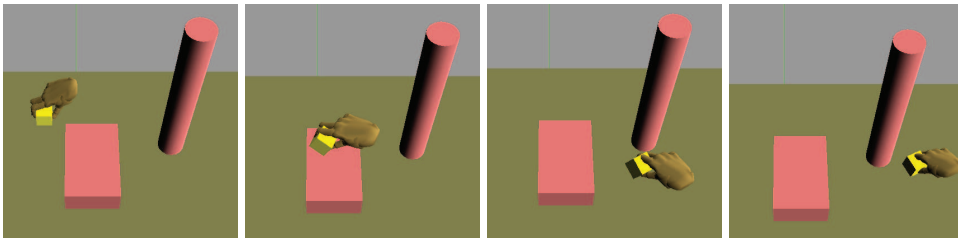


Figure 4.3: Task demonstration for Experiment 1.

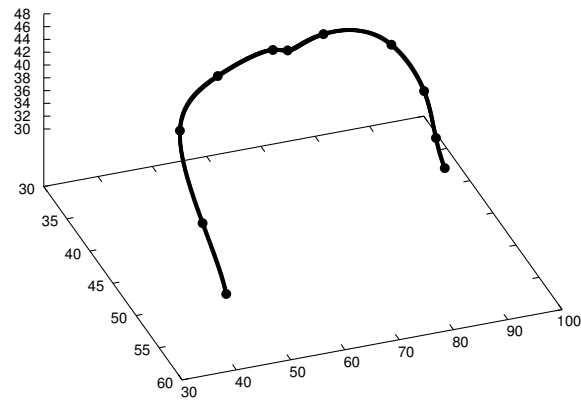


Figure 4.4: Generated NURBS (least squares algorithm) with sampled points for Experiment 1.

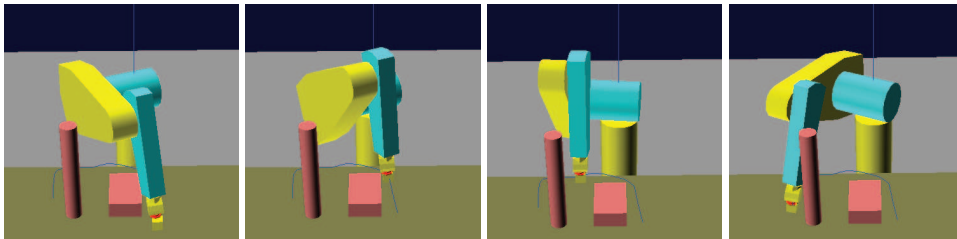


Figure 4.5: Task simulation for Experiment 1.

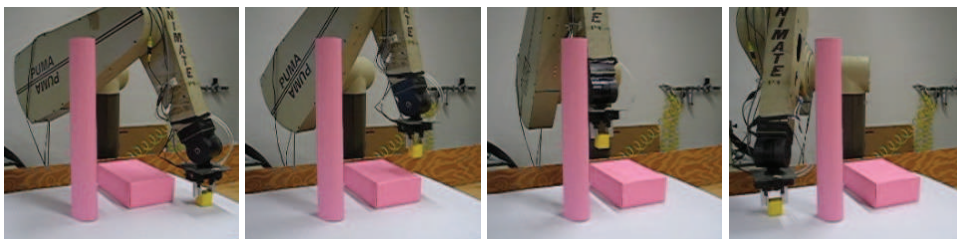


Figure 4.6: Task execution in a real workspace for Experiment 1.

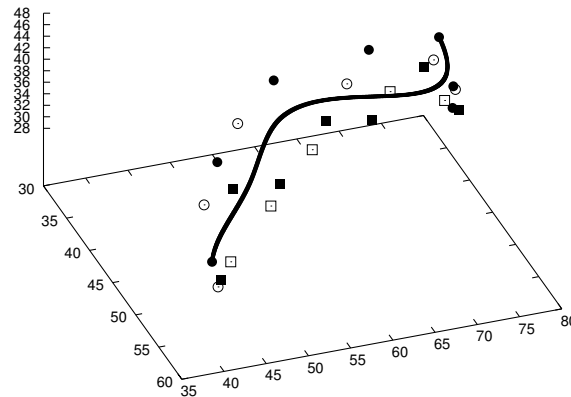


Figure 4.7: Generated NURBS and sampled points for Experiment 2 with sparse data set (least squares algorithm).

quired to grasp the small cubic box and to move it to its final configuration on the working plane, located on the other side of the obstacles, while avoiding the obstacles. Figure 4.4 shows a diagram with the resulting NURBS and the points sampled during the demonstration. The tight fitting of the sampled points highlights the effectiveness of the least squares algorithm.

Figure 4.5 shows the simulation environment with the approximating NURBS drawn in dark (blue) line, which was rendered using the NURBS interface provided by OpenGL; the center of the grasped object follows the curve. Figure 4.6 shows the execution of the task in the real workspace.

The second experiment comprised four demonstrations of the same task in an environment similar to that of experiment 1. Across the four demonstrations the user followed quite different paths to reach the final position, and one of the trajectories has strong "wiggles". Figure 4.7 shows the resulting NURBS and the points sampled from each trial (drawn with different symbols). The spatial sampling of the curve was

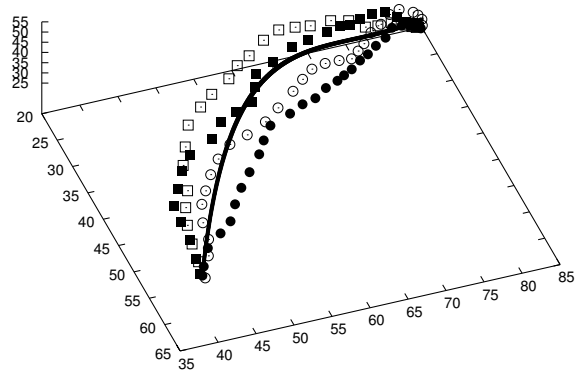
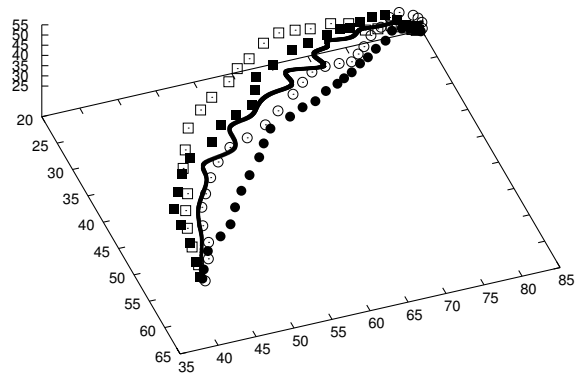


Figure 4.8: Generated NURBS and sampled points for Experiment 2 with dense data set (least squares algorithm on the top image and global approximation algorithm on the bottom image).

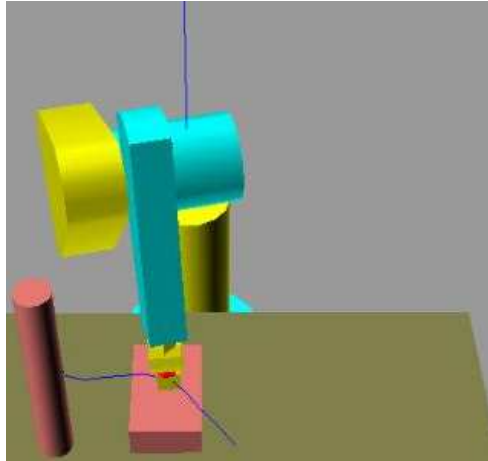


Figure 4.9: The simulation environment for Experiment 2 (global approximation algorithm).

kept low resulting in an average of 7 points for each demonstration. These results show the effectiveness of the least squares algorithm with sparse data, since the best-fit trajectory is smooth and correctly removes the jitter of individual demonstrations.

The same experiment has been exploited to assess the robustness of this algorithm with a dense data set, as shown in. An average of 26 samples were collected for each demonstration. Figure 4.8 shows the resulting NURBS approximated with both the least squares algorithm (left image) and the global approximation algorithm (center image). Figure 4.8 also shows the points sampled from each trial (drawn with different symbols) while figure 4.9 shows the simulation environment with the robot manipulator following the curve generated by the global approximation algorithm.

The NURBS resulting from the least squares algorithm is not satisfactory since it has wobbles, in particular in the central part of the trajectory, where the differences between the four user paths are remarkable. On the contrary the NURBS generated by the global approximation algorithm is smooth and correctly removes the jitter of individual demonstrations. User motion variability indicates that the robot can actually choose a path within a range of free space. The global approximation should

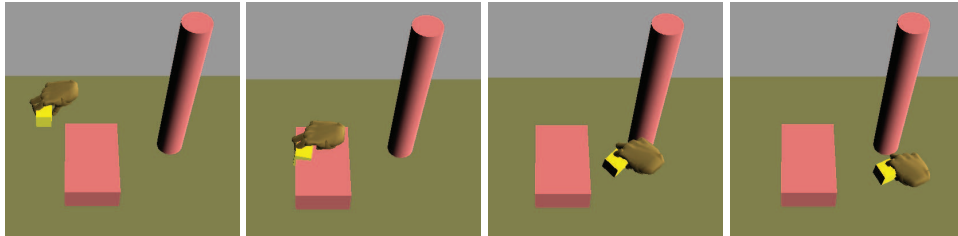


Figure 4.10: Task demonstration for Experiment 3.

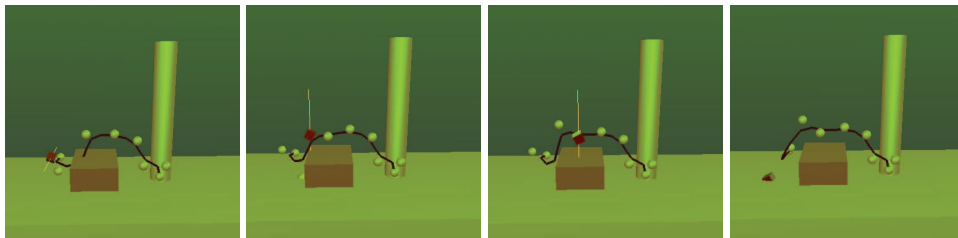


Figure 4.11: NURBS modification with the graphical editor.

be preferred as it takes advantage of this freedom to compute a smooth trajectory.

Experiment 3 consisted of a single demonstration, as in the first one. The user, while performing the task, (deliberately, in this case) moved the grasped object into an obstacle, as shown in figure 4.10. At the end of the simulation, the user decided to manually modify the resulting trajectory to amend the colliding path segment (figure 4.11). The PbD system provides an interactive editor that can be used to directly manipulate the generated NURBS by changing the position of some selected control points on the curve. The new curve is redrawn in real time. The final NURBS can be saved and used as input for the execution phase of the task in the real environment.

The last two experiments tested the effectiveness of the clustering and learning algorithms for stochastic trajectory approximation. Experiment 4 consisted of twelve demonstrations (each trajectory provided 159 samples). The clustering algorithm divided the initial dataset into three clusters of 5, 3 and 4 trajectories, respectively, as shown in Table 4.1.

Table 4.1: Results of trajectory clustering for Experiment 4.

Cluster	Trajectory	Log probability
1	10	-180.253
	12	-185.522
	6	-199.228
	1	-203.882
	8	-214.307
2	11	-152.138
	5	-213.171
	2	-231.993
3	3	-118.992
	7	-192.809
	4	-200.911
	9	-203.675

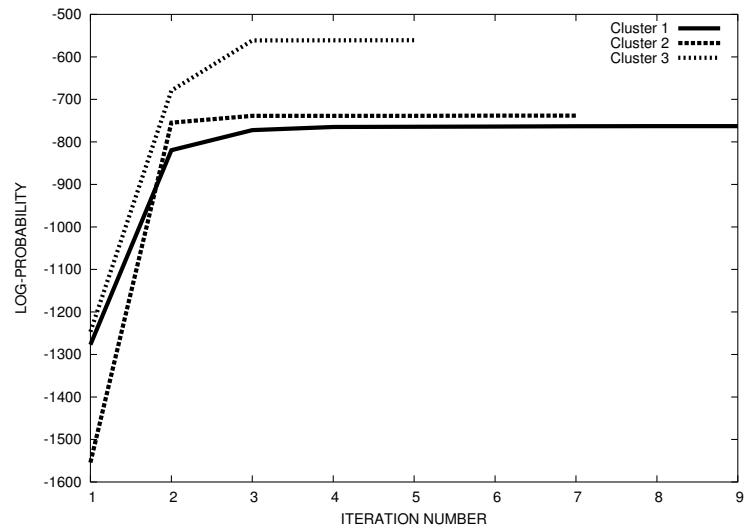


Figure 4.12: Iteration convergence graph for experiment 4.

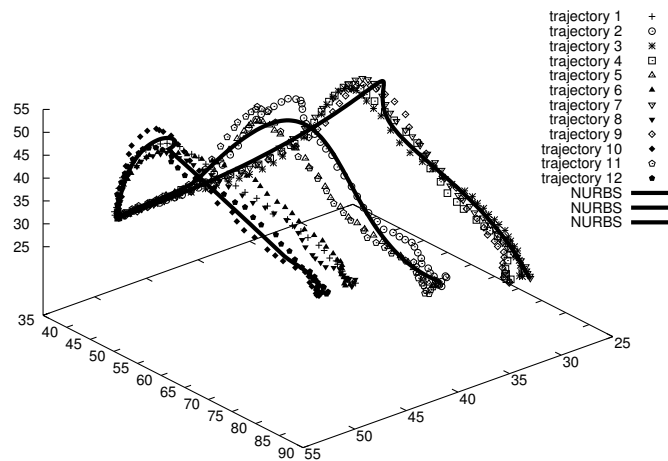


Figure 4.13: Generated NURBS curves and sampled points for Experiment 4.

After the preprocessing phase from the STFT analysis, each trajectory gave a set of 16-dimensional vectors that were then quantized with the *k-means* algorithm and a codebook of 64 vectors in each dimension. Three Hidden Markov Models were then trained on the clustered dataset with $Q = 10$ states and $M = 2$ gaussian components. Figure 4.12 shows the convergence of the training algorithm for the three clusters. The cumulative log-probability of each cluster converges in 4 iterations. Trajectories were then input back to their trained HMM for selection. Table 4.1 shows the log-likelihood of each trajectory within the clusters in descending order. The two highest score trajectories for each cluster were automatically selected as the most consistent ones and used as input for the approximation algorithm.

Figure 4.13 shows a diagram with the resulting NURBS curves for the three clusters and the points sampled during the demonstrations (drawn with different symbols). The spatial subdivision of the trajectories into the three clusters is clearly visible and the tight fitting of the sampled points highlights the effectiveness of the approximation algorithm. Figure 4.14 shows the simulation environment with the three

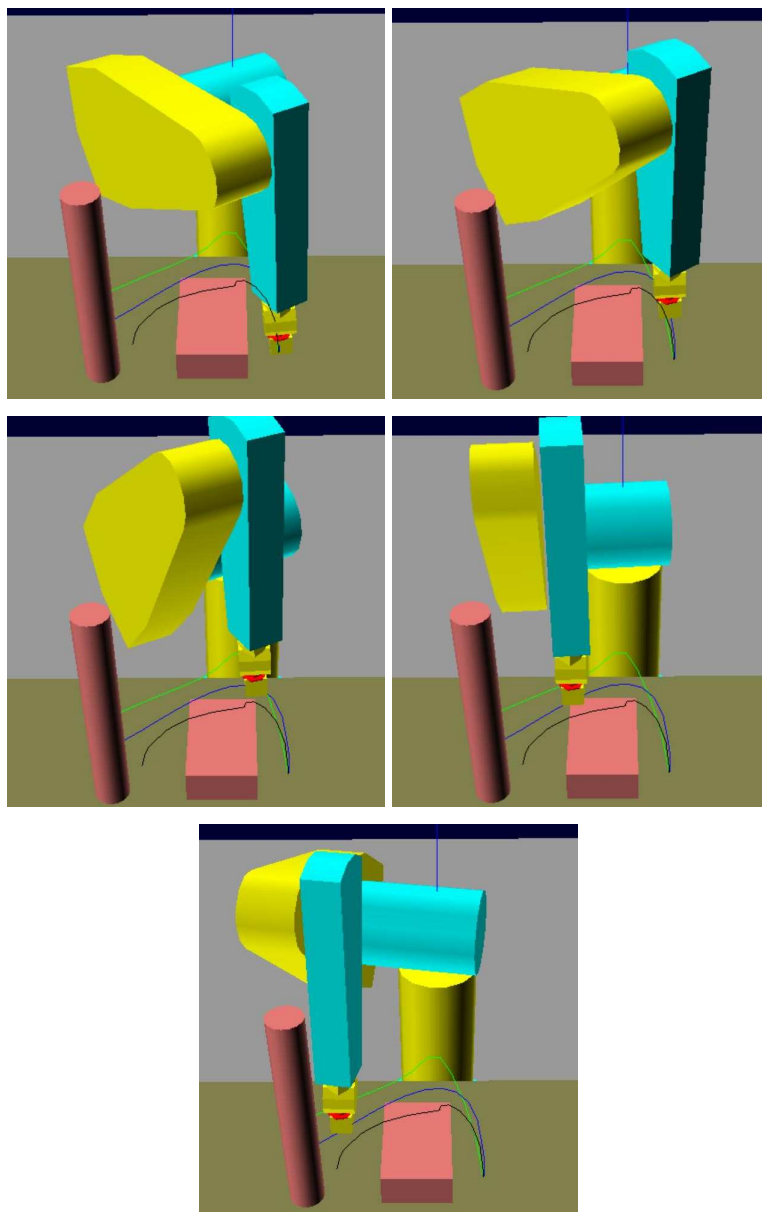


Figure 4.14: Simulation environment for Experiment 4.

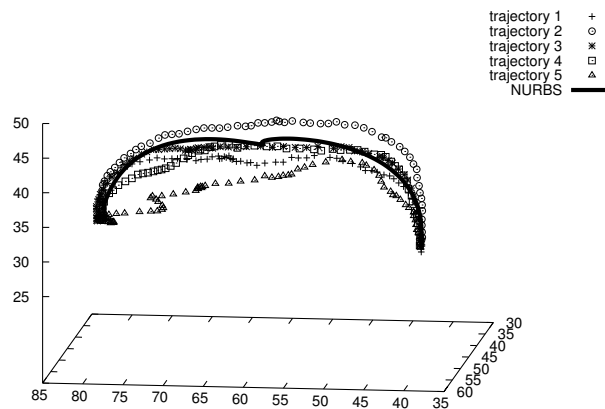


Figure 4.15: Generated NURBS and sampled points for Experiment 5.

approximating NURBS drawn with different shadings. The user selected the second trajectory which corresponds to the central dark (blue) path. In the simulation, the center of the grasped object follows the chosen trajectory. Although the proposed method for trajectory approximation is demonstrator dependent and is not guaranteed to generate collision-free trajectories, the likelihood of collision is drastically decreased by the clustering phase, and the operator can take advantage of the preventive simulation feature of the PbD environment to avoid unsafe path reconstructions.

Experiment 5 assessed the robustness of the stochastic algorithm for trajectory selection. The experiment comprised five demonstrations of the same pick-and-place task (each trajectory provided 200 samples). Across the five demonstrations the user followed quite similar paths to reach the final position, but the fifth trajectory had strong "wiggles". All trajectories were correctly classified into one cluster. Figure 4.15 shows the resulting NURBS and the points sampled from each trial. The trajectory selection procedure was repeated on the same dataset for 1000 times using the same parameters of the previous experiment. For the 94% of the times the fifth trajectory

was correctly classified as inconsistent and did not contribute to the final trajectory synthesis.

Chapter 5

Grasp Recognition for Pregrasp Planning by Demonstration

This chapter describes the grasp recognition module and the pregrasp trajectory generator for the Programming by Demonstration system. Grasp recognition and pregrasp trajectory planning are two issues of fundamental importance in robotic manipulation. In the approach adopted in this thesis, the system classifies the human hand postures in virtual reality. Previous research in grasp classification has never addressed the problem of grasp recognition in a virtual environment.

The algorithm for grasp recognition in VR has been experimentally evaluated both by skilled operators, *i.e.* with several hours of prior experience with the VR system, and by a group of untrained users, novice to the VR setup. Grasp recognition in virtual reality raises several problems that do not occur in a real environment. Objects can be occluded due to a limited view of the scene, feedback is usually limited, and manipulation is typically not subject to physical laws. Moreover, a training session is required to achieve adequate rate of correct classifications. In spite of these drawbacks, virtual grasping provides useful information about the contact points and the contact normals, which can be exploited for classification.

Besides the grasp classification procedure, a grasp mapping strategy and a pre-

grasp planner are presented. Grasp mapping is required to translate the recognized human hand posture, which is acquired from the glove input device, to the robot hand available in the current simulated setup. Pregrasp planning is necessary for an accurate positioning of the end-effector relative to the object to be grasped. The adopted solution is based on a trajectory generator exploiting NURBS.

The chapter is organized as follows. Section 5.1 provides an introduction to the problem of robot grasping. Section 5.2 reviews the state of the art regarding grasp recognition and grasp planning by demonstration. Section 5.3 describes the proposed algorithm for grasp recognition in virtual reality and provides an experimental evaluation. Section 5.4 describes the adopted solution for the grasp mapping problem and the trajectory generation technique for pregrasp planning.

5.1 Introduction to robot grasping

This section provides an overview to the problem of robot grasping, starting with a description of some known human-like robot hands. The phases of a generic grasping process are then introduced. Finally, the importance of abstract grasp classification for grasp synthesis is discussed and different taxonomies of human grasps are reviewed. In particular, Cutkosky's taxonomy is analyzed in detail.

The handling of parts and tools is the commonest action performed by industrial robots. The end effectors typically found in such robots are two-fingered, parallel-jaw grippers. Such grippers can not effectively handle a wide selection of objects. A solution is to build gripping devices that can approach the versatility and flexibility of the human hand. Several dextrous robot hands have been proposed in robotics literature; some examples are provided in the following. It was not until the early eighties that multifingered hands began to show up in laboratories. The Stanford/JPL hand [77] was one of the first human-like hands. It is a three fingered hand with three degrees of freedom per finger driven by a tendon scheme. Instrumentation includes tendon tension sensors, motor position encoders and fingertip tactile sensors. The Utah/MIT dextrous hand [78] has 16 degrees of freedom and each of the three fingers and

the thumb has 4 degrees of freedom. The DLR hand [79], developed at the German Aerospace Center, is a four-fingered human-like hand. The DLR hand has a total of 12 degrees of freedom and is about one and a-half the size of an average human hand. The Robonaut hand [80], developed at NASA's Johnson Space Centre, has five fingers and a total of 14 degrees of freedom and it is equivalent in size to a 95th percentile human man hand. The UB Hand (University of Bologna dexterous Hand) is an Italian robot hand whose development has been carried out since 1985 leading to different version of the hand. The Barrett hand, produced by Barrett Technology, is a three fingered device that has been simulated in this thesis and will be described in detail in section 5.4.1.

A commonly accepted way for the subdivision of the grasping process consists of three phases [81]:

1. *Pregrasp phase*: The first phase of grasping precedes the actual grasp. It is a combination of the trajectory of the hand (hand transportation) and the temporal changes in finger joint parameters in anticipation of the intended grasp. Preshaping is therefore a preparatory step aimed at satisfying physical constraints of the grasp.
2. *Grasp phase*: The pregrasp phase ends and the static grasp phase begins when the hand touches the object and has a stable hold of it.
3. *Manipulation phase*: This phase is characterized by hand motion resulting in the corresponding movement of the object in the environment.

Arbib et al. [82] made some important observations about human reaching and grasping:

- The reach (approach) and grasp movements occur in parallel.
- The reach is concerned with the object characteristics (*e.g.* shape, size and orientation).
- The grasp is also concerned with intended object usage.

Automatic grasp planning is a difficult problem because it requires reasoning in both the geometric domain (motion and contact constraints) and the dynamic domain (forces to be applied). Moreover, motion planning for grasping is made difficult by the large number of degrees of freedom involved. To make grasp planning tractable, rather more symbolic representations are required that capture only the main morphological characteristics of the objects to be grasped. One possible solution is to adopt an appropriate high-level control architecture which includes an abstract grasp classification mechanism. A second strategy that can reduce the complexity of an automatic grasp planner is the exploitation of information provided by the user demonstration of a grasping task. One of the contributions of this thesis is to propose a method for pregrasp planning by means of a trajectory reconstruction algorithm that approximates the hand paths demonstrated by the user. The method is coupled with two procedures, the first for grasp classification and the second for mapping.

Many attempts have been made to establish classifications of prehensile activity. One of the oldest and widely referenced taxonomies is that of Napier [83]. Napier defined the *precision grasp* and the *power grasp* as basic prehensile postures. Power grasps are distinguished by large areas of contact between the grasped object and the surfaces of the fingers and palm, and by a reduced ability to flexibility and manipulability of the objects. In precision grasps the objects are mainly held with the tips of the fingers and thumb. These grasps provide an high degree of manipulability and dexterity.

Many modern taxonomies extended Napier's concept by introducing additional distinctions regarding both object shape and size and also the way the hand applies the constraints. Cutkosky's taxonomy was built by observing real manufacturing tasks such as machining operations with metal parts and hand tools. Workers were observed while working and interviewed. In addition, their perceptions of tactile sensitivity, grasp strength, and dexterity were recorded.

Cutkosky's taxonomy consists of sixteen grasps. Power grasps are divided into two classes: non-prehensile grasps and prehensile grasps. Prehensile grasps are further divided into subclasses such as the thin lateral pinch grasp, the long prismatic

grasps and the compact circular grasps. Precision grasps are divided as well into two main classes, the compact circular grasps and the long prismatic grasps. A partial taxonomy of manufacturing grasps from Cutkosky's taxonomy, used in the classification experiments described in this chapter, is shown in figure 5.2. Cutkosky's taxonomy suffers from a number of limitations. Firstly it is incomplete as numerous everyday grasps, such as the grasp used in writing, are not included. Secondly, the proposed grasps have large variations that can be hardly taken into account with a static classification.

Iberall [84] proposed a grasp classification strategy in terms of opposition space and virtual fingers. One purpose of the work described in this chapter (see 5.3.1) is to combine Cutkosky's taxonomy with the classification proposed by Iberall.

Ikeuchi and Kang [81] proposed a grasp taxonomy which is based on the computation of the *contact web*. The contact web is defined as a 3-D graphical structure connecting the effective points of contact between the hand and the object. Grasp which involve the palm are classified as *volar grasps*, while others are called *non-volar grasps*. All volar grasps are power grasps and all but one type of non-volar grasps are precision grasps. The non-volar grasps are further classified as fingertip grasps and composite non-volar grasps. The fingertip grasps involve only the fingertips while the composite non-volar grasps involve, in addition, surfaces of other segments of fingers. One interesting property of the contact web is that for volar grasps it is spatially non-planar, whereas for most non-volar grasps it is approximately planar.

Kamakura [85] proposed a different classification that places no restrictions on the handled objects or the application domain. The classification comprises 14 types of grasps divided into four main categories: five power grips, four intermediate grips, four precision grips and one thumbless grip. The difference from the other taxonomies is that it mainly focuses on the purpose of grasps to distinguish between them. No conceptual distinction is made on the object shape or the number of fingers involved. The classification is also based on the hand shape and the contact points as the contact web based classification.

5.2 Related work

In this section some prior work on grasp classification and robot grasp simulators is discussed. Two different methods have been proposed for grasp recognition. The first strategy is based on static classification, while the second method relies on dynamic classification of grasp sequences.

In [86] static hand posture classification has been investigated using Neural Networks and relying only on angular data collected by gloves. The work [86] used Cutkosky's taxonomy [87] as the basis for grasp classification and obtained an overall result of about 90% of recognition accuracy. A similar approach has been adopted in [88].

Dynamic grasp recognition has been studied by Zöllner et al. [89], by Bernardin et al. [90] and by Ekvall and Kragić [91, 92]. In [89] the authors proposed a system handling pick and place tasks which was enhanced to gather dynamic grasps. Tactile sensors were mounted in a dataglove and basic grasps such as screwing and twisting were successfully recognized by a learning algorithm based on Support Vector Machine. In [90] the authors proposed a sensor fusion approach for dynamic grasp recognition using composite Hidden Markov Models. The system used both hand shape and contact information obtained from tactile sensors. Grasp recognition referred to twelve patterns according to Kamakura's taxonomy and achieved an accuracy of about 90%. Post-processing was required after dynamic gesture recognition to avoid misclassifications. In [92] a hybrid method for dynamic classification was presented, which combined both HMM classification and hand trajectory classification. Ten grasps were considered from Cutkosky's taxonomy and the results showed a recognition ability of about 70% for a multiple user setting.

A common drawback of static grasp recognition is the requirement of a proper segmentation algorithm to find ideal starting points for the analysis of the hand posture. One of the main objective of the system presented in this thesis is to show that static grasp recognition in virtual reality can achieve good performance since segmentation is easier in a virtual environment.

The system proposed in this thesis aims also at integrating both grasp recognition and robot pregrasp planning, as these two problems have often been decoupled in previous research. Only a few works tried to combine the two, such as the early work of Kang and Ikeuchi [81, 93, 94] that proposed a complete PbD system combining static grasp classification, based on the analytical computation of the contact-web, with grasp synthesis on a real robot manipulator.

The system has been validated through the advanced robot simulator described in the previous chapters. The simulator comprises a robot manipulator and a complex robot hand, as will be shown in section 4. Few free robotics simulators allowing grasp simulation are currently available. One of the most promising is Graspit! [95], a versatile tool that focuses on grasp analysis. Graspit! exploits a dynamic engine and a trajectory generator together with a user friendly user interface.

5.3 Grasp recognition in virtual reality

The virtual environment used in the experiments is shown in figure 5.1. It includes a working plane, a set of standard geometrical objects such as two spheres and two cylinders of different size, and a classical daily life teapot. A subset of eleven grasps from Cutkosky's taxonomy were selected for grasp recognition. The resulting grasp tree is shown in figure 5.2 along with the labels and example images of the grasps. The five grasps belonging to Cutkosky's taxonomy not included in the selected tree are the precision circular disk grasp and four power grasps, namely the circular disk, the adducted thumb grasp, the light tool grasp and the small diameter grasp. These grasps were not included due to the difficulty to differentiate them with other grasps in a practical scenario.

5.3.1 Grasp classification

To reduce the dimensionality of the input state space a preliminary analysis of the variance of the joint angles was carried out. Two experienced users (a male and a female) were asked to replicate each of the virtual grasps ten times. The results showed

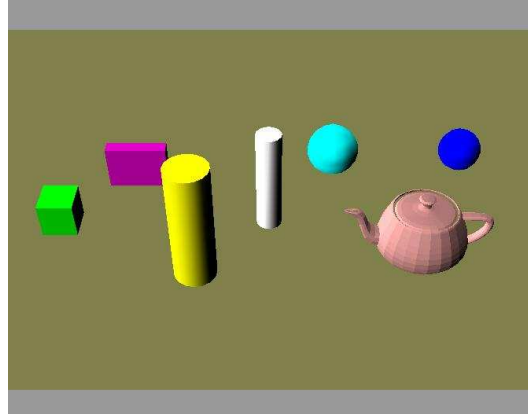


Figure 5.1: The set of objects.

a strong accordance between the two users. The histogram in figure 5.3 shows the mean variance. Three metacarpophalangeal joints (middle, ring and pinkie finger) have a variance lower than $0.1rad$ (joints 8, 12 and 16 in figure 5.3). Therefore these joints were not considered in the grasp classification algorithm, and the data vector representing the hand posture was restricted to 19 values.

The proposed classification algorithm consists of two steps. Firstly, a nearest neighbor algorithm is applied to compute the distance between the hand posture to be classified and each of the 11 patterns representing the recognizable grasps, which were collected in a training session. The algorithm then sorts the grasp indexes starting from the nearest candidate in descending order. The distance between two pattern is computed in the joint space as the euclidean distance between the two vectors of joint angles.

After the scoring phase an heuristic decision process is applied, which is based on a set of predefined rules with the purpose of disambiguating between possible misclassifications. The second phase of the algorithm improves the robustness of the classification, as will be shown in the next section. The heuristic rules exploit information about the contacts and the normals at the virtual contact points. It is assumed that the number of contacts equals the number of elements of the hand (*e.g.*

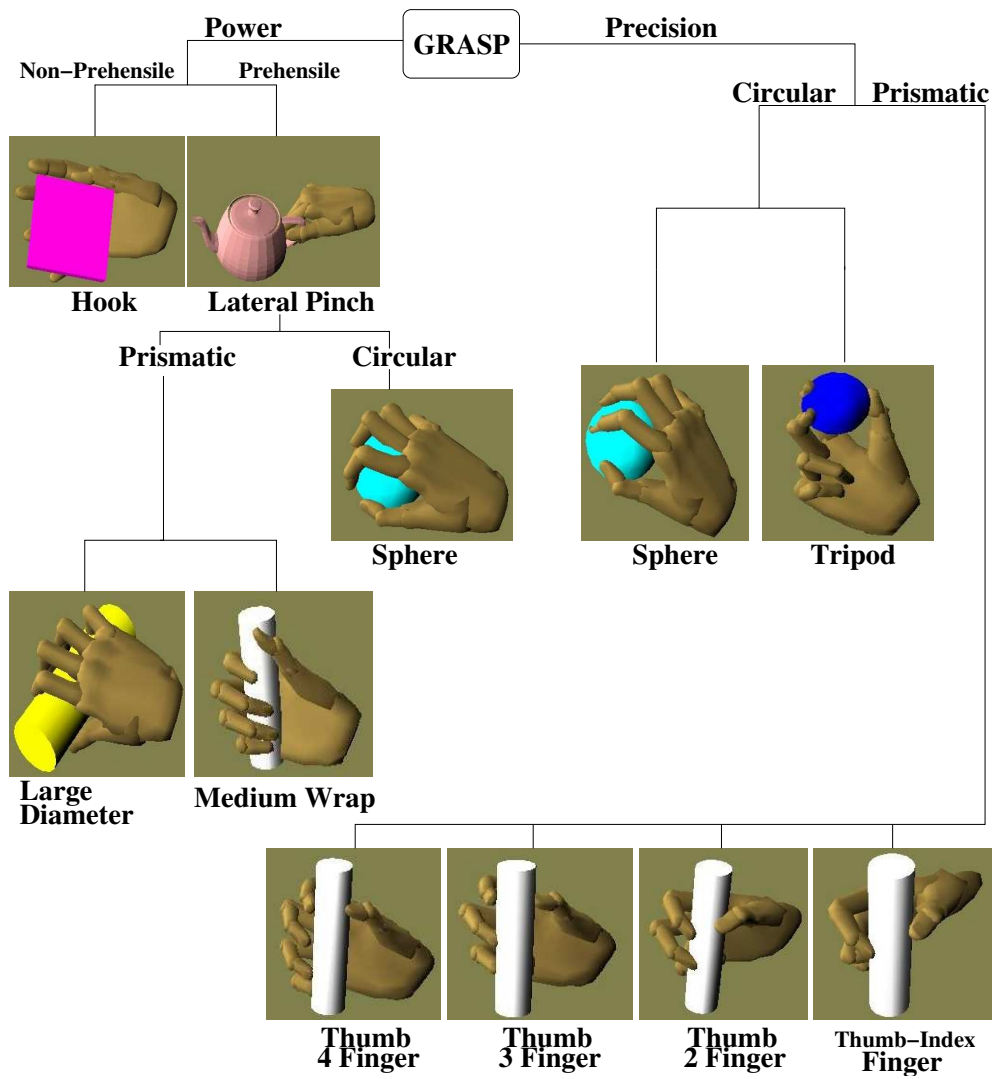


Figure 5.2: Grasps set.

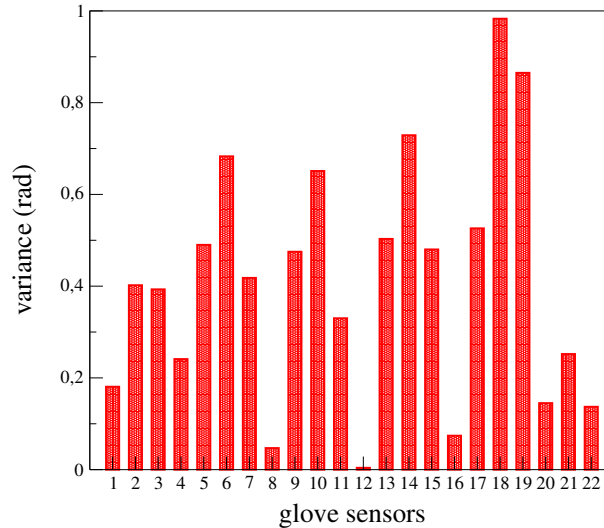


Figure 5.3: Mean variance of the joint angles for two experienced users.

phalanges) colliding with the grasped object. Some examples are provided in the following.

To disambiguate between a circular power grasp and a circular precision grasp the system checks the total number of contacts. If this number is greater than 10 the algorithm classifies the grasp as a power grasp, otherwise the grasp is classified as a precision grasp. To disambiguate between two similar grasps belonging to the same side of the grasp tree, the system looks for similarities in the orientation of the contact normals. For example, this strategy is applied to disambiguate between the the medium wrap and the lateral pinch grasp. The same happens for the precision tripod grasp and the thumb two-finger grasp. Table 5.1 contains the complete set of applied heuristics.

The above heuristic rules, based on the orientation of the contact normals, can be interpreted in terms of virtual fingers. Virtual fingers were first introduced by Arbib, Iberall et al. in [82]. A virtual finger is a group of real fingers acting as a single functional unit. This concept can be used to formally characterize different types of

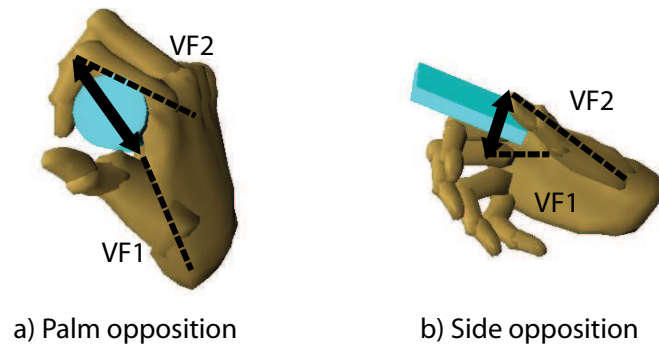


Figure 5.4: Palm and side opposition grasps.

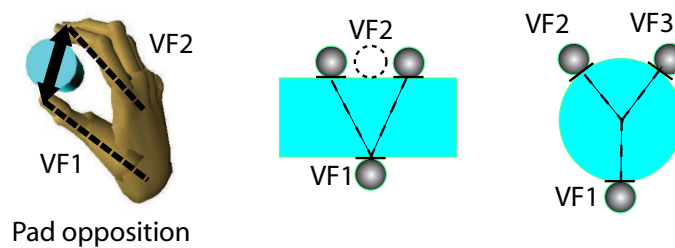


Figure 5.5: Pad opposition grasp (left image) with two examples: two finger prismatic precision grasp (central image) and tripod grasp (right image).

grasps in an abstract way. In [84] Iberall also showed that hand postures of different taxonomies, including Cutkosky's classification, can be described in terms of opposition between virtual fingers. The medium wrap and the lateral pinch grasp, previously cited, are easily identifiable by different types of opposition. The same consideration holds for the precision tripod grasp and the thumb two-finger grasp.

The medium wrap grasp exhibits a palm opposition between two virtual fingers (VF), the palm (VF1) and the digits (VF2). An example of virtual grasp with palm opposition is shown in figure 5.4a. In palm opposition "the object is fixed along an axis roughly normal to the palm of the hand" [84]. The lateral pinch grasp has two types of opposition concurrently, a palm opposition and a side opposition between the thumb (VF1) and the side of the index finger (VF2), which makes it different

Table 5.1: *Applied heuristics for ambiguous grasps.*

Ambiguous Grasps	Heuristics
3 - 8	grasp 3 if $\#contacts > 10$
3 - 6	grasp 3 if $\#contacts > 10$
3 - 2	grasp 3 if $\#contacts > 5$
5 - 2	check contact normals
7 - 10	check contact normals

from the medium wrap as shown in figure 5.4b. Opposition occurs primarily along an axis transverse to the palm. Both the tripod grasp and the thumb two-finger grasp are precision grasps that exhibit a pad opposition (figure 5.5, left image). Opposition occurs along an axis roughly parallel to the palm. However, the thumb two-finger grasp can be interpreted as a two virtual finger grasp (figure 5.5, central image), while the tripod grasp can be classified as a three virtual finger grasp [96], as shown in figure 5.5 (right image).

The given interpretation of grasps in terms of a combination of opposition between virtual fingers suggests that the contact normals, expressed in a reference frame relative to the hand, can be exploited to disambiguate between pairs of grasps. From the information about the contact normals, which is provided by the collision detection engine, it is indeed possible to determine the type of opposition, the number of virtual fingers and therefore the class of the grasp. In particular, the heuristic algorithm defines cones of acceptance for the orientation of the contact normals for each type of recognizable grasp, which provide some degree of tolerance between slightly different grasps.

5.3.2 Experiments

Two experienced subjects and ten unexperienced subjects (five males and five females) participated to the recognition experiment. The mean age was 23 years. Popu-

lation mainly consisted of students of the University of Parma. The CyberTouch was calibrated for each user at the beginning of the session. Moreover, each subject performed a short training session before the experiment, which consisted in a few trials for every grasp type. The experiment consisted of 44 grasp recognitions for each user. Subjects were asked to reproduce each grasp 4 times in a random order. The users were also allowed to change the orientation of the virtual camera in the environment. Table 5.2 summarizes the overall results and provides the recognition statistics for individual grasps considering unexperienced subjects only. Table 5.3 shows the confusion matrix, where entries represent the numbers of trials for a particular grasp (row) that were misclassified to another grasp (column).

Table 5.2: *Results of grasp classification.*

Case study	Mean recognition rate
Experienced users	94%
Unexperienced users	82.8%
Power Grasp	81.5%
1 Hook	87.5%
2 Lateral Pinch	87.5%
3 Circular Sphere	72.5%
4 Large Diameter	92.5%
5 Medium Wrap	67.5%
Precision Grasp	84.2%
6 Circular Sphere	77.5%
7 Circular Tripod	90%
8 Thumb-4 Finger	70%
9 Thumb-3 Finger	100%
10 Thumb-2 Finger	75%
11 Thumb-Index Finger	92.5%

The results of the classification algorithm were promising, as the worst case for the mean recognition rate was 67.5% for the medium wrap power grasp. The performance of the two expert users (94%) was significantly better than the performance of the unexperienced ones (82.8%). Grasp classification for the skilled users was car-

Table 5.3: *Confusion Matrix for grasp classification.*

	1	2	3	4	5	6	7	8	9	10	11
1		3	1			1					
2					3			1			1
3		3			3	2			2	1	
4						3					
5			8					2			3
6			9								
7		1								3	
8			10	1		1					
9											
10							10				
11					3						

ried out with their own training data, while classification for the unskilled ones was carried out with the dataset collected by one of the expert users, so as to simulate a practical scenario where ordinary users cannot be asked to spend too much time for the training session. Table 5.3 shows that the power circular sphere grasp had the highest number of misclassifications. This evidence suggests that other heuristic criteria should be investigated to further improve the grasp classification algorithm.

The goodness of the results is confirmed by the low variance of the recognition rate across the individual grasps, and by the evidence that there were no differences for the classification between power and precision grasps. The algorithm has also proven rather robust to varying object sizes, as the users while grasping were free to choose between the two cylinders and the two spheres in the environment. Some tests were finally conducted by removing the use of the heuristic rules in the grasp classification process. In this case, the recognition rate decreased by about 20%, confirming the importance of the second step of the classification algorithm.

5.4 Pregrasp planning by demonstration

Learning preferential approach directions for object grasping is a fundamental issue in robot manipulation as it can simplify the problem of finding stable grasps. Usually, in a complex environment grasping is constrained by occlusions. A pregrasp planner that imitates the motion of the user is therefore a tool that can help to reduce the search space for feasible grasps. In this section the problem of grasp mapping is first investigated in relation to the available robotic setup, then a pregrasp trajectory generator is presented with examples in a simulated workspace.

5.4.1 Grasp mapping

Grasp mapping is required in order to overcome kinematics dissimilarities between the human hand and the robot gripper used in the actual manipulation phase of the task. Translation of the chosen hand pose to the robot hand is achieved at the joint level.

The gripper used in the current simulation setup is the Barrett hand (figure 5.7), which has three fingers and four degrees of freedom. The hand has one flexion degree of freedom (dof) for each of the three fingers. The fourth dof controls the symmetrical abduction of the two lateral fingers around the fixed thumb. Each finger has also a distal coupled joint. Figure 5.6 shows four examples of grasp mapping (a large diameter grasp, a thumb-2 finger grasp, a spherical and a precision power grasp) along with the corresponding images of the CyberTouch.

In the previous examples, the proximal interphalangeal joints of three fingers (thumb (θ_1), index (θ_2) and middle finger (θ_3)) were mapped to the flexion joints of the robot hand (α , β and γ respectively), while the thumb abduction joint (θ_4) was mapped to the last dof, *i.e.* the spread angle (η). The mapping function MP is a proportional control defined as:

$$MP : (\theta_1, \theta_2, \theta_3, \theta_4) \rightarrow (\alpha(\theta_1), \beta(\theta_2), \gamma(\theta_3), \eta(\theta_4))$$

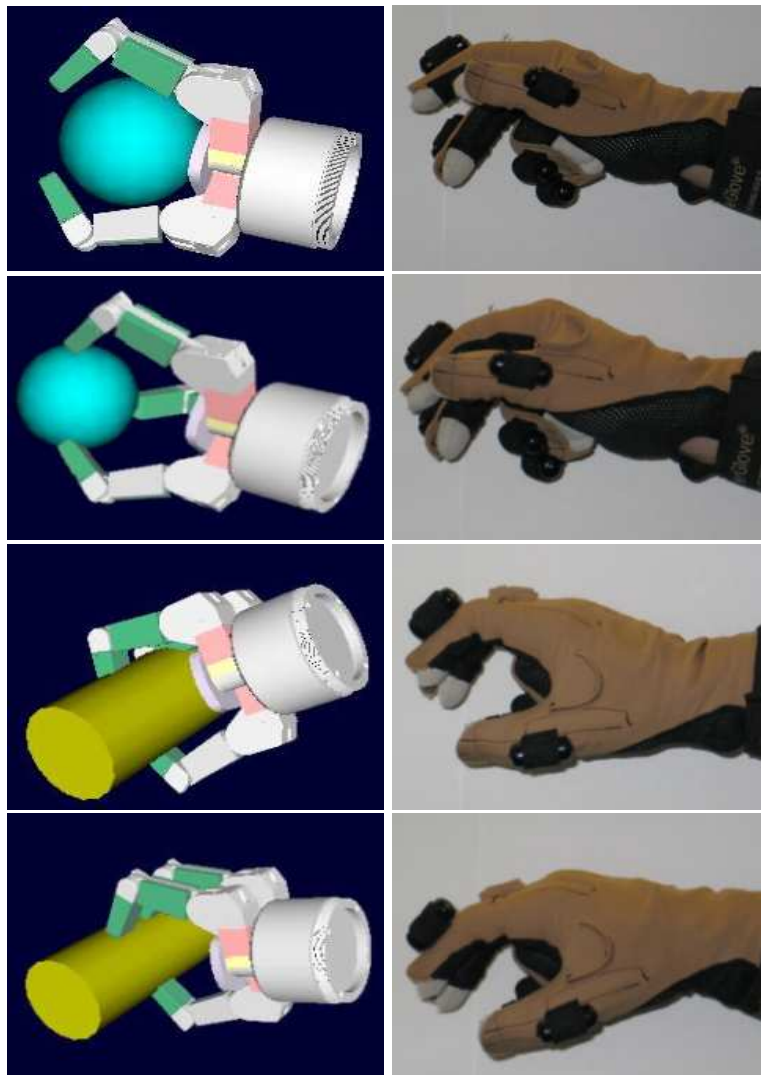


Figure 5.6: Examples of grasp mapping.

where:

$$\begin{cases} \alpha = k_1\theta_1 \\ \beta = k_2\theta_2 \\ \gamma = k_3\theta_3 \\ \eta = k_4\theta_4 \end{cases}$$

The k_i are constant values compatible with the maximum angular amplitude of the sensors.

The use of a fully instrumented glove allows a flexible customization of the mapping strategies, as the joints correspondences can be easily changed according to the preferences of each user. As the Barrett hand cannot replicate all the recognizable grasps, similar grasps are grouped together after recognition. For example, all prismatic precision grasps are grouped into a single thumb-2 finger class.

The grasp mapping module is used for the off-line acquisition of data for each user. The grasping data, namely the four degrees of freedom that describe each grasp are stored in a database along with the label of the corresponding classified grasp. The database is queried after the demonstration phase of each manipulation task. Data collected from the database, along with the samples that describe the pregrasping trajectory, are sent to the pregrasp planner that generates the robot commands.

5.4.2 Pregrasp planning

The pregrasp planner has been tested in a simulated environment comprising a Puma 560 robot arm and a Barrett hand as its end effector. The robot manipulator is controlled in the cartesian space by an inverse kinematics algorithm. The tool point of the Puma arm follows a parametric curve that imitates the pregrasp path demonstrated by the user. While in chapter 3 the orientation of the end effector was constrained to be parallel to the working plane (xy -plane), in this chapter this constrain has been released. Therefore the end effector is able to reach the proper orientation that corresponds to the actual orientation of the human hand in the demonstration phase. Once the tool point reaches the end of the trajectory with the proper orientation, which is also given by the orientation of the tracking device, the joints of the Barrett hand are

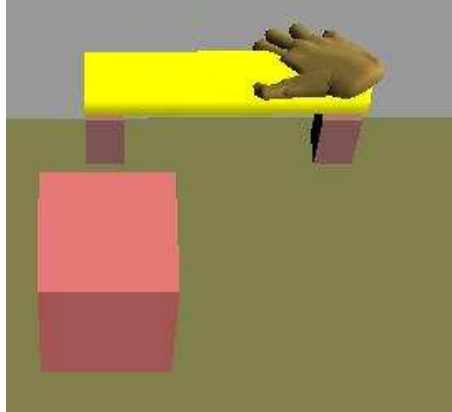


Figure 5.9: Grasp demonstration for experiment 1.

moved according to the corresponding values stored in the database. In the current setup preshaping is stopped at the 90% of the flexion values demonstrated by the user. The final approach to the object is demanded to a grasp execution phase which requires suitable control algorithms and will be investigated in the future.

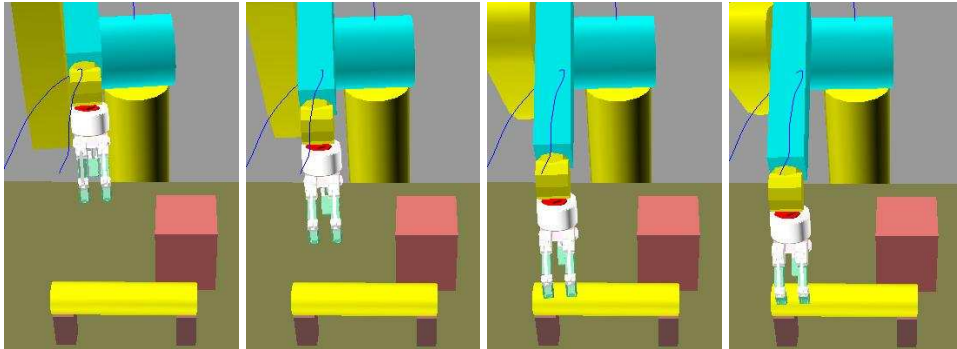


Figure 5.10: Pregrasp simulation for experiment 1.

NURBS curves are used for trajectory reconstruction exploiting the global approximation algorithm with error bound. Figure 5.10 shows an example of a grasping task, and a sequence of images taken from the corresponding robotic simulation.

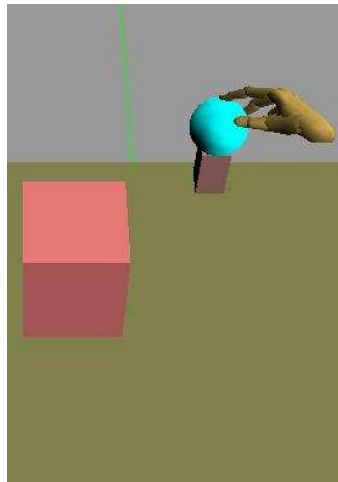


Figure 5.11: Grasp demonstration for experiment 2.

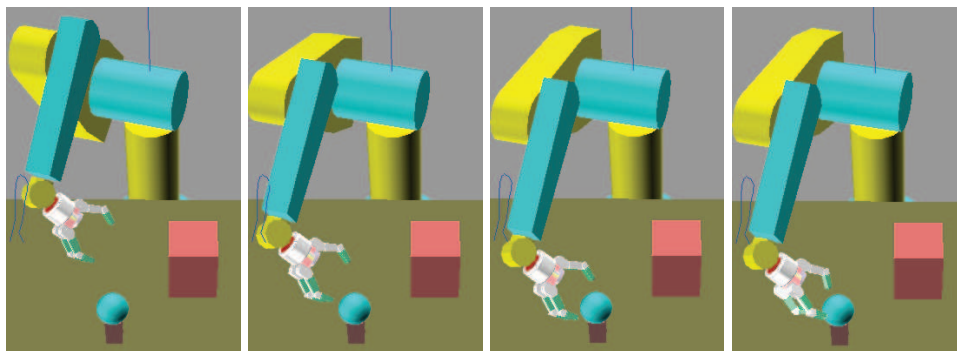


Figure 5.12: Pregrasp simulation for experiment 2.

The environment comprises a yellow cylinder to be grasped, which is stacked onto two fixed supporting boxes, and an obstacle. The provided demonstration shows a prismatic precision grasp (thumb-2 finger). Execution in the simulated environment displays the pregrasp trajectory (shown in blue or dark) followed by the end effector, that was rendered using the NURBS interface provided by OpenGL. Figure 5.11 displays the grasping phase for a second experiment where a sphere, in the same environment of the first experiment, is grasped with a tripod grasp. Figure 5.12 shows the simulation of the pregrasp for the second experiment.

Chapter 6

Evaluation of Virtual Fixtures

In this chapter the effectiveness of several types of virtual fixtures is evaluated in the robot programming by demonstration interface. While all types of virtual fixtures examined yield a significant reduction in the number of errors in tight tolerance peg-in-hole tasks, color and sound fixtures generally outperform a tactile fixture in terms of both execution time of successful trials and error rate. It has been found also that when users perceive that the task is very difficult but the system is providing some help by means of a virtual fixture, they tend to spend more time trying to achieve a successful task execution. Thus, for difficult tasks the benefits of virtual fixturing are better reflected in a reduction of the error rate than in a decreased execution time. These trends are related to the limitations of currently available interfaces for human-robot interaction through virtual environments and to the different strategies adopted by the users to cope with such limitations in high-accuracy tasks.

6.1 Virtual Fixtures

In PbD in simulated environments, human-robot interaction is essentially mediated by the virtual environment. It involves user-interfaces design, human factors, performance evaluation and cognitive psychology. Most traditional human-computer inter-

action systems focus on the design of applications that convey to the user only graphical information through a visual channel. To enhance the perception of advanced data sets such as a virtual environment, a graphical interface can be augmented with additional sensory aids such as auditory and haptic feedbacks [97, 30].

Synthetic aids integrated into a virtual environment are called *virtual fixtures*. In his seminal work, Rosenberg [98] defines virtual fixtures as “abstract sensory information overlaid on top of reflected sensory feedback from a remote environment.” These fixtures can be composed of “haptic, visual, auditory, and tactile sensations, used alone or in cross-modal combinations” [98]. Virtual fixtures have been later extended from telerobotic manipulation to virtual and simulation environments in a number of systems [19, 30, 99, 100, 101]. Virtual fixtures are thus perceptual overlays designed to reduce the physical and psychological demands that arise during the execution of a task. Virtual fixtures have been proven to improve the operator’s performance in terms of both speed and accuracy, since they can alert the user for changes in the state of the environment and support hand-eye coordination for object manipulation tasks [98, 16].

This chapter addresses the problem of evaluating a set of virtual fixtures (implemented using visual, auditory, and tactile sensory feedback) in the demonstration environment. Past studies have found that the quantitative impact of virtual fixtures is task dependent [98, 99, 102]. Thus, the objective of this study is to find whether there is a dominating artificial fixture that can be introduced in the demonstration environment to improve the effectiveness of user interaction. Another primary goal is to determine to what extent a VR glove is suitable for high precision manipulation tasks.

In [103], the benefits of the vibrotactile feedback available in a VR glove have been investigated in a few, relatively simple manipulation tasks which can occur in a PbD interface exploiting a virtual environment. The results demonstrated that this type of haptic feedback decreases the average demonstration times. The focus of the experimental evaluation of this chapter is a specific peg-in-hole task at higher difficulties, introducing additional virtual fixtures such as visual and auditory aids,

both individually and in multimodal combination. Moreover, the application of Fitts' law is investigated as a tool to predict the effectiveness of different types of virtual fixtures at increasing levels of difficulty. Fitts' law is a well-established paradigm for testing human-computer interaction performance [104, 105].

The chapter is organized as follows. Section 6.2 surveys relevant past work on virtual fixtures. Section 6.3 describes the experimental setup and the protocol for the proposed tasks, whereas section 6.4 discusses the results and reports the users' comments. Section 6.5 analyzes the experimental results by applying the method proposed by Fitts. Finally, section 6.6 draws some conclusion and interpretation.

6.2 Related work

Virtual fixtures have been used in several research projects to improve users' performance in task execution. In the following, representative papers investigating virtual fixtures are discussed. It should be remarked that none of these papers investigated the relative or combined effectiveness of multiple virtual fixtures in a quantitative manner in virtual environment for programming by demonstration.

Several studies have evaluated the benefits of virtual fixtures for teleoperation tasks, to enhance teleoperator performance and improve telepresence. Sayers [106] described a teleoperated system for undersea operations enhanced with visual and force feedback. The system confines the boundaries of the end-effector and repels it from obstacles. He also introduced a general terminology for synthetic fixtures, including the definition of point-to-point fixture and closest-surface-surface fixture. Payandeh and Stanisic [100] investigated the applications of visual and force cues as a control aid for training and telemanipulation. They found that these cues improve speed, precision, and reduce operator's workload. Rosenberg [98] proposed an experimental setup for telepresence performance assessment in a peg-in-hole task. He showed that virtual fixtures combining impedance surfaces and auditory feedback increase performance by up to 70%.

Yu et al. [107] proposed a telemanipulation enhancement system. User's motion

intention is combined with real-time environment information for applying appropriate fixture assistance.

Aarno et al. [108] proposed an advanced system for machine-assisted teleoperation tasks where virtual fixtures are generated adaptively. The authors presented a system that uses a standard robotic manipulator to assist humans in various assembly tasks. A high-level task is segmented to subtasks where each of the subtasks has a dedicated virtual fixture obtained from 3D training data. A state sequence analyzer, based on a combination between Hidden Markov Models and Support Vector Machines, learns what subtasks are more probable to follow each other. Specific virtual fixture, corresponding to the most probable state, are then automatically applied.

Virtual fixtures have also been successfully exploited in PbD systems to reduce demonstration time and improve task precision. Lloyd et al. [19] developed a programming by demonstration system for executing contact tasks in a virtual environment. A 3D contact simulator provided first order dynamics and allowed the operator to easily constrain the motion of the objects. Li and Okamura [102] presented a system exploiting a Hidden Markov Model for recognition of operator's motion. Segmentation results are used to provide online adaptation of virtual fixtures to the user for tasks that involve path following and avoidance. Recently, Shing et al. [101] developed a VR-based hand rehabilitation system exploiting auditory and haptic signals. A tracker and a data glove were included in this system. Forty volunteers were recruited to participate in a pick-and-place procedure, with three levels of difficulty and four feedback modes. Task time and collision frequency were the parameters used to evaluate their manipulation performance. The authors concluded that the haptic feedback is a significant signal for improving a subject's performance at high difficulty level. The main difference between the work of Shing and the work presented in this thesis is that haptic feedback in [101] was generated by text displayed on a screen, while in the experimental setup used for this thesis haptic feedback is generated by the vibrotactile actuators of the glove.

The following papers investigated the effect of virtual fixturing based on force feedback on specific tasks. Boud et al. [109], investigated the influence of direct hap-

tic feedback in assembly tasks such as the "Tower of Hanoi". Sallnäs and Zhai [110] studied the role of virtual fixtures for collaborative complex tasks such as handing over objects between two users. They demonstrated that force feedback decreases the error rate but does not influence the time required for passing objects. Dennerlein et al. [111] experimented the use of a force-feedback mouse in a desktop computer human interface for steering and combined steering-targeting tasks, and showed that its use can reduce task execution time.

6.3 Methods and protocols

The case study selected for evaluation of the demonstration environment enhanced with virtual fixtures is a standard peg-in-hole task where the user must first grasp a cylinder and then put it into a container. This task is often adopted as a benchmark for fine manipulation capabilities (*e.g.*, see [98]). Moreover, decreasing the clearance between the peg and the hole allows generation of a series of tasks of increasing difficulty for trend analysis.

6.3.1 Feedback type

The system defines a circular acceptability zone for the insertion task near the center of the container. When the object is released, if its center lies within the acceptability zone the performed task is classified as successful. Tolerance in the vertical direction and size of the peg are kept constant.

Depending on the experiment, one or more virtual aids, described in the following, are activated whenever the center of the cylinder lies inside the acceptability zone. The operator can take advantage of this explicit information by immediately releasing the object, thereby decreasing task execution time in the virtual environment. Around the insertion point the profile of the feedback is mapped with a simple step function that activates the type of feedback under investigation inside the acceptability zone and deactivates this feedback outside. More specifically, the system can incorporate one of three types of virtual fixtures. The first type of virtual aid is

implemented in the system by exploiting the vibrotactile feedback available in the CyberTouch glove. When this fixture is activated, all the actuators are programmed to vibrate at a constant amplitude of $0.3 N$ peak-to-peak.

It has been found that users can discriminate only few levels of vibration intensity. Hence, in this research vibration has been exploited as an on-off stimulus returned at a constant intensity when active. It could be argued that vibration is a poor replacement for full tactile sensation. With currently available technology, however, vibratory feedback is the only type of tactile feedback conveniently integrated into VR gloves [112]. Alternative devices provide force feedback by means of bulky and expensive exo-structures. These devices might be required for delicate teleoperation tasks, but their additional cost does not seem justified in a PbD system, where the user is essentially providing geometrical information.

The second type of aid is a sound feedback at a constant frequency of $1 kHz$ that is generated by the speakers of the workstation. The third type of aid is a visual feedback implemented by changing the color of the cylinder whenever it enters the acceptability zone.

In absence of any virtual fixture, the user must complete the peg-in-hole task relying only on the 3D visual feedback of the virtual workspace.

6.3.2 Peg-in-hole task

The test environment consists of the virtual hand, a working plane, the cylinder to be grasped, and the container.

The testing software for the experiments operates in three phases. First, the program presents to the test subject a training environment that allows the user to practice peg-in-hole tasks with the different virtual aids at two fixed and simple levels of difficulty. Second, the program performs the actual tests; during these tests, the application records the time required to complete individual trials and sequences of trials until success, along with the accumulated errors. Finally, these data are available for statistical analysis and are processed after the experiments.

In all experiments, subjects were asked to grasp the peg, transport it towards the

container, and insert the peg into the container. Subjects were also asked to insert the peg from the top of the container. However, the system did not constrain the motion of the peg in the virtual environment. Hence, moving the grasped object through the side of the container was allowed.

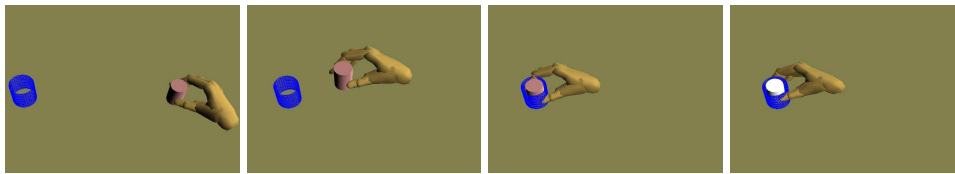


Figure 6.1: Example of a peg-in-hole task in Experiment 1. The final image shows the activation of the additional visual fixture based on color.

Three distinct peg-in-hole experiments have been designed. The first two experiments (Experiments 1 and 2) investigated the relative effectiveness of individual virtual fixtures. The third experiment (Experiment 3) investigated the combined effect of multiple virtual fixtures. Separate groups of users were assigned to each experiment, so that each user participated in only one experiment. Figure 6.1 shows a sequence of images from Experiment 1.

Experiments 1 and 2 are similar, although with some operative differences between them. In both experiments, subjects were presented with five levels of difficulty in increasing order. For each level of difficulty, the user had to complete four correct insertions for each kind of feedback (no aid, vibration, color, sound), for a total of eighty successful field trials. Additional trials were required in case of errors, as described later. For each user, the feedback type was presented in random order at the first level of difficulty, and this order was then maintained at the remaining levels.

After successful completion of four peg-in-hole tasks, a new task was presented to the subject changing the feedback type. After all feedback types had been examined, task difficulty was increased by reducing the acceptability zone. If the user made 15 consecutive errors, the current task was aborted and the next task was presented. A maximum of 60 trials (including errors) for each level of difficulty and feedback type was thus possible. This process continued until the whole test set was completed by

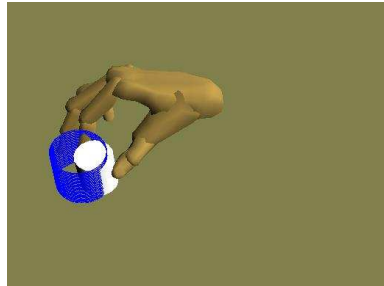


Figure 6.2: Insertion example with larger acceptability zone and visual fixture in Experiment 2.

the user.

When the virtual hand grasped the cylinder the timer was started. When the cylinder was released the timer was stopped regardless of the current position of the peg. If the subject released the object outside the acceptability zone, the error count was incremented and the user had to repeat the trial (up to the maximum number of 15 consecutive errors). The system notified the subject of error occurrences through a visual message on the screen. If the subject released the object inside the acceptability zone, the system recorded both the completion time of the successful trial and the total completion time, *i.e.*, the aggregate time including the previously failed attempts.

In all experiments, the position of the container on the working plane was such that the initial distance h between the cylinder and the container was 90 *cm* in the virtual workspace. The location was chosen to provide a good point of view and a short distance between the transmitter and the receiver of the tracker in the insertion phase, resulting in a good accuracy of sensor readings (to the extent allowed by the FasTrak). The container was drawn in wireframe to provide visual help to the operator during the insertion phase even with no fixture activated.

In Experiment 1 the inner radius of the container is reduced at each difficulty level and the acceptability zone is set equal to this radius. In Experiment 2 an additional aid for the users, typical of virtual environments, is introduced. The acceptability zone is set larger than the size of the container, thus releasing the cylinder partially outside the

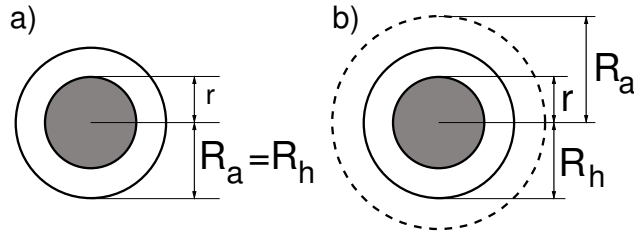


Figure 6.3: Geometrical features of Experiment 1 (left, a) and Experiment 2 (right, b).

border of the container but with the center within the admitted tolerance still leads to a successful classification of the peg-in-hole task (Figure 6.2). In summary, the purpose of Experiment 2 is to investigate the extent at which larger virtual targets improve user's performance. This capability is only available in VR-based PbD interfaces and could possibly improve PbD performance in non-cluttered environments.

Figure 6.3 shows the geometrical features of Experiments 1 and 2. Cylinders in gray represent the peg (with radius r), solid circles represent the hole (with radius R_h), and the dashed circle the acceptability zone (with radius R_a). The insertion task is classified as correct if the following condition holds:

$$\|O - o\| < \alpha(R_h - r)$$

where O is the center of the container and o is the center of the peg. α is a constant that was set as $\alpha = 1$ in Experiment 1 and $\alpha = 2$ in Experiment 2, yielding $R_a = R_h$ and $R_a = 2R_h - r$ respectively. The distance was computed in 2D, since the tolerance in the vertical direction was kept constant.

Experiment 3 evaluates the effect of multiple virtual fixtures at four levels of difficulty. The same peg-in-hole insertion task of Experiment 2 is considered, *i.e.*, with larger acceptability zone. The repeated trials organization is the same of Experiments 1 and 2. In Experiment 3, subjects perform an initial practice session gaining confidence with both individual and multiple feedback types on easy tasks. Then, they start the experiment at the lower index of difficulty receiving all types of feedback

concurrently (vibration, color, sound). Next, subjects perform multiple times the peg-in-hole task using color and sound feedbacks but no vibration. Finally, they perform the task using only either color or sound feedback. Half of the subjects (randomly selected) performed the final step of the experiment with color feedback, and half with sound feedback. Having completed this sequence at a given difficulty level, subjects then move to the next level up to the maximum level.

The purpose of Experiment 3 is to investigate the extent at which, removing one or two feedback types, the remaining feedback types provide adequate support to the user in terms of virtual fixturing.

6.3.3 Index of difficulty and Fitts' law

The previous description of the experimental protocol has emphasized the potential for a comparative analysis among the various feedback modalities, both individually and in multimodal combination. An additional area of investigation is how task execution time increases with respect to a task difficulty metric using each type of virtual fixture. In this section, background information is provided to perform such an investigation using Fitts' law. Fitts' law [104] has been used as a performance predictor in a number of human-computer interaction studies of target acquisition tasks (see [105] for a review). Tasks are described by means of two parameters: a distance or movement amplitude parameter and a target width parameter. The law reveals an intuitive tradeoff in human movement: the faster the human moves, the less precise his movements are, or vice versa: the more severe the constraints are, the slower he moves.

A peg-in-hole task in a PbD system can be described as well by the parameters required for Fitts' law analysis, since emphasis is on task programming rather than on detailed insertion control. Fitts' law analysis has been applied to peg-in-hole tasks assisted by virtual fixturing in [98] and [99]. The original paper by Fitts ([104], p. 387) also included a form of peg-in-hole task in his *Pin Transfer* experiment.

Fitts' law provides a set of general equations which in Experiments 1 and 2 is exploited to estimate from recorded data both the mean time per successful trial

(T_{mean}) and the mean total completion time (T_{tot}). Fitts' prediction states that the mean movement time for targeting tasks such as peg-in-hole depends linearly on the index of difficulty of the task (I_D):

$$T_{mean} = a + b \cdot I_D$$

where a and b are the constant real coefficients of the linear equation to be estimated from the experimental data. For the original experiments reported by Fitts, the index of difficulty is defined as:

$$I_D = \log_2 \left(\frac{A}{W} + c \right)$$

where A is the distance to reach the target, W is the size of the target and c is a constant. For the experiments reported in this chapter, the index of difficulty is defined as:

$$I_D = \log_2 \left(\frac{h}{2(R_a - r)} \right)$$

By decreasing the radius of the acceptability zone R_a in Figure 6.3 down to very small clearance, a set of tasks with increasing index of difficulty is achieved. Fitts pointed out that there is an analogy between his law and the Shannon-Hartley theorem or Shannon's capacity formula, regarding the capacity of an analog communication channel in the presence of white (gaussian) thermal noise:

$$C = B \log_2 \left(\frac{S}{N} + 1 \right)$$

where C is the channel capacity in bits/second, or the maximum rate at which bits can be transmitted such that the probability of bit error can be made arbitrarily small. B is the bandwidth of the channel in Hertz. N is the power of the noise, and S is the average transmitter power, or the power of the signal. Fitts' extended the notions of signal, noise, and channel capacity to the human motor system arguing that the motor system can be viewed as a transmitter of information, where the transmission of one symbol corresponds to the execution of one motor response. Under this analogy, if a

Table 6.1: Summary of subject characteristics for the experiments

	number of subjects	age range (years)	avg age	gender	lefthanded
Exp1	20	22 – 32	25.2	2F, 18M	2
Exp2	10	22 – 25	22.9	0F, 10M	0
Exp3	20	20 – 44	28.1	9F, 11M	0

Table 6.2: Experiment 1 - Completion Times

I_D	no fixture		vibration		color		sound	
	mean time	total time	mean time	total time	mean time	total time	mean time	total time
5.81	7.1 (2.6)	10.0 (3.9)	6.0 (1.5)	8.0 (2.3)	5.0 (1.4)	6.3 (2.5)	4.6 (1.3)	7.0 (3.6)
6.23	6.9 (2.0)	10.1 (5.9)	6.0 (1.9)	7.9 (3.7)	5.3 (1.7)	7.4 (4.5)	5.3 (2.0)	6.9 (2.7)
6.81	7.5 (2.5)	14.3 (6.4)	8.4 (3.6)	14.7 (11.3)	7.5 (2.0)	9.6 (3.5)	7.0 (3.3)	9.7 (4.6)
7.23	9.0 (3.8)	17.4 (10.7)	9.7 (3.4)	16.2 (6.8)	8.5 (2.6)	13.3 (8.9)	7.4 (2.4)	11.0 (4.2)
7.81	10.4 (3.2)	31.1 (21.2)	13.9 (5.1)	33.9 (23.1)	10.9 (3.5)	20.4 (10.0)	9.6 (3.6)	20.1 (13.0)

movement is repeated many times, the average time T_{mean} required to complete the movement, the amplitude A of the movement, and the variability W in the terminal location of the movement are analogous to $1/B$, S and N respectively.

Numerous studies have been conducted to extend the original formulation of the Fitts' law to more complex domains and applications. For example Accot et al. [113] modeled users' performances in trajectory-bases interactions and found robust regularities. In particular, the authors focused on "steering through tunnels" tasks. They found and verified a new expression for this kind of tasks.

6.3.4 Subject protocol

Twenty subjects participated to Experiment 1, ten subjects participated to Experiment 2, and twenty subjects participated to Experiment 3. Each subject participated to only one experiment. Table 6.1 summarizes the main characteristics of the subjects involved in the experiments. Population mainly consisted of students, faculty, and

Table 6.3: Experiment 2 - Completion Times

I_D	no fixture		vibration		color		sound	
	mean time	total time	mean time	total time	mean time	total time	mean time	total time
4.81	5.8 (2.8)	6.4 (2.3)	5.7 (2.5)	5.8 (2.5)	4.7 (2.0)	5.0 (2.7)	4.2 (1.7)	4.2 (1.7)
5.23	5.1 (1.4)	5.9 (2.4)	5.0 (1.8)	5.5 (1.4)	4.4 (1.1)	5.2 (1.9)	4.5 (1.5)	5.3 (1.5)
5.81	5.4 (2.0)	6.5 (2.3)	5.4 (2.0)	6.3 (3.0)	4.9 (2.3)	5.0 (2.2)	4.5 (1.9)	5.5 (2.2)
6.23	5.9 (2.7)	9.6 (7.4)	5.4 (2.6)	7.5 (4.0)	4.7 (2.1)	5.4 (3.3)	4.4 (1.7)	4.8 (1.9)
6.81	7.5 (3.2)	18.1 (10.2)	6.2 (2.5)	13.2 (9.1)	6.4 (3.5)	7.6 (3.6)	6.2 (3.2)	6.5 (3.1)

employees of the University of Parma, who read and signed a consent form. None of them was familiar with the glove and tracker interface and specific setup, although all of them were familiar with mouse-based computer interaction. It might be worthwhile noting that subjects in Experiments 1 and 2 were predominantly male, whereas in Experiment 3 a more varied population has been involved.

The CyberTouch was calibrated for each user at the beginning of the session. As stated before, each subject was also required to perform a training session before the test. During both training and testing, users were free to take as many pauses as they needed for resting, in case of too much fatigue on their arm.

6.4 Experimental results

6.4.1 Relative effectiveness of virtual fixtures

For experimental evaluation, the mean time per successful trial and the mean total completion time were computed by first averaging the successful trials of each case within a subject and then averaging across all the subjects. Tables 6.2 and 6.3 summarize the mean and total completion times for Experiments 1 and 2. Tables 6.4 and 6.5 report the error rate for the two experiments. Statistics include average values and standard deviation. Each task is labelled in Tables 6.2–6.5 by its corresponding index of difficulty I_D . One-way analysis of variance on the data has also been performed in Tables 6.2–6.5 using SPSS version 12.

The mean time T_{mean} for task completion can be written also as $T_{mean} = T_{trans} + T_{ins}$, where T_{trans} is the mean time spent for carrying the grasped object from its initial position to the container and T_{ins} is the mean time spent for the insertion. In the experiments described in this chapter, virtual fixtures only affect the second term; however, T_{trans} cannot be assumed as a constant, since different users performed the tasks with very different speeds and approaches, from extremely cautious to bold. Empirically, transportation times seemed to range from one second to several seconds.

It must be remarked that at the higher indices of difficulty, the peg-in-hole task is very difficult using "fish-tank" VR, considering the unavoidable inaccuracies associated with current VR glove and tracking devices. This difficulty is shown by the high percentage of errors in the two experiments (Tables 6.4 and 6.5).

Tables 6.2 and 6.3 show that when the task is very difficult ($I_D \geq 6.81$) virtual fixtures provide limited advantage in terms of mean completion time of successful trials. It must be considered that at the higher I_D levels, the task is perceived as very difficult by the users and the percentage of errors is very high. At $I_D = 7.81$, the highest level of difficulty investigated, four users were not able to complete the required successful trials using 15 attempts per trial. In this situation, completion times statistics are computed on data values which are likely to include several "lucky trials", *i.e.*, fast successful insertions sporadically occurring in a series of fast failed attempts.

The degree of assistance provided by virtual fixtures at high index of difficulty is thus rather shown by the reduction in the number of errors, which is remarkable in both experiments (Tables 6.4 and 6.5). For example, in Experiment 1 the mean error rate reduces from 61% with no fixture to 38% with the color fixture. This improvement in the mean is statistically significant with $p = 0.05$. Indeed, when users perceive that the task is very difficult but the system is providing some help by means of a virtual fixture, they tend to spend all the time that is required to achieve a precise insertion.

Another interpretation, possibly not conflicting but coexisting with the previous

Table 6.4: Experiment 1 - Error Rates

I_D	no fixture	vibration	color	sound
5.81	0.26 (0.26)	0.24 (0.22)	0.20 (0.16)	0.25 (0.23)
6.23	0.25 (0.25)	0.18 (0.23)	0.22 (0.20)	0.22 (0.21)
6.81	0.45 (0.23)	0.30 (0.27)	0.26 (0.23)	0.29 (0.27)
7.23	0.42 (0.28)	0.37 (0.26)	0.30 (0.19)	0.33 (0.20)
7.81	0.61 (0.20)	0.47 (0.26)	0.38 (0.24)	0.43 (0.24)

Table 6.5: Experiment 2 - Error Rates

I_D	no fixture	vibration	color	sound
4.81	0.13 (0.15)	0.07 (0.13)	0.04 (0.12)	0.00 (0.00)
5.23	0.09 (0.13)	0.05 (0.09)	0.22 (0.23)	0.16 (0.19)
5.81	0.15 (0.17)	0.18 (0.19)	0.11 (0.18)	0.16 (0.19)
6.23	0.34 (0.24)	0.22 (0.25)	0.08 (0.15)	0.08 (0.1)
6.81	0.47 (0.28)	0.44 (0.32)	0.15 (0.23)	0.08 (0.09)

one, stems from the empirical visual observation of the behavior of the subjects. In the range investigated, $I_D = 6.81$ seems to represent a sort of breaking condition for Experiment 1. Starting from this value, in fact, the error rate shows a sudden increase and users begin to pursue diverging strategies: some users try to carefully and slowly achieve the insertion goal relying on the virtual fixture if available, whereas others perform a quick attempt and then move to a new trial upon failure.

In terms of relative merit of the different types of virtual fixtures, Tables 6.2, 6.3 show that color and auditory feedbacks provide more benefit than the vibratory one in the range of difficulty indices investigated. Figure 6.4 shows Experiment 1 mean and total completion times for $I_D \leq 6.81$. The difference in the mean completion times per successful trial using different virtual fixtures or no virtual fixture can be up to a few seconds, although these differences are not always statistically significant.

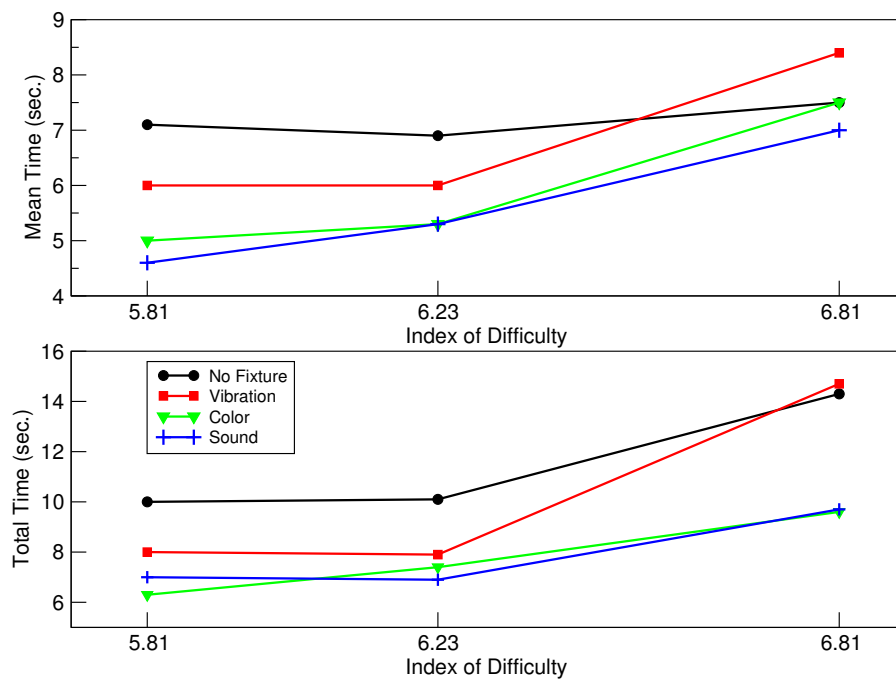


Figure 6.4: Completion time for Experiment 1, tasks with $I_D \leq 6.81$.

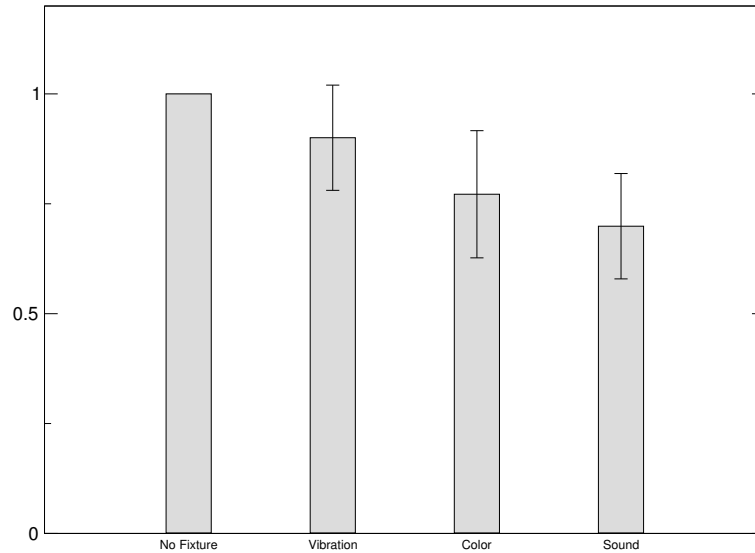


Figure 6.5: Relative improvement in the mean time for Experiment 1, task $I_D = 5.81$.

Figure 6.5 shows the relative improvements in the mean time for the task with $I_D = 5.81$ in Experiment 1. Considering relative improvements allows to factor out the effect of individual abilities and different approaches to task execution. The graph reports the average improvement in mean time across all the subjects for each feedback modality; error bars illustrate the 95% confidence interval. For tactile feedback, these data are in reasonable agreement with [103], where easier manipulation tasks (I_D in the range 3.0–4.0) were investigated.

Figure 6.6 shows the feedback returning the best average performance for the 20 subjects performing Experiment 1. Visual and auditory feedbacks usually allow the user to perform the task faster even when the difficulty of the task increases.

In general, vibration seems to provide disturbances during the insertion phase, in particular when the difficulty index of the task is very high and the user is required to keep the hand steady.

A final remark is related to the importance of visual information during the insertion phase. Figure 6.7 compares Experiments 1 and 2 mean completion time for

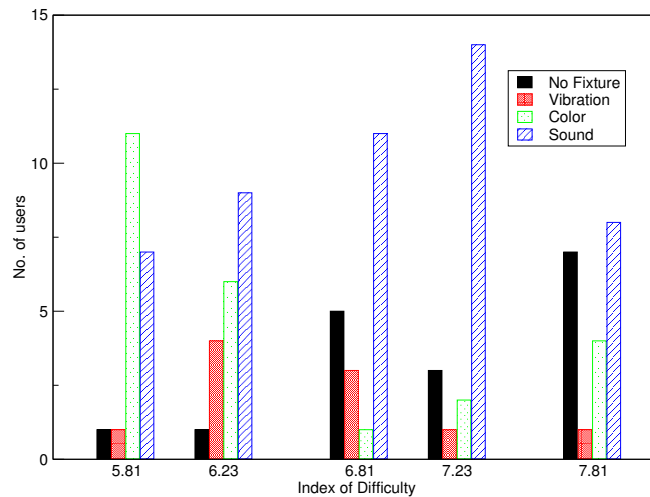


Figure 6.6: Feedback type returning the best mean completion time for the 20 subjects in Experiment 1.

the different feedback modalities. It can be recalled that at a given index of difficulty and feedback type, Experiment 2 only differs from Experiment 1 in the (smaller) size of the target hole, the size of the acceptability zone being of course the same. Yet, the users perform the task faster in Experiment 2, as shown in Figure 6.7. Total completion times and error rates are also better for Experiment 2. Even when the two experiments provide similar T_{mean} values, the smaller container visualized in Experiment 2 leads to a significant reduction in the number of errors.

A possible interpretation is that in Experiment 2 the container, being smaller than the acceptability zone, is actually acting as a view-finder. This result has intriguing implications in the design of virtual environment for optimal human-computer interaction in PbD systems.

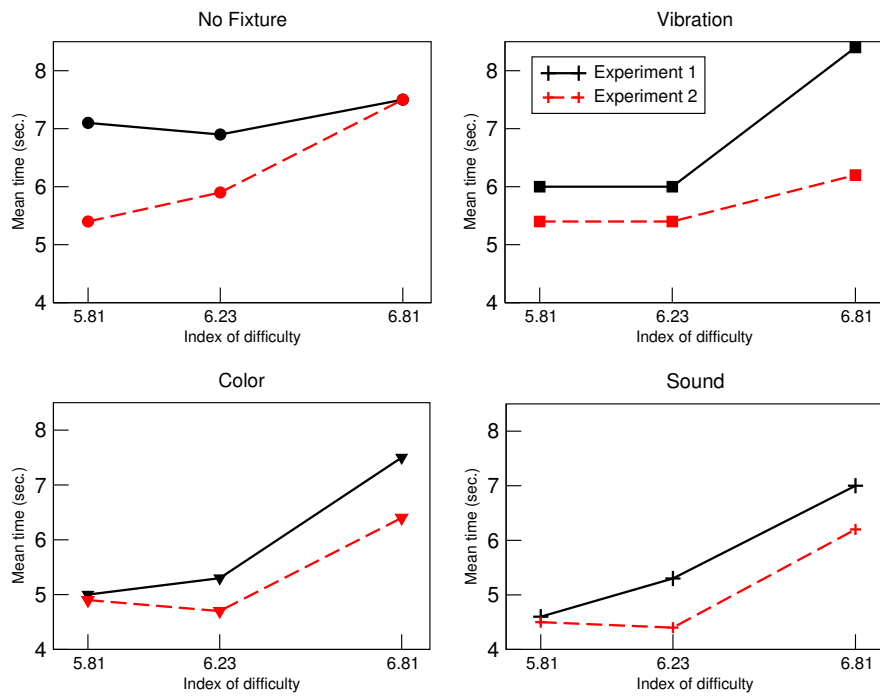


Figure 6.7: Comparison of mean completion times for Experiments 1 and 2.

Table 6.6: Experiment 3 - Mean Completion Times

I_D	sound + color + vibration	sound + color	color	sound
4.81	5.03 (2.14)	4.49 (1.74)	5.93 (3.09)	4.97 (3.16)
5.23	5.24 (2.86)	4.88 (2.63)	6.38 (4.01)	4.58 (1.87)
5.81	6.20 (3.80)	5.48 (2.38)	5.76 (2.54)	5.94 (4.05)
6.23	6.14 (3.29)	5.56 (2.65)	6.01 (3.38)	6.35 (3.69)

6.4.2 Combining multiple virtual fixtures

Tables 6.6 and 6.7 summarize mean completion time statistics, including standard deviation, and the corresponding error rates for Experiment 3. Mean completion time values are computed as described in the previous section, *i.e.* first averaging the successful trials of each case within a subject and then averaging across all subjects.

Figure 6.8 shows that mean task completion times are affected only to a limited extent by the specific set of virtual fixtures provided to the operator (error bars show the 95% confidence interval). At all levels of difficulty, the average task completion time decreases when the vibration feedback is removed, provided that both color and sound virtual fixtures are in place. This detrimental effect of vibration feedback confirms earlier remarks about the difficulties induced by vibration during the insertion phase. The combined sound and color virtual fixtures seem to provide the best user support at most indices of difficulty investigated. When only one feedback modality remains, sound tends to behave better than color at lower indices of difficulty. Due to the large standard deviation in completion times in Figure 6.8, these considerations should be regarded mainly as trends characterizing multimodal virtual fixturing.

Table 6.7: Experiment 3 - Error Rates

I_D	sound + color + vibration	sound + color	color	sound
4.81	0.12 (0.15)	0.17 (0.18)	0.06 (0.10)	0.04 (0.08)
5.23	0.17 (0.22)	0.17 (0.23)	0.06 (0.14)	0.13 (0.14)
5.81	0.26 (0.29)	0.27 (0.23)	0.23 (0.23)	0.41 (0.29)
6.23	0.35 (0.23)	0.30 (0.29)	0.28 (0.18)	0.29 (0.31)

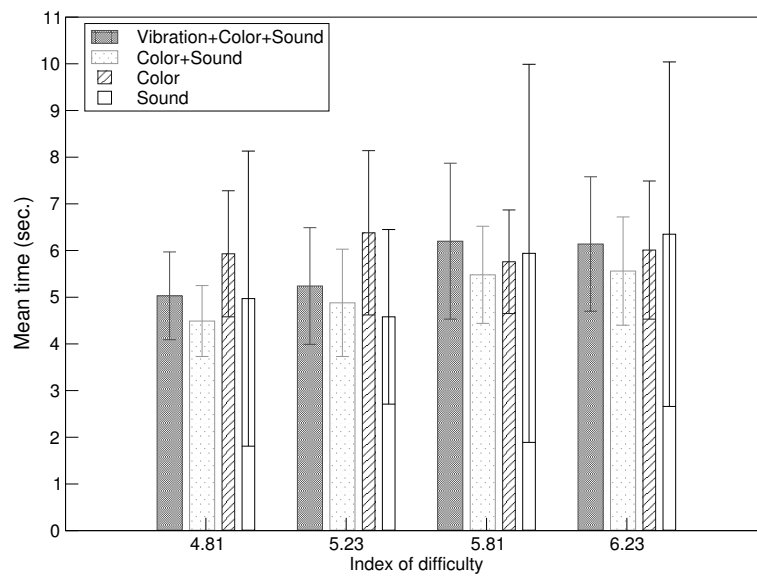


Figure 6.8: Mean completion time for Experiment 3.

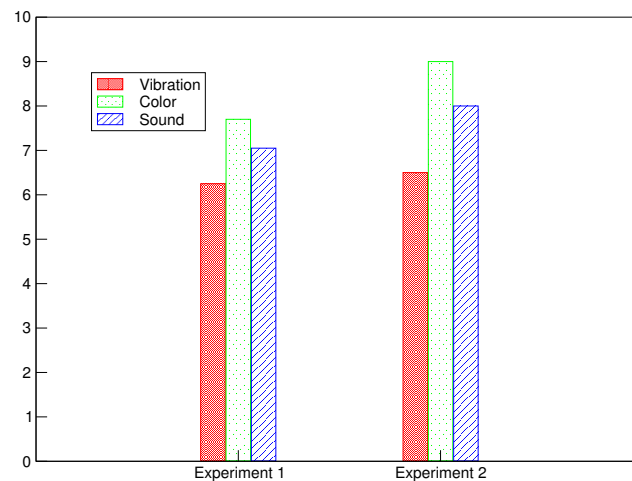


Figure 6.9: Users' marks for Experiments 1 and 2. The case of no fixture is given a reference mark of 1.

6.4.3 Users' remarks

After the experimental sessions of Experiments 1 and 2, each user was requested to express some comments about the task and the relative effectiveness of the various aids. Users filled in a questionnaire ranking the virtual fixtures with a mark ranging from -10 to 10 . They were instructed to consider 1 as a reference value for the case of absence of virtual fixtures, to suggest an appreciation for the moderate graphical support of the default virtual environment.

Average users' marks for the two experiments are shown in figure 6.9. For all the tasks the users preferred, on the average, the help of the color virtual fixture rather than the auditory or vibrotactile feedbacks. The score of the visual feedback was 7.45 in the first experiment and 9 in the second. Sound feedback, in spite of its slightly better experimental results (Figure 6.6), received around 1 point less. In general, users regarded any feedback as extremely useful, even when its usefulness

did not show very clearly in the mean completion time.

A frequent comment from the users was that when the task is very difficult the vibrotactile feedback interferes with a precise insertion of the peg. It is also well-known from physiological and psychophysical studies [97] that tactile perception is a dynamic effect that requires some activation time to trigger user's attention. Moreover, several users pointed out the importance of the grasping phase: if the cylinder is grasped with the thumb penetrating into its surface, some problems may arise while releasing it, because the peg tends to stay attached to the virtual thumb, thereby reducing the precision of the insertion task.

Finally, some users remarked that it is important to release the grasped object carefully to avoid unexpected errors. With the tight clearance investigated and the limitation in the accuracy provided by the glove interface, a sudden release of the cylinder immediately after the activation of the virtual fixture can easily move the user's hand outside the acceptability zone.

These remarks show that the performance limitations of current VR tools hinder their applicability in fine manipulation tasks, such as those required in advanced PbD interfaces for complex robotic assemblies.

6.5 Predicting human performance

Applying Fitts' law analysis to the data in Tables 6.2 and 6.3, it is possible to estimate how the mean completion time (T_{mean}) and total completion time (T_{tot}) increase with the index of difficulty for the different virtual fixtures..

Fitts' predictions for T_{mean} and T_{tot} become:

$$T_{mean} = a_m + b_m \cdot I_D$$

and

$$T_{tot} = a_t + b_t \cdot I_D$$

Table 6.8: Linear regression analysis for the mean completion time

experiment	fixture type	b_m	a_m	r^2	interval
1	no fixture	0.5	4.04	0.53	$I_D \leq 6.81$
1	vibration	2.5	-8.97	0.83	$I_D \leq 6.81$
1	color	2.5	-9.82	0.89	$I_D \leq 6.81$
1	sound	2.41	-9.55	0.98	$I_D \leq 6.81$
2	no fixture	0.87	0.90	0.54	
2	vibration	0.31	3.75	0.33	
2	color	0.77	0.60	0.58	
2	sound	0.82	0.04	0.62	

Table 6.9: Linear regression analysis for the total completion time

experiment	fixture type	b_t	a_t	r^2	interval
1	no fixture	4.44	-16.45	0.83	$I_D \leq 6.81$
1	vibration	6.99	-33.71	0.81	$I_D \leq 6.81$
1	color	3.38	-13.46	0.99	$I_D \leq 6.81$
1	sound	2.86	-10.09	0.80	$I_D \leq 6.81$
2	no fixture	5.54	-22.70	0.72	
2	vibration	3.43	-12.16	0.73	
2	color	1.07	-0.54	0.62	
2	sound	0.82	0.52	0.59	

where a_m , b_m , a_t , and b_t are computed from experimental data using linear regression.

Tables 6.8 and 6.9 summarize the results of the linear regression based on the data of Experiments 1 and 2. Tables include the coefficient of determination r^2 indicating how the model fits the data. The rightmost column reports the validity interval for each linear regression (omitted for regressions on the overall input data).

Unfortunately, the coefficient of determination r^2 is poor for several cases, thus the validity of Fitts' law prediction for these experiments remains questionable. It can be observed that the coefficient of determination is higher when only the easier cases of Experiment 1 are considered. This observation suggests that not all users

have learned to fully exploit the available virtual fixtures at higher index of difficulty tasks or when the fixturing concept of Experiment 2 is in place.

Even though Fitts' law has received broad empirical support in a number of related cases [98], the experiment assumed in this chapter as a reference benchmark for difficult PbD tasks does not fall in the same category. Apparently, task execution times are dominated by the different strategies adopted by the users to cope with the limitations of VR devices for high-accuracy manipulation tasks.

6.6 Discussion

In this chapter the effectiveness of various virtual fixtures for the PbD system have been investigated, focusing on peg-in-hole tasks. It has been found that all types of fixtures help in decreasing the error rate, but color and sound fixtures are more rapidly processed by the user than tactile fixtures and provide better performance in execution time and error rate. Displaying a target area smaller than the actual acceptability zone for a manipulation tasks has been found beneficial under any feedback modality.

When the manipulation task is very difficult, users resort to a number of different strategies, from very careful and slow to aggressive and relatively fast. From their written comments and gradings, in either cases they do attempt to exploit the feedback received, although within radically different strategies. The high variance in observed results does not allow a statistically robust prediction of average user execution time of pick-and-place tasks based on Fitts' law.

PbD of robotic assemblies in virtual environments seems to stretch the requirements posed on VR manipulation tracking devices beyond their current capabilities.

Conclusions

In this dissertation a novel robot programming by demonstration system has been presented. The system focuses on the recognition and reproduction of assembly tasks demonstrated by the user in a virtual environment. A fundamental aspect of the thesis is the application of virtual reality technologies to robot programming. The system exploits advanced devices such as a virtual reality glove and a tracker, and advanced computer graphics features such as scene graphs.

One important contribution of the thesis is the development of a library of robotic components which can be easily integrated to simulate complex mobile platforms. The system has been designed to offer to the user a friendly interface for task demonstration. The user interface allows the user to perform dextrous manipulation tasks in virtual reality by means of the simulation of a virtual human hand with grasping abilities. Novice users, without any prior experience about robot programming, can achieve good performances with a limited training period.

The system has four basic modules. The core of the system is a one-shot learning procedure for the segmentation and recognition of user's action at the task level. The second component of the system is an advanced algorithm for trajectory learning. The proposed algorithm is able to cluster multiple demonstrations of the same manipulation task into qualitatively similar paths. Moreover, the learning algorithm provides a stochastic approach for the selection and the approximation of the most consistent trajectories within each cluster. The proposed algorithm allows to filter out noise or unwanted movements which commonly happen while performing fine manipulation

tasks.

The third component of the system is a grasp recognition module which classifies the human hand postures in virtual reality taking advantage of virtual grasping and information about the contact points and normals, computed in the virtual reality environment. The proposed method opens interesting research directions since previous research in grasp recognition has never addressed the problem of grasp classification in a virtual environment. The grasp recognition module is coupled with a trajectory generator for pregrasp planning by demonstration and a grasp mapping procedure. Grasp mapping is required to translate the recognized human hand posture, which is acquired from the glove, to the robot hand available in the simulated setup. The grasp mapping algorithm exploits a database of predefined grasps taken from Cutkosky's taxonomy.

The final module of the system is responsible of enabling the use of virtual fixtures to assist the user while performing the demonstration of a task. Virtual fixtures are perceptual overlays designed to reduce the physical and psychological demands that arise during the execution of a task. Various types of virtual fixtures have been developed for the PbD, such as a visual, auditory and a vibrotactile feedback. The benefits of the virtual fixtures have been evaluated by novice users for a peg-in-hole task. In particular, it has been found that all types of fixtures help in decreasing the error rate.

While significant aspects of robot programming by demonstration have been addressed in the development of this thesis, there remain other important issues to be explored. Firstly, it may be desirable to integrate a reliable vision system for automatic recognition of the configuration of the objects in the workspace. Having position information about the objects and the obstacles in the workspace would enable an automatic procedure for building the virtual workspace for the demonstration of the tasks.

At present, the grasp planning system is able to simulate the approach trajectory of the robot arm to the object with the proper orientation of the robot gripper. It may be desirable to improve the grasping simulator to achieve full grasping ca-

pabilities by exploiting contact information between the robot and the environment. The grasping simulator can also be enhanced by introducing a dynamic engine in the simulation environment. The use of a dynamic engine can also improve the quality of the demonstration environment by adding more realistic physical constraints to the virtual world.

In conclusion, the work described in this dissertation outlines interesting directions for robot programming by demonstration and provides significant improvements for the application of virtual reality to human robot interaction.

Appendix A

Parametric Curves

NURBS (Non-Uniform Rational B-Spline) are parametric curves that are commonly used in computer graphics and in computer aided design. They are the generalization of non-rational B-splines which are based on rational Bézier curves. Rational Bézier curves are a generalization of Bézier curves. NURBS are useful for a number of reasons:

- they offer one common mathematical form for both standard analytical shapes (*e.g.* conics) and free form shapes;
- they reduce the memory consumption when storing shapes (compared to simpler methods);
- they can be evaluated reasonably fast by numerically stable and accurate algorithms;
- they are invariant under affine as well as perspective transformations.

A Bézier curve of degree n is defined by:

$$C(u) = \sum_{i=0}^n B_{i,n}(u)P_i \quad 0 \leq u \leq 1 \quad (\text{A.1})$$

The geometric coefficients P_i are called control points. The basis functions $B_{i,n}$ are the n -degree Bernstein polynomials given by:

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i} \quad (\text{A.2})$$

The Bézier curves can not represent conic curves. A conic curve (such as a circle) can be represented using a rational function which is defined as the ratio of two polynomials. From this observation, the rational Bézier curve is defined as

$$C(u) = \frac{\sum_{i=0}^n w_i P_i B_{i,n}(u)}{\sum_{i=0}^n w_i B_{i,n}(u)} \quad 0 \leq u \leq 1 \quad (\text{A.3})$$

where the w_i are scalars called the weights.

The equation of the B-spline is memory efficient and also allows for the local control of the curve; *i.e.* the basis functions are not defined over the whole interval but they are constrained to a limited number of subintervals. A B-spline is defined as follows:

$$C(u) = \sum_{i=0}^n N_{i,p} P_i \quad a \leq u \leq b \quad (\text{A.4})$$

where P_i are the control points and $N_{i,p}$ are the p -degree degree B-spline basis functions. The B-spline has breakpoints which are named knots. A sequence of these knots is named the knot vector and it is defined as $U = u_0 \dots, u_t$ which is a non-decreasing sequences of real numbers, *i.e.*, $u_i \leq u_{i+1}$ for $i = 0, \dots, t$. The B-spline basis function of p -degree is defined using the recurrence formula as

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u \leq u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \quad (\text{A.5})$$

The above equation can result in a zero by zero division, that quotient is defined to be zero. As mentioned above, only rational functions can represent a conic curves therefore it is possible to generalize the B-spline curve to obtain a rational representation. This generalization is named Non Uniform Rational B-Spline (NURBS) and it is defined as

$$C(u) = \frac{\sum_{i=0}^n w_i P_i N_{i,p}(u)}{\sum_{i=0}^n w_i N_{i,p}(u)} \quad a \leq u \leq b \quad (\text{A.6})$$

NURBS curves have several important properties.

1. NURBS curve $C(u)$ is a piecewise curve with each component a degree p rational curve. Actually, each component is a rational Bézier curve.
2. Equality $t = n + p + 1$ must be satisfied.
3. If the first $p + 1$ knots and the last $p + 1$ knots are equal to the left end and right end of the domain, the curve is said clamped. A clamped NURBS curve $C(u)$ passes through the two end control points P_0 and P_n .
4. Strong Convex Hull Property: the NURBS curve is contained in the convex hull of its control points.
5. Local Modification Scheme: changing the position of control point P_i only affects the curve $C(u)$ on interval $[u_i, u_{i+p+1})$.
6. $C(u)$ is C^{p-k} continuous at a knot of multiplicity k .
7. Variation Diminishing Property: if the curve is contained in a plane (resp., space), this means no straight line (resp., plane) intersects a NURBS curve more times than it intersects the curve's control polyline.
8. If all weights are equal, a NURBS curve becomes a B-spline curve.

9. Projective Invariance: if a projective transformation is applied to a NURBS curve, the result can be constructed from the projective images of its control points.

A.1 OpenGL NURBS Interface

OpenGL provides a NURBS interface which hides the math required to create such curves behind a few high level functions. The NURBS interface requires a NURBS context object that is passed into each call. To create a context object the following function must be called

```
nurbs = gluNewNurbsRenderer();
```

Every call to this function returns a pointer to a `GLUnurbsObj`, that must be used with each of the other NURBS interface functions. To set the properties of a NURBS, the `gluNurbsProperty()` function must be called. The properties of a NURBS include sampling tolerance and display mode. All NURBS drawing functions have to be surrounded by a `gluBeginCurve()`, `gluEndCurve()` pair. The only parameter that these functions take is the pointer to the NURB object returned above. The curve is actually drawn with the function `gluNurbsCurve()` as follows

```
gluBeginCurve(nurb);
gluNurbsCurve( GLUnurbsObj *nurb,
               GLint KnotCount, GLfloat *Knot,
               GLint Stride, GLfloat *ctlArray, GLint Order,
               GLenum type );
gluEndCurve(nurb);
```

`gluNurbsCurve()` is the function that actually evaluates the NURBS. The first argument is the NURBS context object. The next two are the number of knots and a pointer to the knot vector. Then comes the offset (as a number of single precision floating-point values) between successive curve control points and a pointer to the

control points. The next argument is the order of the curve. The order is equivalent to the number of knots minus the number of control points. The last parameter specifies the type of the curve.

A.2 Nurbs++ library

Nurbs++ [56] is a C++ library created for the manipulation of NURBS curves and surfaces. The core of the package is composed of a NURBS library, a matrix library, an image library and a numerical library. The matrix library offers the basic mathematical operations used by all the other libraries. The image library offers image manipulation routines, the numerical library offers interpolation and approximation algorithms and statistical functions. The NURBS library contains the NURBS related classes. The library offers many function to manipulate NURBS objects and also to display them using OpenGL, VRML or Post-Script. In addition, it is templated so it's easy to have either a float and a double precision library.

Appendix B

Learning of Hidden Markov Models

A discrete HMM is defined by a set of states S ; a transition probability matrix, $A = \{a_{ij}\}$, where

$$a_{ij} = P(q_{t+1} = S_j | q_t = S_i), \quad 1 \leq i, j \leq N \quad (\text{B.1})$$

is the transition probability from state i to state j ; and the observation symbol probability distribution in state j , $B = \{b_j(k)\}$, where

$$b_j(k) = P(v_k \text{ at } t | q_t = S_j), \quad 1 \leq j \leq N, 1 \leq k \leq M \quad (\text{B.2})$$

The individual symbols are denoted as $V = \{v_1, v_2, \dots, v_M\}$. A HMM can be expressed compactly as $\lambda = (A, B, \pi)$, where $\pi = \{\pi_i\}$ is the initial state distribution defined as

$$\pi_i = P(q_1 = S_i), \quad 1 \leq i \leq N \quad (\text{B.3})$$

Given the previous form of HMM there are three basic problems of interest.

1. Given the observation sequence $O = O_1 O_2 \dots O_T$ and the model λ compute

the probability $P(O|\lambda)$ of the observation sequence given the model. This is solved by the forward algorithm.

2. Given the observation sequence O and the model λ choose a state sequence $Q = q_1 q_2 \dots q_T$ which maximize $P(Q|O, \lambda)$. A formal procedure to find this "best" state sequence is the Viterbi algorithm.
3. Train the model parameters $\lambda = (A, B, \pi)$ to maximize $P(O|\lambda)$. This is achieved by the Baum-Welch algorithm.

B.1 Forward-Backward algorithm

Let the forward probability $\alpha_t(i)$ for some model λ be defined as

$$\alpha_t(i) = P(O_1 O_2 \dots O_t, q_t = S_i | \lambda) \quad (\text{B.4})$$

i.e., the probability of the partial observation sequence, $O_1 O_2 \dots O_t$ (until time t) and state S_j at time t , given the model λ . The forward algorithm computes $\alpha_t(i)$ inductively, as follows:

1. Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \quad (\text{B.5})$$

2. Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(O_{t+1}), \quad 1 \leq t \leq T-1, 1 \leq j \leq N \quad (\text{B.6})$$

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i) \quad (\text{B.7})$$

Similarly, a backward variable $\beta_t(i)$ (the backward procedure is used in the Baum-Welch algorithm) can be defined as follows:

$$\beta_t(i) = P(O_{t+1}O_{t+2} \dots O_T | q_t = S_i, \lambda) \quad (\text{B.8})$$

i.e., the probability of the partial observation sequence from $t + 1$ to the end, given the state S_i at time t and the model λ . Again the backward algorithm computes $\beta_t(i)$ inductively, as follows:

1. Initialization:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \quad (\text{B.9})$$

2. Induction:

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \quad t = T - 1, \dots, 1, 1 \leq i \leq N \quad (\text{B.10})$$

B.2 Viterbi algorithm

The Viterbi algorithm finds the single best state sequence $Q = q_1 q_2 \dots q_T$ maximizing $P(Q|O, \lambda)$. The highest probability of a sequence of states at time t ending in state S_j is defined as

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} P(q_1 q_2 \dots q_t = i, O_1 O_2 \dots O_t | \lambda) \quad (\text{B.11})$$

by induction we have

$$\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij} \right] \cdot b_j(O_{t+1}) \quad (\text{B.12})$$

to retrieve the state sequence the argument which maximizes B.12 must be tracked for each t and j . The complete procedure is stated as follows:

1. Initialization:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \quad (\text{B.13})$$

$$\psi_1(i) = 0 \quad (\text{B.14})$$

2. Recursion:

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] \cdot b_j(O_t) \quad 2 \leq t \leq T, 1 \leq j \leq N \quad (\text{B.15})$$

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] \quad 2 \leq t \leq T, 1 \leq j \leq N \quad (\text{B.16})$$

3. Termination:

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)] \quad (\text{B.17})$$

$$q_T^* = \arg \max_{1 \leq i \leq N} [\delta_T(i)] \quad (\text{B.18})$$

4. Path (state sequence) backtracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \dots, 1 \quad (\text{B.19})$$

B.3 Baum-Welch algorithm

The Baum-Welch algorithm is a method to adjust the model parameters (A, B, π) to maximize the probability of the observation sequence given the model. There is no known way to analytically solve for the model which maximizes the probability of the observation sequence. However, it is possible to choose λ such that $P(O|\lambda)$ is locally maximized using an iterative procedure. Let $\xi_t(i, j)$ be the probability of being in state S_j at time t , and in state S_i at time $t+1$, given the model and the observation sequence, *i.e.*

$$\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \quad (\text{B.20})$$

and from the definitions of the forward and backward variables, $\xi_t(i, j)$ can be rewritten as

$$\begin{aligned}
 \xi_t(i, j) &= \frac{P(q_t = S_i, q_{t+1} = S_j, O|\lambda)}{P(O|\lambda)} \\
 &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} \\
 &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \tag{B.21}
 \end{aligned}$$

Let $\gamma_t(i)$ be the probability of being in state S_i at time t , given the observation sequence O and the model λ , *i.e.*

$$\gamma_t(i) = P(q_t = S_i|O, \lambda). \tag{B.22}$$

Equation B.22 can be expressed simply in terms of the forward-backward variables, *i.e.*

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^N \alpha_t(i)\beta_t(i)}. \tag{B.23}$$

The two probabilities $\gamma_t(i)$ and $\xi_t(i, j)$ can be related by summing over j , giving

$$\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j). \tag{B.24}$$

If we sum $\gamma_t(i)$ over the time index t , we get a quantity which can be interpreted as the expected number of times that state S_i is visited, or equivalently, the expected number of transitions made from state S_i (excluding time slot $t = T$ from the summation). Similarly, summation of $\xi_t(i, j)$ over t can be interpreted as the number of transitions from state S_i to state S_j . From the above formulas it is possible to give a method for

the reestimation of the parameters of an HMM, *i.e.*

$$\bar{\pi}_i = \gamma_1(i) \quad (\text{B.25})$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad (\text{B.26})$$

$$\bar{b}_j(k) = \frac{\sum_{\substack{t=1 \\ s.t. O_t=v_k}}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)} \quad (\text{B.27})$$

Equations B.25, B.26 and B.27 provide a reestimated model $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$ which is more likely than model λ , *i.e.* $P(O|\bar{\lambda}) \geq P(O|\lambda)$. The computation of $\bar{\lambda}$ can be iterated to improve the probability of O being observed from the model until some limiting point is reached. The optimization algorithm leads to local maxima only.

B.4 Multiple observation sequences

Equations B.26 and B.27 can be extended in the case of multiple observation sequences by adding together the individual frequencies of occurrence for each sequence. Denoting the set of K observation sequences as

$$\mathbf{O} = [\mathbf{O}^{(1)}, \mathbf{O}^{(2)}, \dots, \mathbf{O}^{(K)}] \quad (\text{B.28})$$

where $\mathbf{O}^k = [O_1^{(k)}, O_2^{(k)}, \dots, O_{T_k}^{(k)}]$ is the k th observation sequence. Assuming that each observation sequence is independent of every other observation sequence, the

goal is to adjust the parameters of the model λ to maximize

$$P(\mathcal{O}|\lambda) = \prod_{k=1}^K P(\mathcal{O}^{(k)}|\lambda) = \prod_{k=1}^K P_k \quad (\text{B.29})$$

The modified reestimation formulas for \bar{a}_{ij} and $\bar{b}_j(l)$ are

$$\bar{a}_{ij} = \frac{\sum_{k=1}^K \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) a_{ij} b_j(\mathcal{O}_{t+1}^{(k)}) \beta_{t+1}^k(j)}{\sum_{k=1}^K \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \quad (\text{B.30})$$

and

$$\bar{b}_j(l) = \frac{\sum_{k=1}^K \frac{1}{P_k} \sum_{\substack{t=1 \\ s.t. \mathcal{O}_t = v_l}}^{T_k-1} \alpha_t^k(j) \beta_t^k(j)}{\sum_{k=1}^K \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(j) \beta_t^k(j)} \quad (\text{B.31})$$

If a left-right model is adopted (Bakis model), the state proceeds from state 1 at $t = 1$ to state N at $t = T$, thus π_i must not be reestimated.

B.5 Continuous observation densities in HMMs

When the observations are continuous signals or vectors then continuous observation densities must be introduced. The most general representation is a finite mixture of the form

$$b_j(\mathcal{O}) = \sum_{m=1}^M c_{jm} \mathfrak{N}(\mathcal{O}, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}), \quad 1 \leq j \leq N \quad (\text{B.32})$$

where \mathcal{O} is the vector being modeled, c_{jm} is the mixture coefficient for the m th mixture in state j and \mathfrak{N} is a gaussian density function with mean vector $\boldsymbol{\mu}_{jm}$ and

covariance matrix \mathbf{U}_{jm} for the m th mixture component in state j . \mathfrak{N} has the following form:

$$\mathfrak{N}[\mathbf{O}, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}] = \frac{1}{(2\pi)^{n/2} |\mathbf{U}_{jm}|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{O} - \boldsymbol{\mu}_{jm})^t \mathbf{U}_{jm}^{-1} (\mathbf{O} - \boldsymbol{\mu}_{jm}) \right] \quad (\text{B.33})$$

where $|\mathbf{U}_{jm}|$ is the determinant of the covariance matrix \mathbf{U}_{jm} . The mixture gains c_{jm} satisfy the stochastic constrain

$$\sum_{m=1}^M c_{jm} = 1, \quad 1 \leq j \leq N \quad (\text{B.34})$$

where $c_{jm} \geq 0$ so that the probability density function is properly normalized, *i.e.*

$$\int_{-\infty}^{\infty} b_j(\mathbf{x}) d\mathbf{x} = 1, \quad 1 \leq j \leq N \quad (\text{B.35})$$

It can be shown that the reestimation formulas for the coefficient of the mixture density, *i.e.* c_{jm} , $\boldsymbol{\mu}_{jm}$ and \mathbf{U}_{jm} are of the form

$$\bar{c}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k)}{T \sum_{t=1}^T \sum_{k=1}^M \gamma_t(j, k)} \quad (\text{B.36})$$

$$\bar{\boldsymbol{\mu}}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) \cdot \mathbf{O}_t}{\sum_{t=1}^T \gamma_t(j, k)} \quad (\text{B.37})$$

$$\bar{U}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) \cdot (\mathbf{O}_t - \boldsymbol{\mu}_{jk})(\mathbf{O}_t - \boldsymbol{\mu}_{jk})^t}{\sum_{t=1}^T \gamma_t(j, k)} \quad (\text{B.38})$$

where $\gamma_t(j, k)$ is the probability of being in state j at time t with the k th mixture component accounting for \mathbf{O}_t , *i.e.*

$$\gamma_t(j, k) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} \cdot \frac{c_{jk}\mathfrak{N}(\mathbf{O}_t, \boldsymbol{\mu}_{jk}, \mathbf{U}_{jk})}{\sum_{m=1}^M c_{jm}\mathfrak{N}(\mathbf{O}_t, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm})} \quad (\text{B.39})$$

The reestimation formula for a_{ij} is identical for the one used for discrete observation densities. In the case of multiple observation sequences the reestimation equations can be found by applying the same normalization procedure, described in section B.4, for the discrete models.

Bibliography

- [1] G. Hirzinger, B. Brunner, J. Dietrich, and J. Heindl. Sensor-Based Space Robotics-ROTEX and its Telerobotic Features. *IEEE Trans. on Industrial Electronics*, 9(5), 1993.
- [2] S. Jia, Y. Hada, G. Ye, and K. Takase. Distributed Telecare Robotic Systems Using CORBA as a Communication Architecture. In *Int'l Conf. on Robotics and Automation*, Washington, DC, May 2002.
- [3] C. Preusche, J. Hoogen, D. Reintsema, G. Schmidt, and G. Hirzinger. Flexible multimodal telepresent assembly using a generic interconnection framework. In *Int'l Conf. on Robotics and Automation*, Washington, DC, May 2002.
- [4] K. Ikeuchi and T. Suehiro. Towards an assembly plan from observation, Part I: Task recognition with polyhedral objects. *IEEE Trans. Robot. Automat.*, 10(3):368–385, 1994.
- [5] R. Zöllner, O. Rogalla, R. Dillmann, and M. Zöllner. Understanding Users Intention: Programming Fine Manipulation Tasks by Demonstration. *IEEE/RSJ Int'l Conference on Intelligent Robots and Systems, IROS 2002*, September 2002.
- [6] A. Cypher, editor. *Watch What I do: Programming by Demonstration*. The MIT Press, 1993.

-
- [7] R. A. Brooks. Intelligence without Reason. In *12th Int'l Joint Conf. on Artificial Intelligence*, 1991.
- [8] E. Drumwright and M. Matarić. Generating and recognizing free-space movements in humanoid robots. In *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 1672 – 1678, Las Vegas, Nevada, October 2003.
- [9] E. Drumwright, O. Jenkins, and M. Matarić. Exemplar-Based Primitives for Humanoid Movement Classification and Control. In *Int'l Conf. on Robotics and Automation*, New Orleans, LA, April 2004.
- [10] H. Tominaga and K. Ikeuchi. Acquiring Manipulation Skills through Observation. *IEEE Int'l Conf. on Multisensor Fusion and Integration for Intelligent Systems*, pages 7–12, Aug. 1999.
- [11] N. Delson and H. West. Robot Programming by Human Demonstration: The Use of Human Inconsistency in Improving 3D Robot Trajectories. In *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 1248 – 1255, September 1994.
- [12] P. Viviani and C.A. Terzuolo. Trajectory determines movement dynamics. *Neuroscience*, 7:431–437, 1982.
- [13] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by Watching: Extracting Reusable Task Knowledge from Visual Observation of Human Performance. *IEEE Trans. on Robotics and Automation*, 10(6), 1994.
- [14] K. Ogawara, J. Takamatsu, H. Kimura, and K. Ikeuchi. Extraction of Essential Interactions through Multiple Observations of Human Demonstrations. *IEEE Trans. on Industrial Electronics*, 50(4), 2003.
- [15] R. Dillmann, O. Rogalla, M. Ehrenmann, R. Zöllner, and M. Bordegoni. Learning Robot Behaviour and Skills Based on Human Demonstration and Advice: The Machine Learning Paradigm. In *9th Int'l Symp. of Robotics Research*, 1999.

- [16] C. P. Sayers and R. P. Paul. An operator interface for teleprogramming employing synthetic fixtures. *Presence*, 3:309–320, Fall 1994.
- [17] T. Takahashi and T. Sakai. Teaching robot's movement in virtual reality. *IEEE/RSJ Int. Workshop on Intelligent robots and systems, IROS 1991*, nov 1991.
- [18] H. Ogata and T. Takahashi. Robotic Assembly Operation Teaching in a Virtual Environment. *IEEE Trans. Robot. Automat.*, 10(3):391–399, June 1994.
- [19] E. Lloyd, J. S. Beis, D. K. Pai, and D. G. Lowe. Programming Contact Tasks Using a Reality-Based Virtual Environment Integrated with Vision. *IEEE Trans. Robot. Automat.*, 15(3):423–434, June 1999.
- [20] H. Kawasaki, K. Kanji, and G. Parker. Teaching for multi-fingered robots based on motion intention in virtual reality. In *IEEE Int conference on industrial electronics, control and instrumentation (IECON)*, Nagoya, Japan, Oct. 2000.
- [21] K. Ogawara, S. Iba, T. Tanuki, H. Kimura, and K. Ikeuchi. Acquiring hand-action models by attention point analysis. In *Int'l Conf. on Robotics and Automation*, pages 465–470, May 2001.
- [22] K. Ogawara, S. Iba, T. Tanuki, H. Kimura, and K. Ikeuchi. Recognition of human task by attention point analysis. In *IEEE/RSJ Intl Conference on Intelligent Robots and Systems (IROS)*, 2000.
- [23] M. Ehrenmann, P. Steinhaus, and R. Dillmann. A multisensor system for observation of user actions in programming by demonstration. In *Multisensor Fusion and Integration for Intelligent Systems*, Taipei, Taiwan, Aug. 1999.
- [24] M. Ehrenmann, R. Zöllner, O. Rogalla, and R. Dillmann. Programming Service Tasks in Household Environments by Human Demonstration. In *11th IEEE International Workshop on Robot and Human Interactive Communication*, 2002.

- [25] R. D. Zöllner and R. Dillmann. Using multiple probabilistic hypothesis for programming one and two hand manipulation by demonstration. In *Int'l Conf. on Intelligent Robots and Systems*, pages 2926 – 2931, Las Vegas, NV, October 2003.
- [26] G.E. Hovland, P. Sikka, and B.J. McCarragher. Skill acquisition from human demonstration using a Hidden Markov Model. In *IEEE Int'l Conf. on Robotics and Automation*, pages 2706 – 2711, Minneapolis, MN, April 1996.
- [27] R. Amit and M. Matarić. Learning movement sequences from demonstration. In *Int'l Conf. on Development and Learning*, Cambridge, Massachusetts, June 2002.
- [28] A. Billard, Y. Epars, S. Calinon, G. Cheng, S. Schaal, and G. Cheng. Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47(2-3):69–77, 2004.
- [29] S. Schaal, S. Vijayakumar, A. D'Souza, A. Ijspeert, and J. Nakanishi. Real-time statistical learning for robotics and human augmentation. In *Proc. Tenth International Symposium on Robotics Research (ISRR)*, Victoria, Australia, 2001.
- [30] G. Burdea and P. Coiffet. *Virtual Reality Technology*. John Wiley & Sons Inc, New York, USA, 2003.
- [31] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *Proc. ACM Computer Graphics, SIGGRAPH Conf.*, pages 135–142, Anaheim, CA, 1993.
- [32] C. Ware, K. Arthur, and K. S. Booth. Fish tank virtual reality. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 37 – 42, 1993.

- [33] C. Ware and R. Balakrishnan. Reaching for objects in VR displays: lag and frame rate. *ACM Transactions on Computer-Human Interaction*, 1(4):331–356, 1994.
- [34] R. M. Satava and S. B. Jones. Medical applications of virtual environments. In Kay M. Stanney, editor, *Handbook of Virtual Environments; Design, Implementation, and Applications*, pages 937–957. Erlbaum, Mahwah NJ, 2002.
- [35] A.C. Boud, D. J. Haniff, C. Baber, and S.J. Steiner. Virtual reality and augmented reality as a training tool for assembly tasks. In *Proceedings of IEEE International Conference on Information Visualization*, pages 32–36, 1999.
- [36] P. Wellner, W. Mackay, and R. Gold. Computer augmented environments: Back to the real world. *Commun. ACM*, 36(7):24–26, July 1993.
- [37] R. T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, August 1997.
- [38] E. Freund and J. Rossmann. Projective Virtual Reality: Bridging the Gap between Virtual Reality and Robotics. *IEEE Trans. on Robotics and Automation*, 15(3):411–422, June 1999.
- [39] <http://www.immersion.com/>.
- [40] <http://www.polhemus.com/>.
- [41] G. Drew Kessler and L. Hodges. Evaluation of the cyberglove as a whole hand input device. *ACM Transactions on Computer-Human Interaction*, 2(4):263–283, December 1995.
- [42] Virtualhand suite v2.0 - programmer's guide. Technical report, 2175 Park Boulevard Palo Alto, California 94306 USA.
- [43] <http://www.robotics.utexas.edu>.

- [44] S. Bottazzi, S. Caselli, M. Reggiani, and M. Amoretti. A Software Framework based on Real-Time CORBA for Telerobotic Systems. In *International Conference on Intelligent Robots and System (IROS)*, volume 3, pages 3011 – 3017, October 2002.
- [45] M. Henning and S. Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, 1999.
- [46] <http://www.omg.org/>.
- [47] D.C. Schmidt, D.L. Levine, and S. Mungee. The Design of the TAO Real-Time Object Request Broker. *Computer Communications*, 21(4):294–324, 1998.
- [48] J. Lloyd and V. Hayward. *Multi-RCCL User’s Guide*, 1992.
- [49] I.R. Belousov, R. Chellali, and G.J. Clapworthy. Virtual reality tools for internet robotics. *IEEE International Conference on Robotics and Automation*, 2:1878–1883, 2001.
- [50] R.I. Abenavoli, A. Amoroso, H. Kormanski, and K. Rudzinska. Mobile platform simulation and control via virtual reality. *IEEE Proc. on Intelligent Transportation Systems*, 2:1589– 1594, 12-15 Oct. 2003.
- [51] M. Gutierrez, R. Ott, D. Thalmann, and F. Vexo. Mediators: virtual haptic interfaces for tele-operated robots. *13th IEEE International Workshop on Robot and Human Interactive Communication*, pages 515 – 520, 20-22 Sept. 2004.
- [52] B.R Duffy, G.M.P. O’Hare, R.P.S O’Donoghue, C.F.B. Rooney, and R.W. Collier. Reality & virtual reality in mobile robotics. *1st International Workshop on Managing Interactions in Smart Environments, MANSE*, December 1999.
- [53] J. Miura, K. Iwase, and Y. Shirai. Interactive teaching of a mobile robot. *IEEE International Conference on Robotics and Automation*, pages 3389–3394, 2005.

- [54] S. Iba, C. Paredis, and P. Khosla. Interactive multimodal robot programming. *International Journal of Robotics Research*, 24(1):83–104, January 2005.
- [55] <http://www.exactdynamics.nl/>.
- [56] <http://libnurbs.sourceforge.net/>.
- [57] A. Ude and R. Dillmann. Trajectory reconstruction from stereo image sequences for teaching robot paths. In *Int'l Symp. Industrial Robots*, pages 407 – 414, Tokyo, Japan, Nov. 1993.
- [58] A. Billard and S. Schaal. Robust learning of arm trajectories through human demonstration. In *Int'l Conf. on Intelligent Robots and Systems*, pages 734 – 739, Maui, Hawaii, USA, Oct-Nov 2001.
- [59] M. Riley, A. Ude, and C.G. Atkeson. Methods for motion generation and interaction with a humanoid robot: Case studies of dancing and catching. In *AAAI and CMU Workshop on Interactive Robotics and Entertainment*, Pittsburgh, Pennsylvania, April 2000.
- [60] A. Ude, C.G. Atkeson, and M. Riley. Planning of joint trajectories for humanoid robots using B-spline wavelets. In *Int'l Conf. on Robotics and Automation*, pages 2223 – 2228, San Francisco, CA, April 2000.
- [61] M. Riley, A. Ude, K. Wade, and C.G. Atkeson. Enabling real-time full-body imitation: A natural way of transferring human movement to humanoids. In *Int'l Conf. on Robotics and Automation*, pages 2368 – 2374, Taipei, Taiwan, September 2003.
- [62] C.H. Lee. A Phase Space Spline Smoother for Fitting Trajectories. *IEEE Trans. on Systems, Man, and Cybernetics-Part B: Cybernetics*, 34(1):346–356, February 2004.

- [63] J. Yang, Y. Xu, and C.S. Chen. Hidden Markov Model Approach to Skill Learning and Its Application to Telerobotics. *IEEE Trans. on Robotics and Automation*, 10(5):621–631, October 1994.
- [64] J. Yang, Y. Xu, and C.S. Chen. Human action learning via Hidden Markov Model. *IEEE Trans. on Systems, Man and Cybernetics, Part A*, 27(1):34 – 44, Jan. 1997.
- [65] S.K. Tso and K.P. Liu. Demonstrated trajectory selection by Hidden Markov Model. In *Int'l Conf. on Robotics and Automation*, pages 2713 – 2718, Albuquerque, New Mexico, April 1997.
- [66] Wentao Yu, R. Dubey, and N. Pernalet. Robotic therapy for persons with disabilities using Hidden Markov Model based skill learning. In *IEEE Int'l Conf. on Robotics and Automation*, pages 2074–2079, New Orleans, LA, April 2004.
- [67] P. Pook and D.H. Ballard. Recognizing teleoperated manipulations. In *Int'l Conf. on Robotics and Automation*, pages 578–585, Atlanta, GA, May 1993.
- [68] W. Bluethmann, R. Ambrose, M. Diftler, E. Huber, A. Fagg, M. Rosenstein, R. Platt, R. Grupen, C. Breazeal, A. Brooks, A. Lockerd, R. A. Peters II, O. C. Jenkins, M. Matarić, and M. Bugajska. Building an autonomous humanoid tool user. In *IEEE International Conference on Humanoid Robots*, Los Angeles, CA, 2004.
- [69] J. Lieberman and C. Breazeal. Improvements on action parsing and action interpolation for learning through demonstration. In *IEEE International Conference on Humanoid Robots*, Los Angeles, CA, 2004.
- [70] T. Inamura, I. Toshima, and H. Tanie. Embodied symbol emergence based on mimesis theory. *International Journal of Robotics Research*, 23(4-5):363–377, 2004.

- [71] S. Calinon and A. Billard. Stochastic gesture production and recognition model for a humanoid robot. In *IEEE/RSJ Intl Conference on Intelligent Robots and Systems (IROS)*, Sendai, Japan, 2004.
- [72] K. Ogawara, J. Takamatsu, H. Kimura, and K. Ikeuchi. Modeling Manipulation Interactions by Hidden Markov Models. In *Int'l Conf. on Intelligent Robots and Systems*, pages 1096 – 1101, Lausanne, Switzerland, October 2002.
- [73] L.R Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.
- [74] L. Piegl. On NURBS: A Survey. *IEEE Computer Graphics and Applications*, 11(1):55–71, Jan 1991.
- [75] L. Piegl and W. Tiller. *The NURBS book (2nd ed.)*. Springer-Verlag, New York, NY, USA, 1997.
- [76] <http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html>.
- [77] J.K. Salisbury. *Kinematic and Force Analysis of Articulated Hands*. PhD thesis, Stanford University, 1982.
- [78] S.C. Jacobsen, E.K. Iversen, D.F. Knutti, R.T. Johnson, and K.B. Biggers. Design of the Utah/MIT Dexterous Hand. In *Int'l Conf. on Robotics and Automation*, San Francisco, Ca, April 1986.
- [79] H. Liu, P. Meusel, J. Butterfass, and G. Hirzinger. DLR's multisensory articulated hand II. the parallel torque/position control system. In *Int'l Conf. on Robotics and Automation*, pages 2081–2086, Leuven, Belgium, May 1998.
- [80] C. Lovchik and M. A. Diftler. The robonaut hand: A dexterous robot hand for space. In *Int'l Conf. on Robotics and Automation*, pages 907–912, Detroit, Michigan, May 1999.

- [81] S. B. Kang and K. Ikeuchi. Toward Automatic Robot Instruction from Perception-Recognizing a Grasp from Observation. *IEEE Trans. Robot. Automat.*, 9(4):432–443, 1993.
- [82] M. A. Arbib, T. Iberall, and D Lyons. Coordinated control programs for control of the hands. Hand function and the neocortex. *Experimental Brain Research Supplemental 10*, pages 111–29. Springer-Verlag, 1985.
- [83] Napier J.R. The prehensile movement of the human hand. *Journal Bone Joint Surgery*, 38B:902–913, 1956.
- [84] T. Iberall. The nature of human prehension: Three dextrous hands in one. In *IEEE Intl Conference on Robotics and Automation, (ICRA)*, pages 396–401, April 1987.
- [85] Kamakura N., Matsuo M., Ishii H., Mitsuboshi F., and Miura Y. Patterns of static prehension in normal hands. *The American Journal of Occupational Therapy*, 34(7):437–445, 1980.
- [86] H. Friedrich, V. Grossmann, M. Ehrenmann, O. Rogalla, R. Zöllner, and R. Dillmann. Towards cognitive elementary operators: grasp classification using neural network classifiers. In *IASTED International Conference on Intelligent Systems and Control*, Santa Barbara, USA, 1999.
- [87] M.R. Cutkosky. On Grasp Choice, Grasp Models, and the Design of Hands for Manufacturing Tasks. *IEEE Trans. Robot. Automat.*, 5(3):269–279, 1989.
- [88] T. Wojtara and K. Nonami. Hand Posture Detection by Neural Network and Grasp Mapping for a Master Slave Hand System. In *IEEE/RSJ Intl Conference on Intelligent Robots and Systems (IROS)*, pages 866–871, Sendai, Japan, September 2004.
- [89] R. Zöllner, O. Rogalla, Zöllner J. M., and R Dillmann. Dynamic Grasp Recognition within the Framework of Programming by Demonstration. In *10th IEEE*

International Workshop on Robot and Human Interactive Communication (Roman), Bordeaux and Paris, France, 2001.

- [90] K. Bernardin, K. Ogawara, K. Ikeuchi, and R. Dillmann. A Sensor Fusion Approach for Recognizing Continuous Human Grasping Sequences Using Hidden Markov Models. *IEEE Trans. Robotics*, 21(1):47–57, 2005.
- [91] S. Ekvall and D. Kragić. Interactive Grasp Learning Based on Human Demonstration. In *IEEE Intl Conference on Robotics and Automation, (ICRA)*, New Orleans, USA, April 2004.
- [92] S. Ekvall and D. Kragić. Grasp Recognition for Programming by Demonstration. In *IEEE Intl Conference on Robotics and Automation, (ICRA)*, Barcelona, Spain, April 2005.
- [93] S. B. Kang and K. Ikeuchi. Toward Automatic Robot Instruction from Perception-Temporal Segmentation of Tasks from Human Hand Motion. *IEEE Trans. Robot. Automat.*, 11(5):670–681, 1995.
- [94] S. B. Kang and K. Ikeuchi. Toward Automatic Robot Instruction from Perception-Mapping Human Grasps to Manipulator Grasps. *IEEE Trans. Robot. Automat.*, 13(1):81–95, 1997.
- [95] A. T. Miller and P. K. Allen. Graspit!: A Versatile Simulator for Grasp Analysis. In *ASME Intl Mechanical Engineering Congress*, pages 1251–1258, Orlando, USA, November 2000.
- [96] M. R. Cutkosky and R. D. Howe. Human grasp choice and robotic grasp analysis. *Dextrous Robot Hands*, chapter 1, pages 111–29. Springer-Verlag, 1990.
- [97] K. S. Hale and K. M. Stanney. Deriving haptic design guidelines from human physiological, psychophysical, and neurological foundations. *IEEE Computer Graphics and Applications*, 24:33–39, March/April 2004.

- [98] L.B. Rosenberg. Virtual fixtures: Perceptual tools for telerobotic manipulation. In *Virtual Reality Annual International Symposium*, pages 76 – 82, Seattle, USA, 1993.
- [99] R. Gupta, T. Sheridan, and D. Whitney. Experiments using multimodal virtual environments in design for assembly analysis. *Presence*, 6:318–338, 1997.
- [100] S. Payandeh and Z. Stanisc. On application of virtual fixtures as an aid for telemanipulation and training. In *Proc. Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 18–23, 2002.
- [101] C.-Y. Shing, C.-P. Fung, T.-Y. Chuang, I-W. Penn, and J.-L. Doong. The study of auditory and haptic signals in a virtual reality-based hand rehabilitation system. *Robotica*, 21:211–218, March 2003.
- [102] M. Li and A. M. Okamura. Recognition of operator motions for real-time assistance using virtual fixtures. In *Proc. Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 125–131, 2003.
- [103] J. Aleotti, S. Caselli, and M. Reggiani. Leveraging on a virtual environment for robot programming by demonstration. *Robotics and Autonomous Systems, Special issue: Robot Learning from Demonstration*, 47(2-3):153–161, 2004.
- [104] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381–391, 1954.
- [105] I.S. MacKenzie and W. Buxton. Extending fitts’ law to two-dimensional tasks. In *Proc. ACM Conference on Human Factors in Computing Systems*, pages 219–226, 1992.
- [106] C. Sayers. *Remote Control Robotics*. Springer-Verlag, New York, USA, 1999.
- [107] Wentao Yu, R. Dubey, and N. Pernalet. Telemanipulation enhancement through user’s motion intention recognition and fixture assistance.

- In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 2004.
- [108] D. Aarno, S. Ekvall, and D. Kragić. Adaptive virtual fixtures for machine-assisted teleoperation tasks. In *Int'l Conf. on Robotics and Automation*, Barcelona, Spain, 2005.
- [109] A. C. Boud, C. Baber, and S. J. Steiner. Virtual reality: A tool for assembly? *Presence*, 9:486–496, October 2000.
- [110] E. L. Sallnäs and S. Zhai. Force-feedback improves performance for steering and combined steering-targeting tasks. In *Proceedings of INTERACT, IFIP Conference on Human-Computer Interaction*, pages 97–104, 2003.
- [111] J. T. Dennerlein, D. B. Martin, and C. Hasser. Force-feedback improves performance for steering and combined steering-targeting tasks. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 423–429, 2000.
- [112] L.-T. Cheng, R. Kazman, and J. Robinson. Vibrotactile feedback in delicate virtual reality operations. In *Proc. ACM Int'l Conference on Multimedia*, pages 243–251, 1996.
- [113] J. Accot and S. Zhai. Beyond Fitts' law: models for trajectory-based HCI tasks. In *Proceedings of ACM CHI Conference on Human Factors in Computing Systems*, pages 295–302, 1997.