THESIS

This is to certify that the

dissertation entitled

Visual Learning and Its Application to Sensorimotor Actions

presented by

Wey-Shiuan Hwang

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science and Engineering

[illegible]
Major professor

Date 12/17/99

DATE DUE	DATE DUE	DATE DUE
[illegible] 2003		

Visual Learning and Its Application to Sensorimotor Actions

By

Wey-Shiuan Hwang

A Dissertation

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

1999

ABSTRACT

VISUAL LEARNING AND ITS APPLICATION TO SENSORIMOTOR ACTIONS

By

Wey-Shiuan Hwang

The capability of recognition is a key indicator of the capability of an autonomous agent. However, high dimensionality and variation of sensory input make learning for recognition a very challenging task. The goal of the work presented here is to enable a system to learn directly from unsegmented and unedited sensory streams while interacting with the environment including human teachers.

Automatically generate states for the tasks that the programmer does not know or even understand. Thus, the internal representation must be automatically generated. Although the framework is applicable to various types of learning mode, this work concentrates on the mode where desired actions are imposed in real time during training.

A major technical challenge to realize the above objective is to automatically establish the mapping between a high dimensional input space and an output space. This mapping is accomplished by a doubly clustered subspace-based hierarchical discriminating regression (HDR) tree proposed in this work to efficiently deal with both

classification and regression problems in high dimensional space. The major characteristics of this algorithm include: (1) Clustering is performed in both output space and input space at each internal node and thus the term "doubly clustered." (2) Discriminants in the input space are automatically derived from the clusters in the input space. (3) A hierarchical probability distribution model is applied to the resulting discriminating subspace at each internal node. This realizes a coarse-to-fine approximation of probability distribution of the input samples. (4) A ample-size dependent negative-log-likelihood (NLL) is introduced to relax the per-class sample requirement. (5) The execution of HDR tree is fast, due to the logarithmic time complexity of the HDR tree.

To learn interactively in real time in the environment, an incremental version of HDR tree (IHDR tree) was designed to meet this requirement. The IHDR tree rejects or accepts a learning sample according to the real time response. A forgetting process is applied to the IHDR tree algorithm to constrain the growth of the memory while not having a sudden decrease in the execution performance. In order to have a longer context for the robot, the current state of the robot and previous action feedback were also a part of the input to the robot.

The HDR tree algorithms were tested for different types of data: synthetic data for examining the near-optimal performance, large raw face-image data bases along with a comparison with some major existing methods, such as CART, C5.0 and OC1. In addition to these data, the IHDR tree was applied to the vision-based navigation problem using simulated data. The proposed algorithm was also applied on the real-time tracking and reaching tasks for the robot application.

ACKNOWLEDGMENTS

I wish to thank my thesis advisor Professor Juyang Weng for his guidance and encouragement in the development of this thesis. He provided me with numerous ideas, suggestions, and comments. Without him, this dissertation would not have been completed. I am very lucky to have such a distinguished and responsible professor as my advisor. I am grateful to professors George Stockman, Sridhar Mahadevan, Ning Xi, and Lijian Yang, for serving my committee, sharing ideas with me, and providing critical comments on my dissertation.

I benefited from many discussions with members who participated SHOLSLIF project, especially Shaoyun Chen, Yuntao Cui, Sally Howden, and Dan Swets. During the development of SAIL project, I enjoyed many open discussions with, Greg Bloy, Colin Evans, Mahesh Krishnaswamy, Yong-Beom Lee, Gongjun Li, Guo L. Liu, Jason Sperber, Changjiang Yang, and Yilu Zhang.

During my stay in the Pattern Recognition and Image Processing (PRIP) Laboratory at Michigan State University, I met many distinguished visitors. I would like to thank Dr. Sei-Wan Chen from National Taiwan Normal University and Dr. Chaur-Chin Chen form National Chiao Tung University in Taiwan for many motivated discussions and ideas sharing. A special acknowledgment goes to Dr. JianZhong Qian and Dr. Ming Fang of Siemens Research Corporate, where I spent two summers as a research intern. A broad exposure to various research problems made my internship a valuable experience.

Thanks also to PRIPpies: Paul Albee, Vera Bakic, Scott Connell, Nicolae Duta, Dan Gutchess, Rein-Lein Hsu, Erin Mcgarrity, Salil Prabhakar, Arun Ross, Aditya Vailaya, Jianguo Wang and former PRIPpies Jinlong Chen, Chitra Dorai, Marrie-Piere Dubuisson, Kamen Geuntchev, Lin Hong, Kalle Karu, Yonghong Li, Jinhui Liu, Jianchang Mao, Sharathcha Pankanti, and Nalini Ratha, Bin Yu, and Yu Zhong. Many of them have become good friends of mine. I enjoy many open discussions and wonderful group activities that made my graduate study unforgettable.

Finally, my sincere thank goes to my dear wife Meng-Ling Hsieh for her love, patience, encouragement, support, and understanding. I share all my academic accomplishments with her. Thanks to my daughter, Sophia, who was born in this period, for having brought a lot of joy to my family. I would also like to express my heartful thanks to my mother, Shiow-Jy Chen, for her lifelong love, sacrifice and support. Without her love, sacrifice, and hard work during my childhood and teenage years, I would not have received any education.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

It is well known that vision is extremely difficult, especially for tasks such as recognizing objects in more general settings. Recognition of objects must cope with a wide variety of factors, such as lighting, viewing angle, viewing distance and object changes (e.g., facial expressions). It is known that learning plays a central role in the development of human versatile visual capabilities and it takes place over a long period (e.g., Carey [16], Hubel [30], Anderson [5], Martinez & Kessner [36]). Human vision appears to be more a process of learning and recalling than relying on an understanding of the physical processes of image formation and object-modeling (e.g., the "Thatcher's illusion" [91] and the overhead light source assumption in shape from shading [80]). A large amount of visual data is processed along with other information is learned by a human individual everyday.

1.1 Existing machine learning mode

The importance of learning in developing intelligent machines has long been recognized. However, learning has been used as a technique for manually developing a static system, instead of a fundamental vehicle for continued development. With some thinking, we may realize that the continued development requires more than just learning for longer time. The current prevailing mode of developing an advanced system using learning techniques requires two major stages:

Content-level programming. Content-level programming here means the programming task for modeling the content or knowledge represented in the information of sensory input, other than preprocessing such as Fourier transforms. Modeling visual shape is an example of content-level programming. One must (1) start with a particular problem and understand it, (2) create a problem-space representation for the problem, called problem-specific model, (3) convert the actual problem into a problem-specific model using the selected representation, (4) design an algorithm, (5) write a program that implement the algorithm. The human designers must explicitly define the content-level mapping between the sensory input and the problem-specific model. Here are some examples. For face recognition, one designs 3 neural networks, one for each of the three facial parts, eyes, nose and mouth, respectively, and then designs another neural network that combines the output of the three neural networks. For speech recognition, a separate HMM is responsible for each word and other HMMs for phrases that use outputs from word-level networks. For language understanding, a particular syntax model will take care of a particular syntax. For mobile robot navigation using reinforcement learning, each state of the robot is defined for a specific location of the predefined navigation space.

Data compilation and training. Next, humans manually compile sensory data, which includes creating scenarios, collecting sensory data, labeling data, segmenting data, ordering data, feeding data, etc. For supervised learning, typically a class label is assigned to each segmented data item. For some simpler problems that can be sufficiently modeled by a simulated environment, simulation techniques are extensively used instead of using the real sensory data.

Two major restrictions are direct consequences from such a mode of system development — low quantity and low quality of information fed into the system. In terms of quantity, the computers are allowed to observe far less environmental variation, context variation, and content domain variation than they really need for the tasks that they are assigned for. It is difficult to conduct extensive system training due to the large amount of manual labor that is required in preparing the training data. In terms of quality, these compiled training data are very much *disconnected* from the environment from which the data arise. The rich meaning of the live sensory experience is degenerated into isolated segments, each being tied to a class label which is meaningless to the system. The lack of environmental context of manually fed training segments make it impossible for machines to learn beyond what is modeled by the content-level programming. Unfortunately, what is modeled by content-level programming is typically insufficient for the challenging tasks at hand, due to our human limitation in understanding and fully modeling the complex mechanisms of human cognitive process.

For example, recognition of human faces must cope with a wide variety of variation factors, such as lighting, viewing angle, viewing distance, and facial changes (e.g., expressions, hair styles, eye wears etc). A similar situation is true in speech recognition (e.g., variation in time warping, coarticulation, intonation, age, gender, etc) and language understanding (ambiguity without context, ambiguity without understanding, cultural differences, language differences). It is extremely difficult, if not impossible, for human designers to adequately represent and model these factors, to design effective knowledge-level rules for them, to collect sufficient training data to cover the variations, and to keep the system up to date. Certain degree of automation is very desirable for these challenging problems.

1.2 Comparisons of different approaches

Existing approaches to artificial intelligence fall into the following four categories:

The world-knowledge-based approaches typically require a predefined problem space or world space. Human programmers manually model knowledge and spoon feed knowledge. Researchers in each subfield have been manually developing knowledge-level theories and methods, and using them to write programs or build hardware. Then,they manually "spoon feed" knowledge into the systems at the programming level (e.g., CART [59], CYC [48] [49], lexical database, WordNet [58]). Such an approach may produce a system that appears to produce some intelligent results. However, the limitation of such systems have been recognized [12] [22]. Such a methodology requires a huge amount of human labor and it faces a fundamental limit of humans to fully model and specify the cognitive process required by challenging robotic tasks.

The behavior-based approaches avoid modeling world and instead models robot behavior. The subsumption architecture was proposed by Brooks to allow a more sophisticated behavior layer to be added to the existing primitive behavior layers [14]. Each layer is a finite state machine, with states defined and named by the programmer. The programmer is also responsible to program the state machine in each layer for a desired behavior. Thus, this approach can be characterized by the terms "manually-modeling-behavior and hand-coding-behavior". Aloimonos [4] and others also advocated behavior-based approach for active vision.

The evolutionary approaches are motivated by evolution of biological species. The law of survival of the fittest is used to select advantageous genotypes which code the structure and/or behavior of simulated robots [54] [28]. So far, the selection process have been mostly simulated by computers using a simulated environment, due to the obvious difficulties in carrying on evolutionary process with a large number of robots and performing a long-time physical evolution. The simulation method is attractive due to the low cost benefit. Also, the approach does not require the programmer to code knowledge or behavior rules. However, evolutionary approaches leave the hard task of intelligent system design to the process of random trials and environment selection, an extremely slow and costly process. Three major issues stand out: (1) The Chromosome representation for a sophisticated system. The more sophisticated the system is, the more sophisticated the chromosome is. (2) The extremely high cost of real-robot evolution when high-dimensional perception and sophisticated actions are required, such as vision, speech and language. Simulation is not sufficient for challenging vision, speech and language functionalities. Each system must experience the real-world. (3) The time, required to find a good chromosome, is on the order of a large number of generations. So far, genetic algorithms are typically used for simple environment (e.g., symbolic) and simple behaviors (e.g., symbolic) with carefully designed environment-specific and behavior-specific chromosome representation (see, e.g., Animate and AutonoMouse.

The learning-based approaches include all the existing learning methods, such as supervised learning and reinforcement learning. Learning approaches are typically more efficient than the corresponding evolutionary approaches since the learning mechanism of the system is hand-coded by the programmer with the former, but is either absent or has to emerge from the trial-and-selection process with the latter. For perception of high-dimensional inputs, learning seems the only viable method. Various learning methods have produced impressive results for challenging cognition tasks involving complex modalities, such as visual recognition (e.g.,[92] [88] [61]), speech recognition (e.g., using HMM [79] [37]), vision-guided robot manipulation (e.g., [31]) and vision-guided navigation (e.g.,[97]). Supervised learning is typically more efficient than reinforcement learning. However, learning-based methods have not yet produced systems that truly understand anything. The major reasons include: (1) All the existing learning methods can only be used for a specific problem at a time. For example, an neural network is trained for mapping from every image in a set of face images to a name label. In reinforcement learning using Q-learning algorithm, a problem space must be given first. Then, the human designer must translate the problem state to the internal representation (e.g., states) of the system model (e.g., Q-learning model). However, a system only for a particular problem cannot truly understand anything. (2) A huge amount of manual labor required in translating a specific problem to a learning tool. (3) A huge amount of manual labor required in training a system, including collecting data and testing the system. These reasons hinder further scaling up to more general, larger-size robotic tasks.

In actuality, a particular system may use a combination of several approaches. For example, Robot-Soar [46] combines a world-knowledge-based approach with a learning approach. It requires the human to feed knowledge about the environment and to define the problem space. Then, the system learns to perform predefined tasks. The learning classifier system uses a combination of reinforcement learning and genetic algorithm.

Table 1.1 summarizes the four existing approaches and the new developmental approach. Among the five approaches in the table, the developmental approach appears to require the least amount of human labor in terms of system design; but it requires a large number of population individuals and a huge amount of genetic search time, which makes physical evolution for complex systems impractical. On the other hand, the knowledge-based, behavior-based, and conventional learning-based approaches all require extensive human labor in problem-specific or behavior-specific design, which makes a general-purpose system impractical. The developmental approach seems to be in the middle in terms of human designer's effort and the system developmental cost (time and money). It requires human designers to properly design a general-purpose

Table 1.1: Comparison of Approaches

Approach	Species architecture	World knowledge	System behavior	Task specific
Knowledge-based	programming	manual modeling	manual modeling	Yes
Behavior-based	programming	avoid modeling	manual modeling	Yes
Learning-based	programming	model with	model with	Yes
		parameters	parameters	Yes
Evolutionary	genetic search	model with	model with	Yes
		parameters	parameters	Yes
Developmental	programming	avoid modeling	avoid modeling	No

learning mechanism, but they do not need to explicitly program for the content of what is to be learned — neither for the world knowledge nor for the system behavior. It requires only one or a few physical systems to be built to learn. These physical systems take advantage the richness of intelligence in the human environment. For example, suppose that in reinforcement learning in a computer simulation environment, a punishment is given when a sequence of system actions eventually lead to a failure. The system does not know which action in the action sequence is wrong. In the developmental approach, however, the human teacher can analyze the observed system action sequence and can tell the system which action is wrong by (a) first bringing the system to the right context (e.g., lead it to the location of its bad action) and then (b) giving it a punishment. Such a very powerful tell-you-what-it-is-for mechanism can speed up learning tremendously. This type of information-rich learning environment is more powerful than the time-discounted average reward model in Q-learning and the time-average reward model in R-learning [75] in dealing with ubiquitous delayed-reward situations in learning.

1.3 Developmental learning

Development must be automatic. What is the operational mode of automatic development? A machine agent M may have several sensors. At the time of "birth", its sensors fall into one of the two categories, biased and unbiased[1]. If the agent has a predefined (innate) preference for the signal from a sensor, this sensor is then called biased. Otherwise, it is an unbiased sensor, although a preference can be developed by the agent later through learning. For example, a human being has an innate preference to sweet and bitter tastes from the taste sensor, but does not have a strong preference to visual images of various furniture items. By definition, the *extroceptive*, *proprioceptive* and *interoceptive* sensors are, respectively, those that sense stimuli from external environment (e.g., visual), relative position of internal control (e.g., arm position), and internal events (e.g., internal clock).

1.3.1 AA-learning

We introduce a computational definition of AA-learning (named after *a*utomated, *a*nimal-like learning *without claiming to be complete*) for a machine agent.

Definition 1 *A machine agent M conducts AA-learning at discrete time instances if after its "birth" the following conditions are met for all the time instances $t = 0, 1, 2, \ldots$. (I) M has a number of sensors (biased or unbiased, extroceptive, proprioceptive, or interoceptive), whose signal at time t is collectively denoted by $x(t)$. (II) M has a number of effectors, whose control signal at time t is collectively denoted by*

[1]This is an engineering definition. For a biological organism, it is hard to say that any of its sensors is absolutely unbiased.

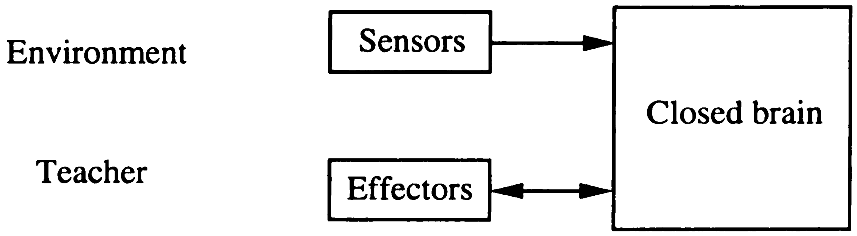


Figure 1.1: An AA-learning agent has two types of channels to interact with the environment, sensors and effectors. The double arrow for the effectors means that the actions imposed by the environment (e.g., human) can be sensed by the "brain."

$a(t)$. The effectors include extro-effectors (those acting on the external world) and intero-effectors (those acting on internal mechanism, e.g., attention). (III) M has a "brain" denoted by $b(t)$ at time t. (IV) At each time t, the time-varying state-update function f_t updates the "brain" based on sensory input $x(t)$ and the current "brain" $b(t)$:

$$b(t+1) = f_t(b(t), x(t)) \tag{1.1}$$

and the action-generation function g_t generates the effector control signal based on the updated "brain" $b(t+1)$:

$$a(t+1) = g_t(b(t+1)) \tag{1.2}$$

where $a(t+1)$ can be a part of the next sensory input $x(t+1)$. (V) The "brain" of M is closed in that after the birth (the first operation), $b(t)$ cannot be altered directly by human teachers for teaching purposes. It can only be updated according to Eq. (1.1).

Fig. 1.1 illustrates the relationship between an AA-learning agent and the world. The design for a "brain" representation $b(t)$, the time-varying state-update function f_t, and the action-generation function g_t determines the AA-learning mechanism as well as the maturation schedule. It is worth noting that t can be dropped from f_t and g_t in the definition since $b(t)$ is not restricted in the definition. For example, any time varying function $f_t(b(t), x(t))$ can be represented by a time invariant function $f(x(t), b(t), t)$ and $(b(t), t)$ can be defined as the "brain." The definition for continuous time is analogous.

From the definition we can see that AA-learning does not require two separate learning and performance phases. The machine agent learns while performing. This is important for continuous, open-ended cognitive development. AA-learning does not require humans to provide edited and segmented sensory input (i.e., no need to spoon-feed data). The system accepts a continuous, unsegmented sensory input stream on-line.

The design of the AA-learning mechanism can take into account various phenomena known about animal learning and human learning. Therefore, this seems to be new ground where artificial intelligence and biological intelligence can converge.

1.3.2 States

In behavior-based learning approaches, the states of an agent are manually bound to a set of predefined task concepts before training [39] [55]. An AA-learning algorithm must automatically generate states without being given any task.

The state of the"brain" is denoted by a state vector $s(t)$. If $s(t)$ is considered a random process, Eqs. (1.1) and (1.2) are closely related to the formulations for Markov decision processes (MDP) [75], or HMMs (hidden Markov models) if the action part is omitted [78] [37]. Indeed, the state transition function f and the decision function g can be based on probability distributions shown below to take into account the uncertainty in states, observations and actions:

$$\mathrm{P}(s(t+1) = s' \mid x(t), s(t) = s)$$

and

$$\mathrm{P}(a(t+1) = a \mid s(t+1) = s')$$

where $\mathrm{P}(\cdot)$ denotes the probability. However, the states in MDPs have been typically defined as a set of symbols and there is no distance metric defined to measure the similarity between any two symbols (see, however, various MDP generalization techniques surveyed by Kaelbling, Littman & Moore [39]).

We define a state s to be a vector in a high dimensional space $\mathcal{S}$. Thus, our state has an explicit representation. $\mathcal{S}$ must contain all the possible sensory input $x \in \mathcal{X}$. In contrast to existing MDP methods, we require that the state records temporal context. Thus, we define the state space to be $\mathcal{S} = \mathcal{X} \times \mathrm{R}(\mathcal{S})$, where $\times$ denotes Cartesian product and $\mathrm{R}(\cdot)$ denotes a re-sampling operator.

1.3.3 Learning types

Eqs. (1.1) and (1.2) identify four components of the AA-learning agent for each time instance t:

$$(a(t+1), s(t+1), s(t), x(t)). \tag{1.3}$$

They involve three entities: action, state, and sensor.

Depending on whether the action is imposed or not, the learning can be classified into *action-imposed* learning and *action-autonomous* learning. *Action-imposed* learning is such that the extro-effector part of $a(t+1)$ is supplied by the trainer. For example, hand-in-hand learning can be used by human adult to teach a child how to use a pen. Otherwise, the learning is *action-autonomous* learning.

Depending on whether the state $s(t)$ is imposed or not, learning can be classified into *state-imposed* and *state-autonomous.* The state-imposed learning is such that $s(t)$ and $s(t+1)$ are set by the human trainer during the learning. If a learning method requires a task-specific representation, the representation typically determines the meaning of states and thus the learning must use state-imposed learning. AA-learning is state-autonomous learning. As explained earlier, with AA-learning, the state of the system is determined by the developmental algorithm autonomously. Another concept is the state-readability. If the state of the system is not directly readable to the teacher, the learning is state-readable. Otherwise, it is state-unreadable.

Depending on whether the unbiased sensor is used or not, the learning can be classified into reinforcement learning and communicative learning. *Reinforcement* learning is such that a *biased* sensor is used to reinforce or punish certain response from the machine agent. *Communicative* learning is such that only *unbiased* sensors are used in learning. This requires that the agent to correctly interpret the signal from unbiased sensors, either it is an instruction for action, an encouragement, an explanation, etc. Learning by a human adult is mostly conducted in the communicative learning mode.

The learning type can be represented by a 3-tuple (A, S, X) where $A \in \{i, a\}$ denotes if action is imposed or autonomous, $S \in \{i, a\}$ denotes the state is imposed or autonomous, and $X \in \{r, c\}$ denotes the biased sensor is used or not. There are 8 different 3-tuples, representing a total of 8 different learning types. AA-learning is state-autonomous learning. Thus, there are 4 types of AA-learning: Type (1) action-imposed and reinforcement, Type (2) action-imposed and communicative, Type (3) action-autonomous and reinforcement, and Type (4) action-autonomous and communicative. It is worth noting that these four types are typically interleaved in a natural learning environment for animals and humans. These definitions are required beyond the coarse classic definition of supervised and unsupervised learning[2]. This thesis only studies the type (2) action-imposed and communicative learning and other types are not addressed further in this thesis. If the trainer imposes an action on an effector at any time through, e.g., a joystick, the system performs action-imposed learning for that effector. Otherwise, the system performs action-autonomous learning, during which communicative learning is used.

[2]From the most strict definition of unsupervised learning, all the above learning modes are supervised by human to some degree. It is difficult to identify any type of learning that is completely unsupervised.

1.4 Learning in hight dimensional space

To realize an AA-learning, a powerful tool to map high dimensional sensory inputs to continuous action outputs is required. There are two issues for this mapping tools.

- Representation: There are two possible representation: the model-based and the appearance-based. The model-based approaches use manually defined features to represent objects in the images. A lot of efforts has been made using this approach [35] [45]. The focus of this approach is to design an efficient algorithm from a set of manually selected features. This method is appealing because of its efficiency. However, the model-based approach is difficult to generalize. For example, the face features become useless when a car image database is presented. To make domain-extensible, we can not use the model-based representation.

  The appearance-based approaches have recently drawn a lot of attention in machine vision [92] [61] [88]. Using an appearance-based approach, any sensory input is considered as a vector in a high-dimensional space, no matter the input is an image or something else. The retrieval task then becomes the problem of finding the best match among the training samples. To use fewer features to represent a set of images, the principal component analysis (PCA) has been used for face recognition [92]. PCA can reconstruct the images with the least mean square errors. However, the PCA features which are good for image reconstruction are not necessarily good for recognition. The features derived from Fisher's linear discriminant analysis (LDA) are better for recognition [88] provided the samples are sufficiently rich to cover the typical variation within each class.

However, LDA cannot, by itself, judiciously group similar classes into a single class for coarse classification. Such a grouping will equivalently increase the number of samples in each merged class and thus will allow better estimate of the within-class variation which is critical for the effectiveness of LDA. We cast a classification problem into a regression problem so that each class is represented by a vector in the output space. The similarity of classes can be measured through distance in the output space, which guides class merging in the input space for each coarse classification. Even in the case where the users assign distance-preserved class vectors, merged classes have distance measures that reflect how classes are merged.

- Organization and incremental learning: It is very important for a real time system to well-organize the represented features so that the retrieval is both fast and successful. Linear search is very time-consuming which makes it not practical for a real time system. One way to solve this problem is to use a classification and regression tree.

  Traditionally, classification and regression trees use univariate split at each internal node, such as in CART [11], C5.0 [77] and many others. This means that the partition by each node uses a hyper-plane that are orthogonal to an axis in the input space X. Multivariate linear splits correspond to partition hyper-planes that are typically not orthogonal to any axis of the input space. Trees that use multivariate splits are called oblique tree. As early as in the mid 70's, Freidman [23] proposed a discriminating node-split for building a tree which would result in an oblique tree. OC1 by Murthy et al. 1994 [40] and SHOSLIF tree by Weng 1994 [95] and 1996 [96] are two methods for constructing oblique trees. For an extensive survey of decision trees, see a recent survey by Murthy [62]. OC1 uses iterative search to find a split. SHOSLIF uses principal component analysis (PCA) and linear discriminant analysis (LDA) to directly compute splits. Multivariate nonlinear splits correspond to curved partition surfaces with any orientation. An incremental hierarchical discriminant regression (IHDR) tree algorithm is proposed to use multivariate nonlinear splits, with multivariate linear splits as a special case.

1.5 Thesis overview

The rest of this thesis is organized as follows. Chapter 2 introduces a new techniques for the regression and classification problems. This algorithm casts classification problems (class labels as output) and regression problems (numeric values as output) into a unified regression problem. The method automatically derives discriminating features in input space by clustering samples from output space. A decision tree structure is adopted to facilitate the speed of the execution. A sample-size dependent negative-log-likelihood measurement is proposed for the smooth transition among different statistic measurement which is suitable for different sample population. Several experiments were conducted to verify the performance of the proposed algorithms.

In Chapter 3, the incremental version of the hierarchical discriminant regression tree (IHDR tree) is described. This is necessary for the on-line training and testing of the robotic tasks. The proposed algorithms use amnesic average to compute the necessary statistics so that the computation complexity is low and the real time performance can be achieved. In addition to the experiments done in Chapter 2, this algorithm was tested for the vision-based navigation problem using real images collected in the indoor environment.

The current technology in computer vision and pattern recognition requires humans to collect images, store images, segment images for computers and train computer recognition systems using these images. It is unlikely that such a manual labor process can meet the demands of many challenging recognition tasks that are critical for generating intelligent behavior, such as face recognition, object recognition and speech recognition. In Chapter 4, a system to learn directly from continuous image stream is described. We demonstrate the performance of the algorithm on the problem of face recognition using video sequences from different subjects.

In Chapter 5, the proposed algorithms were applied to the real robot. The architecture of the SAIL (named after Self-organizing Autonomous Incremental Learner) is described in this chapter. Since the robot equips with several sensors and actuators, multiple thread processes are implemented for the software settings. The tracking tasks were experimented in the real robot testing.

Finally, Chapter 6 concludes the dissertation by summarizing the contribution and giving the future research directions.

Chapter 2

Hierarchical Discriminant Regression Tree: Batch Learning Mode

2.1 Introduction

The capability of computers to efficiently and effectively retrieve information from image databases gives a significant impact on the progress of digital library technology. A central task of a multimedia information system is to efficiently store, fast and correctly retrieve, and easily manage images in the database [67] [77].

An essential issue for image database is the representation of the image. We can categorize the content-based image retrieval into two types: the model-based and the appearance-based. The model-based approach uses manually defined features to represent objects in the images. A lot of efforts has been made in this approach [47] [7] [35] [45] [27]. Most of them have been focusing on designing an efficient algorithm from a set of manually selected features. The strength of model-based approach is the efficiency in representing images. With a proper design and a restricted domain of images, only a very small number of parameters is sufficient to represent the objects in the image and to distinguish among different objects. However, the model-based approach is difficult to generalize. For examples, given a face image database, the designer needs to manually find the features for faces. The face features become be useless when a car image database is presented. The designer has to find another set of features for car images. Such a process of manually designing features cannot scale up to large and complex domains since there are countless models to be built.

The appearance approach has recently drawn much attention in machine vision [92] [61] [63] [96]. Instead of relying on human designer to define features, the appearance-based approach enables machines to automatically derive features from image samples. To do so, it considers a two dimensional image as a long vector. Statistical classification tools are applied directly to the sample vectors. One example is the nearest neighbor (NN) classifier. As is well-known, NN classifier is very time and space consuming for high-dimensional image space or a large image database. To use fewer features to represent a set of images, the principal component analysis (PCA) has been used for face recognition [92] [70]. PCA can optimistically reconstruct the images represented with the least mean square errors. However, the features which can well represent the original data set are not necessarily good for the purpose of classification. The features derived from the linear discriminant analysis (LDA) are meant for well distinguishing different classes and thus are relatively better for the purpose of classification, provided that the samples contain sufficient information [88].

The second issue for image database is how to organize the represented features so that the retrieval is both fast and successful. Linear search is very time-consuming which makes it not practical for very large image databases. One way to solve this problem is to use a decision tree. A well designed decision tree can retrieve a matched sample with a logarithmic time complexity. This is a very useful characteristic for large image databases. There is a very rich literature about decision trees, see surveys [19] [82] [29] [62] [11]. However, the applications of decision trees have been traditionally for a low-dimensional feature space with manually selected features. This is true largely because humans cannot define a large number of useful features. Appearance-based approach drastically changed this situation. Traditional decision trees for low-dimensional input space have been found not suited for input dimensionality of a few thousands and up, even after data-dimensional reduction using techniques such as PCA. A major reason is the high complexity of sample distribution that cannot be adequately captured by a single-level PCA. As demonstrated by [87], if a different subspace is computed at each internal node of the tree, a better generalization power results. A series of limitations of existing tree classifiers have motivated the work presented here with advances and advantages summarized in the abstract.

2.2 Classification and regression

The tasks of discriminant analysis can be categorized into two types according to their outputs: class-label (symbolic) output and numerical output. The former case is called classification and the latter case is called regression. A classification task can be stated as follows. Given training sample set $L = \{(x_i, l_i) \mid i = 1, 2, \ldots, n\}$, where $x_i \in \mathcal{X}$ is an input (feature) vector and l_i is the symbolic label of x_i, the task is to determine the class label of any unknown input $x \in \mathcal{X}$. A regression task is similar to the corresponding classification one except that the class label l_i is replaced by a vector y_i in output space so that $y_i \in \mathcal{Y}, i = 1, 2, \ldots, n$. A regression task is stated as follows. Given training set $L' = \{(x_i, y_i) \mid i = 1, 2, \ldots, n\}$ and any testing vector $x \in \mathcal{X}$, the goal is to estimate the vector $y(x) \in \mathcal{Y}$.

In a classification problem, the class labels themselves do not provide more information in terms of how different two classes are. We would like to cast a classification task regressive one so that we can conveniently form coarse classes by merging some original classes. These coarse classes are useful for performing coarse-to-fine classification and regression using a decision tree, as we will explain later in this chapter.

There are three ways to cast a classification task into a regression one. (1) If a cost matrix $[c_{ij}]$ is readily available from application, where c_{ij} is the cost of confusing classes i and j, one can embed n class labels into an $(n-1)$-dimensional Euclidean outputs space by assigning vector y_i to class i, $i = 1, 2, \ldots, n$, so that $||y_i - y_i||$ is as close to as c_{ij}, as much as possible. This process is not always possible since a pre-defined cost matrix $[c_{ij}]$ is not always easy to provide. (2) Canonical mapping.

Map n class labels into an n dimensional output space so that the i-th class label corresponds to a vector in which the i-th component is 1 and all other components are zeros. After this mapping, the distance between any two different class labels is the same: 1. This label mapping does not assign different distances to different output vectors but will do so for coarse classes in a coarse-to-fine classification. (3) Mapping labels into the input space. Each sample (x_i, l_i) belonging to class l_i is converted to (x_i, y_i) where y_i, the vector class label, is the mean of all x_i that belong to the same class. This label-mapping scheme considers the distance in input space as that between different classes. In many applications, this is a desirable way. How can we get the class label from the numerical vector in $\mathcal{X}$? In each leaf node of the regression tree, each training sample (x_i, y_i) has a link to label l_i so that when (x_i, y_i) is found as a good match for unknown input x, l_i is directly output as the class label. There is no need to search for the nearest neighbor in the output space for the corresponding class label.

To cast a classification problem into a regression one can be achieved by the above three methods. The other way around is not always possible. One cannot map a numeric output space into a set of class labels without losing the numeric properties among an infinite number of possible numerical vectors. Therefore, a regression problem is more general than the corresponding classification problem. As can be seen, the numerical representation in output space allows us to form hierarchical discriminating subspaces for constructing a decision tree.

2.3 Discriminant analysis for numerical output

Consider the general regression problem: approximating a mapping $h : \mathcal{X} \mapsto \mathcal{Y}$ from a set of training samples $\{(x_i, y_i) \mid x_i \in \mathcal{X},\ y_i \in \mathcal{Y},\ i = 1, 2, \ldots, n\}$. If y_i was a class label, linear discriminant analysis (LDA) [24] can be applied since the within-class scatter and between-class scatter matrices are all defined. Unfortunately, if each class has only very few classes, the within-class scatter matrix is poorly estimated and the LDA is not very effective. If the classification problem is cast into a regression one, it is possible to form coarse classes each having more samples which enables better estimation of the within class scatter matrix. However, if y_i is a numerical output, which can take any value for each component, it is a challenge to figure out an effective discriminant analysis procedure.

To attack this challenge, a new hierarchical statistical modeling method is introduced. Consider the mapping $h : \mathcal{X} \mapsto \mathcal{Y}$, which is to be approximated by a regression tree[1], called hierarchical discriminant regression (HDR) tree, for the high dimensional space $\mathcal{X}$. Our goal is to automatically derive discriminant features although no class label is available (other than the numerical vectors in space $\mathcal{Y}$). In addition, for computational efficiency, each sample (x_i, y_i) must be processed to update the HDR tree using only a minimal amount of computation.

Two types of clusters are formed at each node of the HDR tree — y-clusters and x-clusters. as shown in Fig. 2.1. The y-clusters are clusters in the output space $\mathcal{Y}$ and x-clusters are those in the input space $\mathcal{X}$. There are a maximum of q (e.g.,

[1]A regression tree is, by definition, a decision tree whose output is a numeric vector while a classification tree is a decision tree whose output is a class label [11].

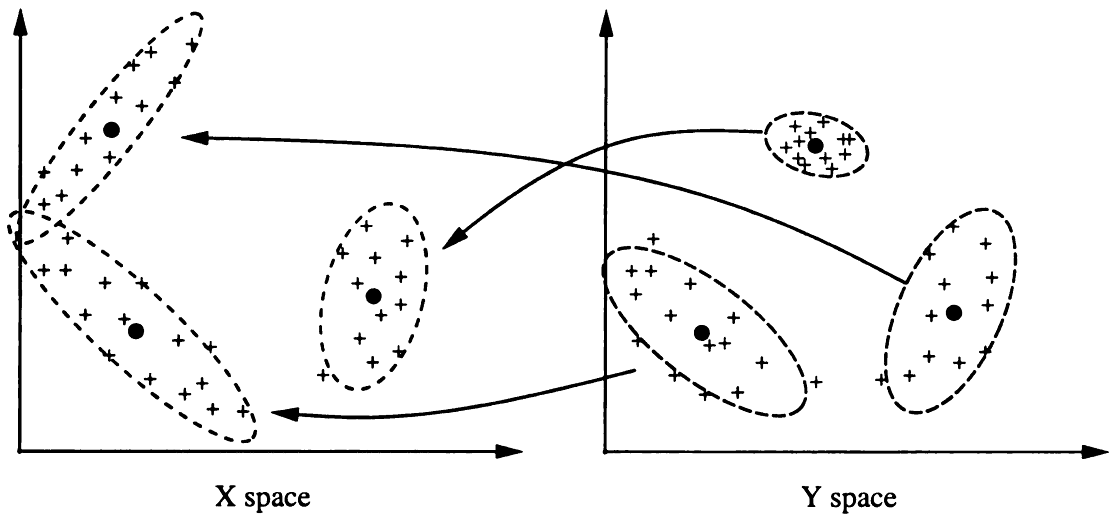


Figure 2.1: Y-clusters in space $\mathcal{Y}$ and the corresponding x-clusters in space $\mathcal{X}$. The first and the second order statistics are updated for each cluster. By default, the normalized Mahalanobis distance is used for x-cluster and the Euclidean distance is used for distance to y-cluster.

$q = 6$) clusters of each type at each node. The q y-clusters determine the virtual class label of each training sample (x, y) based on its y part. The virtual class label is used to determine which x-cluster the input sample (x, y) should update using its x part. Each x-cluster approximates the sample population in $\mathcal{X}$ space for the samples that belong to it. It May spawn a child node from the current node if a finer approximation is required. At each node, y in (x, y) finds the nearest y-cluster in, e.g., Euclidean distance (more general distance metrics can also be used). This y-cluster indicates to which corresponding x-cluster the input (x, y) belongs. Then, the x part of (x, y) is used to compute the statistics of the x-cluster (the mean vector and the covariance matrix). These statistics of every x-cluster are used to estimate the probability for the sample (x, y) to belong to the x-cluster, whose probability distribution is modeled as a multidimensional Gaussian at this level. A total of q centers of the q x-clusters give $q - 1$ discriminant features which span $(q - 1)$-

dimensional discriminant space. A probability-based distance (to be discussed in Section 2.4) from x to each of the q x-clusters is used to determine which x-cluster should be further searched. If the probability is high enough, the sample (x, y) should further search the corresponding child (maybe more than one but with an upper bound k) recursively, until the corresponding terminal nodes are found.

For computational efficiency, none of the x-clusters and y-clusters keep actual input samples, unlike the traditional clustering methods. Only the first orders of statistics are used to represent the clusters. For example, each y-cluster keeps the mean vector and the covariance matrix, depending on the distance metric in the $\mathcal{Y}$ space, while each x-cluster keeps the mean vector and the full covariance matrix in an efficient form.

In summary, the algorithm recursively builds a HDR tree from a set of training samples. The deeper a node is in the tree, the smaller the variances of its x-clusters are. The following is the outline of the algorithm for tree building and retrieval.

Procedure 1 **`BuildSubtree`**: *Given a node N and a subset S' of the training samples that belong to N, among the samples in $S = \{(x_i, y_i) \mid x_i \in \mathcal{X},\ y_i \in \mathcal{Y},\ i = 1, 2, \ldots, n\}$, build the subtree which roots from the node N using S recursively. At most q clusters are allowed in one node.*

1. *Let p be the number of the clusters in node N.*

    - *Call* `Clustering-Y` *(procedure 2) to obtain p y-clusters.*

    - *If y_i belongs to i-th y-cluster, then x_i belongs to i-th x-cluster.*

2. *Compute the mean and covariance matrices of each x-cluster.*

3. *For every x_i of (x_i, y_i) in S':*

    - *Find the nearest x-cluster j according to the probability-based distances.*

    - *Suppose that the sample (x_i, y_i) belongs to cluster j.*

4. *For each cluster j, only a portion of samples S_j was assigned to the x-cluster. If the largest Euclidean distance among y_i's in the x-cluster is larger than a number δ_y, a child node N_j of N is created from the x-cluster and this procedure is called recursively with input samples S_j and node N_j. The number δ_y represents the sensitivity of the HDR tree in the space $\mathcal{Y}$.*

Procedure 2 `Clustering-Y`: *Given a set of output vectors $Y = (y_1, y_2, \ldots, y_n)$, return p y-clusters. $p \leq q$, where q represents the maximum clusters allowed in one node.*

1. *Let the mean Y_1 of y-cluster 1 be y_1. Set $p = 1$ and $i = 2$.*

2. *For i from 2 to n do*

    (a) *Find the nearest y-cluster j for y_i.*

    (b) *Compute the Euclidean distance $d = dist(y_i, Y_j)$.*

    (c) *If $d \geq \delta_y$ and $p < q$, let the mean Y_{p+1} of y-cluster $p+1$ be y_i. Set $p = p+1$.*

    (d) *Otherwise, update y-cluster j using the new member y_i.*

The procedure to create a HDR tree just calls procedure `BuildSubtree` with root R and all the training samples $S = \{(x_i, y_i) \mid x_i \in \mathcal{X},\ y_i \in \mathcal{Y},\ i = 1, 2, \ldots, n\}$. The procedure for query the HDR tree for an unknown sample x is described in below.

Procedure 3 `Retrieval`*: Given a HDR tree T and sample x, estimate the corresponding output vector y. A parameter k specifies the upper bound in the width of parallel tree search.*

1. *From the root of the tree, compute the probability-based distance to every cluster in the node. Select at most top k x-clusters which have the smallest probability-based distances to x. These x-clusters are called active x-clusters.*

2. *For every active cluster received, check if it points to a child node. If it does, mark it inactive and explore its child node by computing the probability-based distances of x-clusters in the child node. At most k^2 active x-clusters can be returned.*

3. *Mark at most k active x-clusters according to the smallest probability-based distances.*

4. *Do the above steps 2 through 3 recursively until all the resulting active x-clusters are all terminal.*

5. *Let the cluster c have the shortest distance among all reached leaf nodes. Output the corresponding mean of its y-cluster as the estimated output y for x.*

The above tree gives a coarse-to-fine probability model. If Gaussian distribution is used to model each x-cluster, this is a *hierarchical version* of the well-known mixture-of-Gaussian distribution models: the deeper the tree is, the more Gaussians are used and the finer are these Gaussians. At shallow levels, the sample distribution is approximated by a mixture of large Gaussians (with large variances). At deep levels, the sample distribution is approximated by a mixture of many small Gaussians (with small variances). The multiple search paths guided by probability allow a sample x that falls in-between two or more Gaussians at each shallow level to explore the tree branches that contain its neighboring x-clusters. Those x-clusters to which the sample (x, y) has little chance to belong are excluded for further exploration. This results in the well-known logarithmic time complex for tree retrieval: $O(\log m)$ where m is the number of leaf nodes in the tree, assuming that the number of samples in each leaf node is bounded above by a constant.

2.4 Distance in discriminating space

2.4.1 Discriminating subspace

In the above section, it is necessary to estimate the distance for an input vector x to belong to an x-cluster. For a real-time system, it is typically the case that the system cannot afford to keep all the samples in each cluster. Thus, each cluster will be represented by some statistical measures with an assumed distribution.

We first consider x-clusters. Each x-cluster is represented by its mean as its center and the covariance matrix as its size. However, since the dimensionality of the space $\mathcal{X}$ is typically very high, it is not practical to directly keep the covariance matrix.

If the dimensionality of $\mathcal{X}$ is 3000, for example, each covariance matrix requires $3000 \times 3000 = 9,000,000$ numbers! We adopt a more efficient method.

As explained in Section 2.3, each internal node keeps up to q x-clusters. The centers of these q x-clusters are denoted by

$$C = \{c_1, c_2, ..., c_q \mid c_i \in \mathcal{X}, i = 1, 2, ..., q\}. \qquad (2.1)$$

The locations of these q centers tell us the subspace $\mathcal{D}$ in which these q centers lie. $\mathcal{D}$ is a discriminant space since the clusters are formed based on the clusters in space $\mathcal{Y}$. In this space, the between-cluster scatter and within-cluster scatter can be computed. Suppose that the number of samples in cluster i is n_i and thus the grand total of samples is $n = \sum_{i=1}^{q} n_i$. The mean of the cluster center, denoted by c is computed as

$$c = \frac{1}{n} \sum_{i=1}^{q} n_i c_i.$$

The covariance matrix of cluster i is denoted by Γ_i, $i = 1, 2, ..., q$. The within-cluster scatter matrix is the weighted average of the within-cluster scatter matrices:

$$S_w = \frac{1}{n} \sum_{i=1}^{q} n_i \Gamma_i. \qquad (2.2)$$

The between-cluster scatter matrix is the sample covariance matrix for the cluster centers:

$$S_b = \frac{1}{n}\sum_{i=1}^{q} n_i(c_i - c)(c_i - c)^T \tag{2.3}$$

The sample mixture matrix is the covariance matrix of all the samples regardless of their cluster assignments, and it is also equal to

$$S_m = S_w + S_b.$$

The Fisher's linear discriminant analysis [24] [100] finds a subspace that maximizes the ratio of between-cluster scatter and within-cluster scatter: $|S_b|/|S_w|$. Since the entire discriminant space $\mathcal{D}$ is used, it is not necessary to consider the within-cluster scatter here in finding $\mathcal{D}$ and thus simplifies the computation. Once this discriminating space $\mathcal{D}$ is found, the size-dependent negative-log-likelihood (SDNLL) distance, as discussed in Section 2.4.2, will be applied to take care of the reliability of each dimension in $\mathcal{D}$ using information that is richer than the matrix S_w.

2.4.2 Size-dependent negative-log-likelihood

The characteristics of different metrics

We need a system that fully uses the information available no matter how many samples have been observed. This is an issue that has received little attention since it is typically assumed that sufficient samples are available for each class but it is often not the case in practice. In terms of belongingness of a vector to a cluster, there are three measures, likelihood, Mahalanobis distance and Euclidean distance. The likelihood assumes Gaussian density and thus uses the covariance matrix of each individual cluster. It requires that each x-cluster has enough samples to estimate the $(q-1) \times (q-1)$ covariance matrix. It is the most demanding among the three in the richness of observations. The Mahalanobis distance uses the average of covariance matrices. It is less demanding since it just requires that the average of covariance matrix has reasonably rich observations but not necessarily every x-cluster. The Euclidean distance is estimated by $\rho^2 I$ and thus has only one parameter ρ. Thus it is the least demanding. When very few samples are available for all the clusters, the Euclidean distance is the suited distance.

Let us consider the negative-log-likelihood (NLL) defined from Gaussian density of dimensionality $q-1$:

$$G(x, c_i) = \frac{1}{2}(x - c_i)^T \Gamma_i^{-1} (x - c_i) + \frac{q-1}{2} \ln(2\pi) + \frac{1}{2} \ln(|\Gamma_i|). \tag{2.4}$$

We call it Gaussian NLL for x to belong to the cluster i. c_i and Γ_i are the cluster mean and covariance matrix, respectively. Similarly, the Mahalanobis NLL and Euclidean NLL are defined as:

$$M(x, c_i) = \frac{1}{2}(x - c_i)^T \Gamma^{-1} (x - c_i) + \frac{q-1}{2} \ln(2\pi) + \frac{1}{2} \ln(|\Gamma|). \tag{2.5}$$

$$E(x, c_i) = \frac{1}{2}(x - c_i)^T \rho^2 I^{-1}(x - c_i) + \frac{q-1}{2}\ln(2\pi) + \frac{1}{2}\ln(|\rho^2 I|). \tag{2.6}$$

Γ is the within-class scatter matrix of each node — the average of covariance matrices of q clusters.

When the number of samples in the node is small, the Euclidean NLL is preferred. Gradually, as the number of samples increases, the within-class scatter matrix of q x-clusters are better estimated. Then, the Mahalanobis NLL is more suitable. When a cluster has very rich observations, the full Gaussian NLL for it will be the best choice.

In order to properly decide when and how to transit among the three NLLs, different characteristics of them are discussed here. Suppose that the input space is $\mathcal{X}$ and the discriminating subspace for an internal node is $\mathcal{D}$. The Euclidean NLL treats all the dimensions in the discriminating subspace $\mathcal{D}$ the same way, although some dimensionalities can be more important than others. It has only one parameter ρ to estimate. The Mahalanobis NLL uses within-class scatter matrix S_w computed from all the samples in all the q x-clusters. It uses the inverse matrix S_w^{-1} as the weight in computing NLL. The meaning of this Mahalanobis matrix weight S_w^{-1} is as follows. The matrix S_w^{-1} properly rotates the original basis $b_1, b_2, ..., b_{q-1}$ of the subspace $\mathcal{D}$ so that the correlation about the new basis vectors $b'_1, b'_2, ..., b'_{q-1}$ all vanishes. Then S_w^{-1} applies to each rotated basis vector b'_i, $i = 1, ..., q-1$, a weight which is the inverse of the sample variance along b'_i. By the way, using Mahalanobis

NLL as the weight for subspace $\mathcal{D}$ is equivalent to using Euclidean NLL in the basis computed from Fisher's LDA procedure [24] [88]. Thus, the Mahalanobis NLL takes care of the reliability of different input components but the Euclidean NLL does not. The former does not only decorrelate the input components, but also weight each decorrelated new components. The number of parameters in S_w is $q(q-1)/2$, and thus, the Mahalanobis NLL requires more samples than the Euclidean NLL. Next, consider the Gaussian NLL. As can be seen, the Mahalanobis NLL does not treat different x-lusters differently because it uses a single within-class scatter matrix S_w for all the q x-clusters in each internal node. This is not the case with Gaussian NLL. Using Gaussian NLL, $L(x, c_i)$ in Eq.(2.4) uses the covariance matrix Γ_i of x-cluster i. In other words, Gaussian NLL not only decorrelates the correlations but also applied a different weight at different location along each rotated basis. Note that the decision boundary of the Euclidean NLL and the Mahalanobis NLL is linear and that by the Gaussian NLL is quadratic.

In other words, consider the partition of the space $\mathcal{D}$ into two regions, one containing c_1 and the other containing c_2. If $L(x, c_1) < L(x, c_2)$, then x is with c_1. Otherwise x is with c_2. The boundary is marked by all x's that satisfy $L(x, c_1) = L(x, c_2)$. Such a boundary is a hyperline (linear) if Euclidean or Mahalanobis NLL is used for $L(\cdot, \cdot)$. The boundary is a quadratic in general if the Gaussian NLL is used.

The transition among different metrics

How do we realize a proper automatic transition from the three NLLs? We would like to make this transition smooth when the number of samples increases. Two alternative measurements of maturity for each cluster i can be defined, the number of samples n_i and the elapsed time t_i since its creation. The former is appropriate for off-line application where the elapsed time is not applicable. The latter is more suited for real-time application where a very large number of similar samples may be observed during a short time period, causing n_i to increase significantly without having observed enough variation in the data. In the following, n_i is used as the maturity measurement. For each node, the within-class scatter S_w is computed, which has a total of $n = \sum_{i=1}^{q} n_i$ samples. For each x-cluster in the node, we start with a scalar covariance matrix $\rho^2 I$. We would like to use the same Gaussian NLL expression as the measure for belongingness but gradually change the estimated covariance matrix according to the number of samples received for the required matrix. For the three types of NLLs, we have three matrices, $\rho^2 I$, S_w, and Γ_i. Consider the number of scales received to estimate each parameters, called number of scales per parameter (NSPP), of the element of the matrices. The NSPP for $\rho^2 I$ is $(n-1)(q-1)$, since the first sample does not give any estimate of the variance and each independent vector contains $q - 1$ scales. A bounded NSPP is defined to limit the growth of NSPP so that other matrices that contain more scalars can take over when there are a sufficient number of samples for them. Thus, the bounded NSPP for $\rho^2 I$ is

$$b_e = \min\{(n-1)(q-1), n_s\}$$

where n_s denotes the switch point for the next more complete matrix to take over. At this switch point, a weight of about 50% will be used for $\rho^2 I$ and another weight of 50% for the next matrix. How large should n_s be? Consider a serious of random variables drawn independently from a distribution with a variance σ^2, the expected sample mean of n random variables has a covariance $\sigma^2/(n-1)$. We can choose a switch confidence value α for $1/(n-1)$. When $1/(n-1) = \alpha$, we consider that the estimate can take about a 50% weight. Thus, $n = 1/\alpha + 1$. As an example, let $\alpha = 0.1$ meaning that we trust the estimate with 50% weight when the expected variance of the estimate is reduced to about 10% of that of a single random variable. We get then $n = 11$, which leads to $n_s = 11$.

Next, consider the NSPP for S_w matrix for the Mahalanobis NLL. The number of independent vectors received is $n-q$ because each of the q x-cluster requires a vector to form its mean vector. Thus, there are $(n-q)(q-1)$ independent scalars. There are $(q-1)q/2$ estimated parameters in the (symmetric) matrix S_w. Thus, the NSPP for S_w is

$$\frac{(n-q)(q-1)}{(q-1)q/2} = \frac{2(n-q)}{q}.$$

To avoid the value to be negative when $n < q$, we take NSPP for S_w to be

$$\max\left\{\frac{2(n-q)}{q}, 0\right\}.$$

The bounded NSPP for S_w is

$$b_m = \min\left\{\max\left\{\frac{2(n-q)}{q}, 0\right\}, n_s\right\}.$$

Since the Gaussian NLL cannot be trusted until every matrix Γ_i has received enough samples, we define the NSPP for Γ_i estimates for the Gaussian NLL as the minimum among all the q x-clusters:

$$b_g = \min_{1 \le i \le q} \left\{ \frac{2(n_i - 1)}{q} \right\}, \tag{2.7}$$

This is somewhat conservative since it may be the cases where x-cluster that has the least samples is not among the nearest x-clusters. The above NSPP is meant to contain the worst error (when the x-cluster with the fewest samples is the nearest x-cluster). Alternatively, if we are interested in containing the mean error, we may choose NSPP to be the average number of samples per x-cluster:

$$b_g = \frac{1}{q} \sum_{i=1}^{q} \left\{ \frac{2(n_i - 1)}{q} \right\} = \frac{2(n - q)}{q^2}. \tag{2.8}$$

In the above computation, we only consider the x-clusters that have at least one sample. It is worth noting that the NSPP for the Gaussian NLL does not need to be bounded, since among our models it is the best estimate with a large number of samples.

Let us consider the three NSPP: b_e, b_m and b_g. From the definitions, we know that they grow roughly at rates n, $2n/q$ and $2n/q^2$, respectively. b_e gets saturated by n_s the earliest. Then, b_m does. b_g never saturates. Table 2.1 summarizes the result of the NSPP values of the above derivation.

We define a *size-dependent scatter matrix* (SDSM) W_i as a weighted sum of three

Table 2.1: Characteristics of Three Types of Scatter Matrices

Type	Euclidean $\rho^2 I$	Mahalanobis S_w	Gaussian Γ_i
NSPP	$(n-1)(q-1)$	$\frac{2(n-q)}{q}$	$\frac{2(n-q)}{q^2}$

matrices:

$$W_i = w_e \rho^2 I + w_m S_w + w_g \Gamma_i \tag{2.9}$$

where $w_e = b_e/b$, $w_m = b_m/b$, $w_g = b_g/b$ and b is a normalization factor so that these three weights sum to 1: $b = b_e + b_m + b_g$. Using this size-dependent scatter matrix W_i, the *size-dependent negative log likelihood* (SDNLL) for x to belong to the x-cluster with center c_i is defined as

$$L(x, c_i) = \frac{1}{2}(x - c_i)^T W_i^{-1}(x - c_i) + \frac{q-1}{2}\ln(2\pi) + \frac{1}{2}\ln(|W_i|). \tag{2.10}$$

It is worth noting the relation between LDA and SDNLL metric. Fisher's LDA in space $\mathcal{D}$ gives a basis for a subspace $\mathcal{D}' \subseteq \mathcal{D}$. This basis is a properly oriented and scaled version for $\mathcal{D}$ so that the within-cluster scatter in $\mathcal{D}'$ is a unit matrix [24] (Sections 2.3 and 10.2). In other words, all the basis vectors in D' are already weighted according to the within-cluster scatter matrix S_w of D. If $\mathcal{D}'$ has the same dimensionality as $\mathcal{D}$, the Euclidean distance in $\mathcal{D}'$ is equivalent to the Mahalanobis distance in $\mathcal{D}$, up to a scale factor. However, if the covariance matrices are very different across different x-clusters and each of them has enough samples to allow a good estimate of individual covariance matrix, Fisher's LDA in space $\mathcal{D}$ is not as good as Gaussian likelihood. The SDNLL in (2.10) allows automatic and smooth transition between three different types of likelihood, Euclidean, Mahalanobis and Gaussian, according to the predicted effectiveness of each likelihood.

2.4.3 Computational considerations

We are now ready to discuss the computational steps for the previous procedures.

The first issue is how to represent the space $\mathcal{D}$ which is spanned by the centers of x-clusters in Eq.(2.1). These centers are vectors in $\mathcal{X}$, which typically has a very high dimensionality. The matrix weighted squared distance from a vector $x \in \mathcal{X}$ to each X-cluster with center c_i is defined by

$$d^2(x, c_i) = (x - c_i)^T W_i^{-1} (x - c_i) \tag{2.11}$$

which is the first term of Eq.(2.10).

We have two major issues to deal with. First, the SDSM W_i is very large if we represent it in $\mathcal{X}$ directly. Second, the sample covariance matrix, which we will be using to estimate matrix W_i, is not invertible before the number of samples has reached the high dimensionality of $\mathcal{X}$.

A way to address the first issue is to represent the discriminating space $\mathcal{D}$ by a basis of orthonormal vectors. Using the method explained in Appendix A, we keep an orthonormal basis of the linear manifold $\mathcal{D}$. To address the second issue, we represent the covariance matrix in the orthonormal basis for subspace $\mathcal{D}$ instead of $\mathcal{X}$. Since the dimensionality of D is at most $q-1$, the matrix W_i in the orthonormal basis is much smaller than that in the original space $\mathcal{X}$.

The computational steps are described as follows. Suppose that the dimensionality of space $\mathcal{X}$ is d. From q x-cluster centers in Eq.(2.1) in $\mathcal{X}$, use the GSO procedure in Appendix A to compute the $q-1$ orthonormal basis vectors $M = [\epsilon_1, \epsilon_2, ..., \epsilon_{q-1}]$, where each column ϵ is a unit basis vector, $i = 1, 2, \ldots, q-1$, and M is a $d \times (q-1)$ matrix. For each x-cluster center c_i, its coordinate vector in the orthonormal basis M is given by

$$e_i = M^T c_i.$$

Thus, each x-cluster c_i is represented by only a $(q-1)$-dimensional vector e_i. Given an unknown vector $x \in \mathcal{X}$, project it onto the basis $e = M^t x$. Then, the matrix-weighted squared distance in Eq.(2.11) is computed only in $(q-1)$-dimensional space using the basis M. The SDSM W_i for each x-cluster in then only a $(q-1) \times (q-1)$ square symmetric matrix, of which only $q(q-1)/2$ parameters need to be estimated. When $q = 6$, for example, this number is 15.

Given a column vector v represented in the discriminating subspace with an orthonormal basis whose vectors are the columns of matrix M, the representation of v in the original space $\mathcal{X}$ is $x = Mv$.

To compute the matrix weighted squared distance in Eq.(2.11), we should use a numerically efficient method. For example, we can use Cholesky factorization [25] (Sec. 4.2) which is for a positive definite matrix (which is symmetric). The Cholesky decomposition algorithm computes a lower triangular matrix L from W so that W is represented by $W = LL^T$. The procedure is relegated to Appendix B.

With the lower triangular matrix L, we first compute the difference vector from the input vector x and each x-cluster center c_i: $v = x - c_i$. The matrix weighted squared distance is given by

$$d^2(x, c_i) = v^T W_i^{-1} v = v^T (LL^T)^{-1} v = (L^{-1}v)^T (L^{-1}v). \qquad (2.12)$$

We solve for y in the linear equation $Ly = v$ and then $y = L^{-1}v$ and $d^2(x, c_i) = (L^{-1}v)^T(L^{-1}v) = ||y||^2$. Since L is a lower triangular matrix, the solution for y in $Ly = v$ is trivial since we simply use the backsubtitution method as described in [74] (page 42).

2.5 The experimental results

Several experiments were conducted using the proposed new HDR algorithm. First, we present the experimental results using synthetic data. Then we show the experimental results for real face images. In addition to these, the proposed algorithm was also applied to the data with manually extracted features from images.

2.5.1 Experiments using synthetic data

The motivation of using synthetic data for test is to investigate the behavior of different distance matrices and to examine the near optimality potential of our new algorithm with known distributions as a ground truth (but our algorithm does not know the distribution).

The first experiment used data set that has 3 clusters. The number of dimension is two. Each cluster was modeled by a Gaussian distribution. The centers of the clusters are at $(0, 0)$, $(5, 0)$, and $(0, 5)$, respectively. The covariance of the first cluster is an identity matrix. Those for the second and the third are,

$$\begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 4 & 0 \\ 0 & 2.25 \end{bmatrix},$$

respectively. We first show the effects of the number of samples. In Fig. 2.2, every cluster has 500 samples to estimate the required statistics for each measurement. The Bayesian decision boundaries were obtained by using the true means and covariance matrices for each cluster. To compute the Gaussian decision boundaries, we estimated the sample means and covariance for each cluster by using the training samples. As can be seen, the Gaussian NLL is very close to the Bayesian decision boundaries because we have sufficient number of samples to calculate those statistics. The difference between the Gaussian NLL and Mahalanobis NLL is that the covariance matrix for each cluster is replaced by the average of individual covariance matrices for the Mahalanobis NLL. Thus the decision boundaries for the Mahalanobis NLL are linear. The Euclidean NLL used identical matrices as the covariance matrices for each cluster and the boundaries are also linear. he decision boundary of SDNLL then is very close to that of Gaussian NLL which is an appropriate distance metric here because of the large number of available samples.

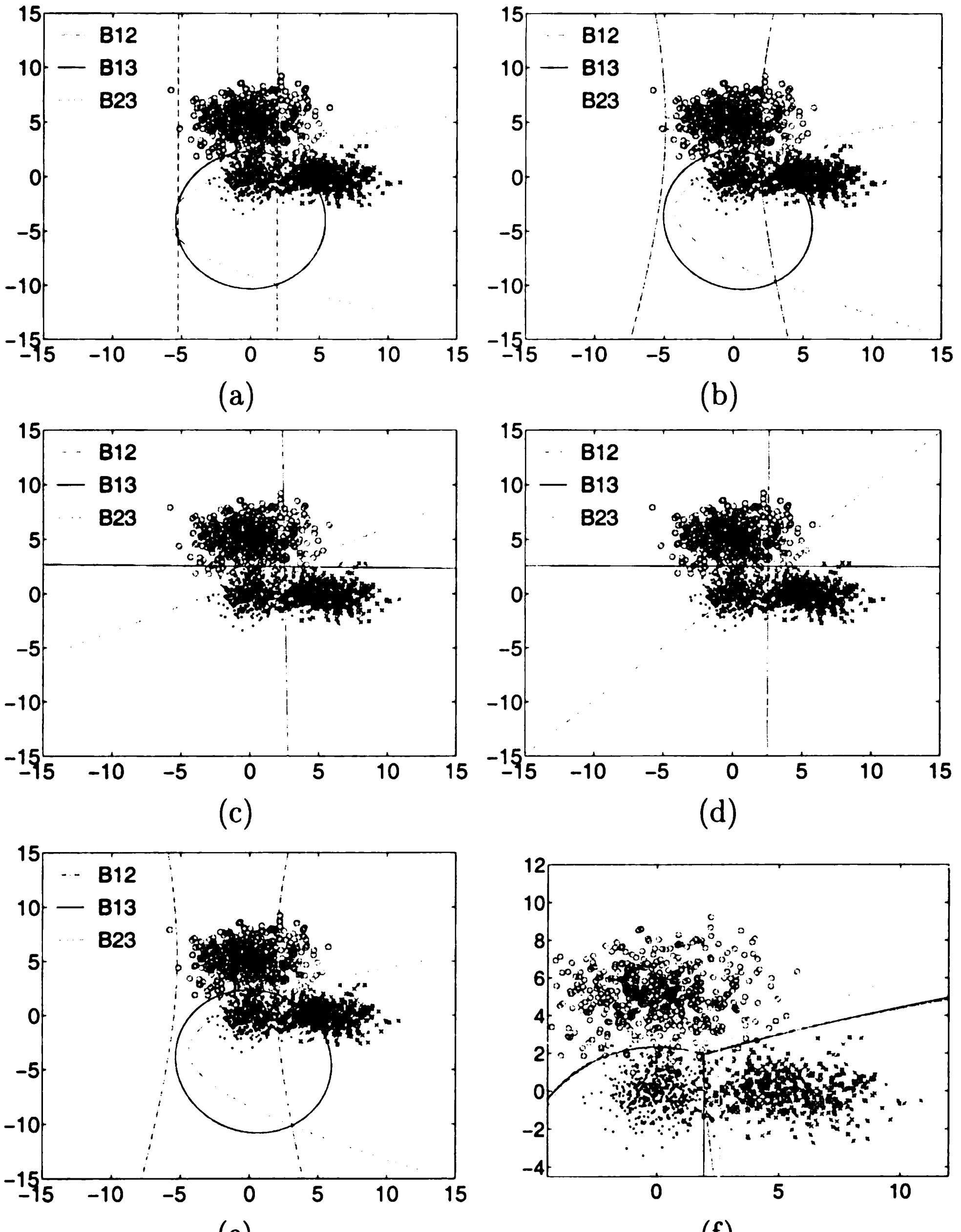


Figure 2.2: Decision boundaries estimated by 500 samples per class. Lines 'B12' mean decision boundaries between class 1 and class 2. Similarly, lines 'B13' represent boundaries between class 1 and class 3 and lines 'B23' represent boundaries between class 2 and class 3. Lines 'B' mean decision boundaries for Bayesian decision rule. This method uses the ground truth for distribution and thus is independent of samples. Lines 'E' are for Euclidean distance measured from a scale covariance matrix $\rho^2 I$. Lines 'G' are measured by Gaussian NLL using estimated full sample covariance matrices for all clusters. Lines 'M' are for Mahalanobis distance using a single estimated covariance matrix S_w. Lines 'L' use our SDNLL. (a) Bayesian decision boundaries. (b) Gaussian likelihood boundaries. (c) Mahalanobis likelihood boundaries. (d) Euclidean likelihood boundaries. (e) SDNLL boundaries. (f) The overall view.

There are 50 samples per class in Fig. 2.3. In this case, either Gaussian distance and Mahalanobis distance has reasonable decision boundaries. The decision boundary of SDNLL is between those of Gaussian distance and Mahalanobis distance. In Fig. 2.4, only five samples per class are used to estimate the decision boundaries. Since the number of samples is very small, the SDNLL is very much that of the Euclidean distance, which results in a reasonable boundary as shown. Fig. 2.5 shows the behaviors under unbalanced sample situation where the 3rd cluster receives much fewer samples than the first while the number of samples for the 2nd cluster is in-between. Fig. 2.5 indicated that the SDNLL distance metric behaves in the way we wanted.

For the large-sample case of Fig. 2.2, we would like to examine how close the error rates are to the best possible Bayesian error rates which use the ground truth about the distribution instead of samples. In the experiment, we used 500 samples per class to train. Testing is performed on the other 500 samples for each class. The bases derived from the our algorithm are $(0.89, -0.45)$ and $(-0.45, -0.89)$. The error rate is estimated on the basis we derived. The following table shows the classifications from (1) ground truth of distribution using Bayesian optimality (Bayesian-GT), (2) the parameters estimated from the training data based on Bayesian optimality (Bayesian-Sample), and (3) the proposed new algorithm (SDNLL). Table 2.2 shows that the classification errors. Of course, our method would not be able to be close to the Bayesian error rates if there are not enough samples per class.

The second experiment presented here is for 3-D where the discriminating space $\mathcal{D}$ is 2-D. There were 3 clusters, each being modeled by a Gaussian distribution with

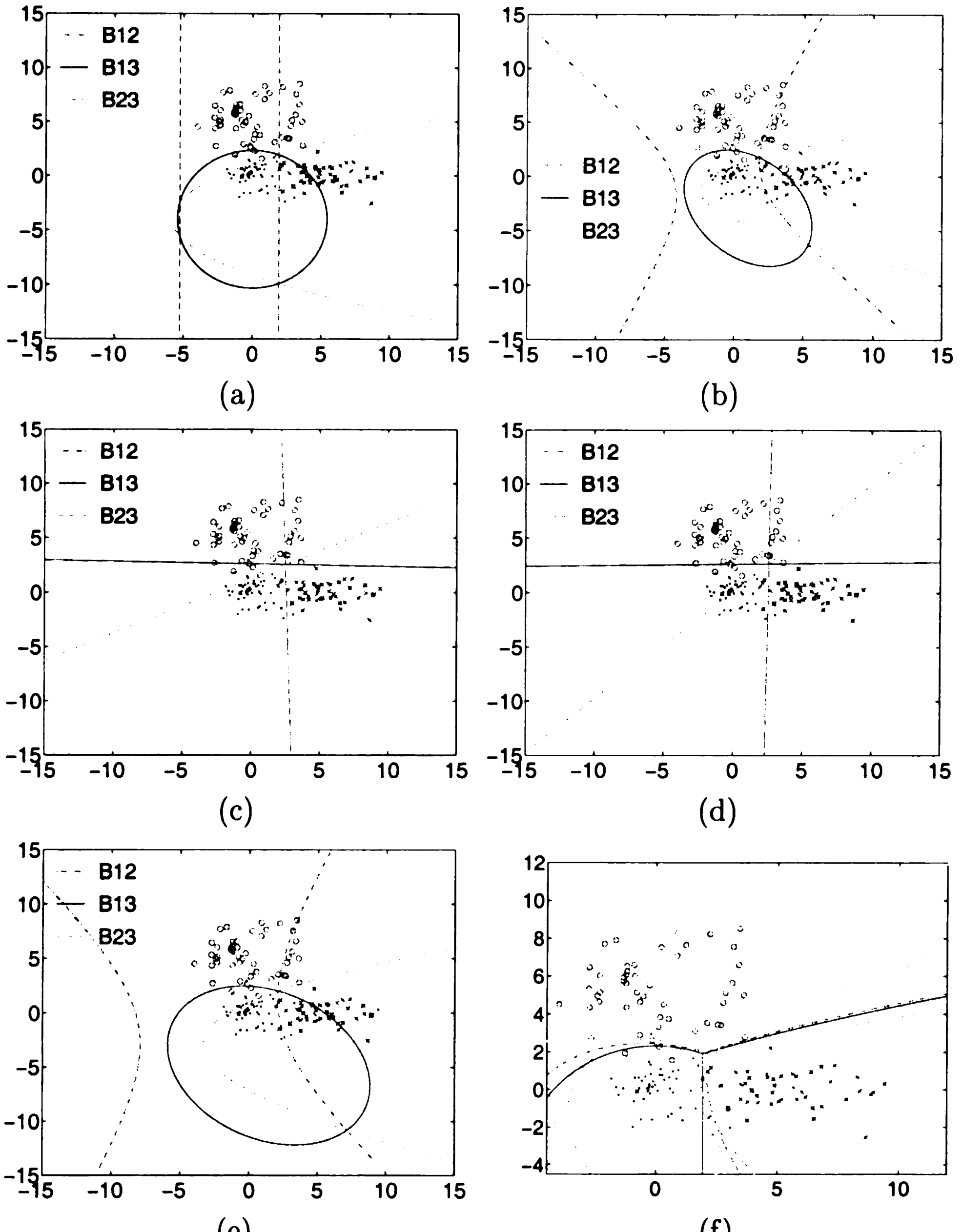


Figure 2.3: Decision boundaries estimated by 50 samples per class. (a) Bayesian decision boundaries. (b) Gaussian likelihood boundaries. (c) Mahalanobis likelihood boundaries. (d) Euclidean likelihood boundaries. (e) SDNLL boundaries. (f) The overall view.

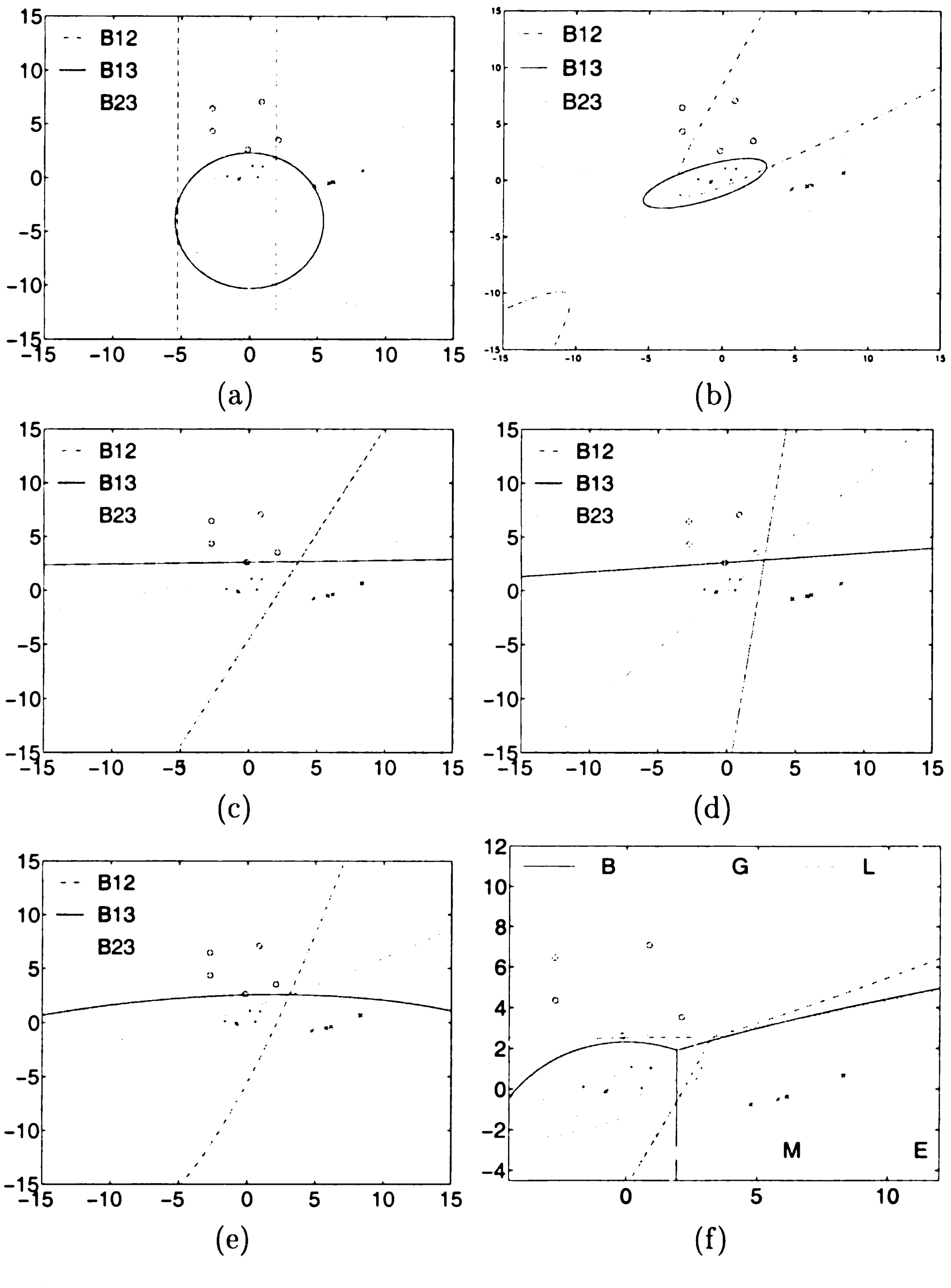


Figure 2.4: Decision boundaries estimated by 5 samples per class. (a) Bayesian decision boundaries. (b) Gaussian likelihood boundaries. (c) Mahalanobis likelihood boundaries. (d) Euclidean likelihood boundaries. (e) SDNLL boundaries. (f) The overall view.

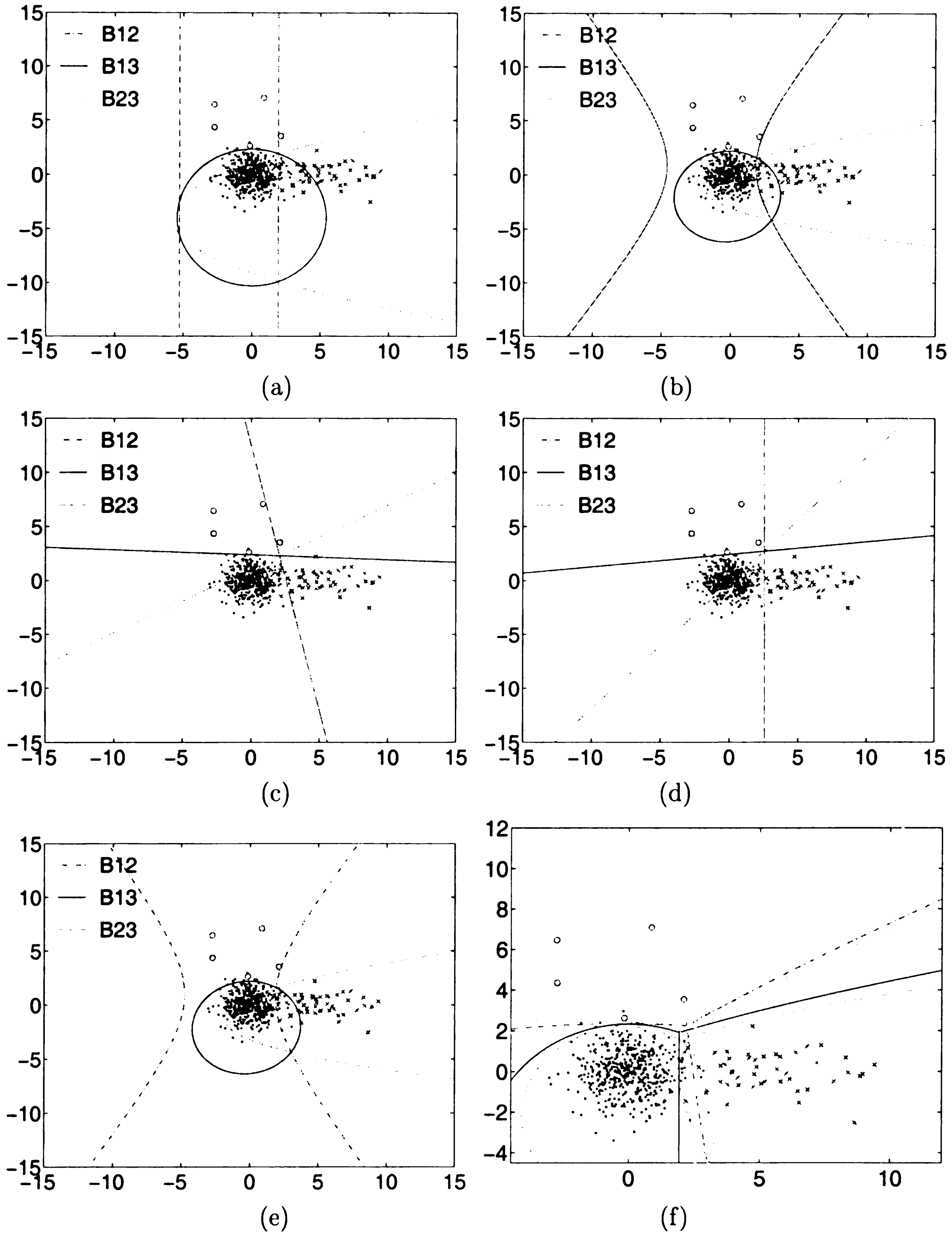


Figure 2.5: Decision boundaries for unbalanced samples distribution. Class one has 500 samples. Class two has 50 samples and class three has only 5 samples. (a) Bayesian decision boundaries. (b) Gaussian likelihood boundaries. (c) Mahalanobis likelihood boundaries. (d) Euclidean likelihood boundaries. (e) SDNLL boundaries. (f) The overall view.

Table 2.2: Optimality for 2-D Synthetic Data

Error rate	Bayesian-GT	Bayesian-Sample	SDNLL
class 1	0.0402	0.0443	0.0498
class 2	0.0660	0.0673	0.0682
class 3	0.0289	0.0298	0.0317

means, respectively, $(0,0,0)$, $(5,0,0)$, $(0,5,0)$ and covariance matrices

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 4 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 2.25 \end{bmatrix}.$$

There were 500 samples per class for training and testing, respectively. The training data is plotted in Fig. 2.6.

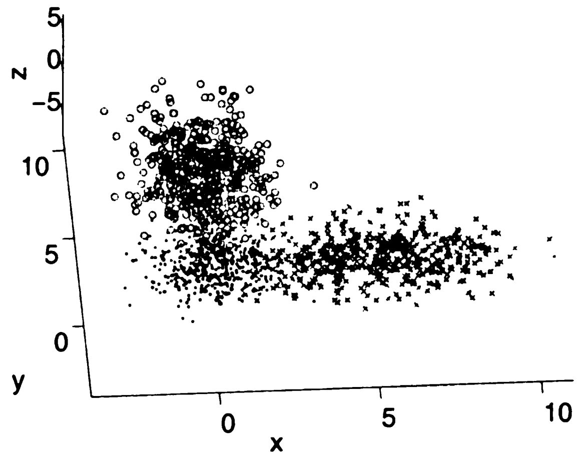


Figure 2.6: The data used in the second experiment.

We know that the basis is on x-y plane for the ground truth and we expect the deriving discriminating space $\mathcal{D}$ to be roughly so. The basis derived from the proposed algorithm is: $(0.89, -0.45, 0.001)$, $(-0.45, -0.89, -0.025)$ which are very close

Table 2.3: Optimality for 3-D Synthetic Data

		class 1	class 2	class 3
Bayesian-GT	average error	0.0678	0.0494	0.0232
	σ	0.0082	0.0096	0.0068
Bayesian-Sample	average error	0.0694	0.0498	0.023
	σ	0.011	0.0108	0.0074
SDNLL	average error	0.0511	0.058	0.027
	σ	0.011	0.0109	0.0076
SL12		0.411	0.85	0.89
SL13		0.00	0.00	0.01
SL23		0.00	0.00	0.01

to what we expect. The error rate is estimated on the basis we derived. Table 2.3 shows the classification results from Bayesian-GT, Bayesian-Sample rule and our new algorithm SDNLL, like the case of 2D. We used significance level ([50], pp. 268) to measure the error difference among each method. In Table 2.3, $SL12$ means the significance level for Bayesian-GT and Bayesian-Samples. Similarly, $SL13$ means the significance level for Bayesian-GT and SDNLL and $SL23$ means the significance level for Bayesian-Samples and SDNLL. The higher value of significance level indicates the more similar results between two methods. A total of 10000 trials have indicated the error differences between Bayesian-GT and Bayesian-Sample are statistically insignificant. Of course, the Bayesian-Sample algorithm cannot deal cases with small sample and unbalanced samples.

The third experiment used a high-dimensionality and six clusters. Each class was modeled by a Gaussian distribution in 100 dimensions with means $(0,\ldots,0)$, $(0,5,0,\ldots,0)$, $(0,0,5,0,\ldots,0)$, $(0,0,0,5,0,\ldots,0)$, $(0,0,0,0,5,0,\ldots,0)$,

Table 2.4: Optimality for 100-D Synthetic Data

		class 1	class 2	class 3	class 4	class 5	class 6
Bayesian-GT	avg. err.	0.112	0.025	0.025	0.026	0.025	0.026
	σ	0.014	0.0076	0.0067	0.0075	0.0086	0.0059
Bayesian-Sample	avg. err.	0.113	0.025	0.025	0.027	0.0255	0.026
	σ	0.016	0.0084	0.007	0.0079	0.0089	0.0064
SDNLL	avg. err.	0.090	0.0289	0.029	0.03	0.029	0.029
	σ	0.014	0.009	0.0081	0.0082	0.009	0.0068
SL12		0.69	0.71	0.718	0.7	0.69	0.75
SL13		0.00	0.015	0.0024	0.012	0.019	0.0089
SL23		0.00	0.046	0.0079	0.037	0.053	0.025

$(0,0,0,0,0,5,0,\ldots,0)$, respectively. The covariance matrix for class 0 is an identity matrix. The covariance matrix for the class $i, i > 0$ is an identity matrix except that the (i,i) element is equal to 2.25. There were 500 samples per class for training and the other 500 samples per class for testing. We expect the basis for discriminating subspace $\mathcal{D}$ is very much in the first six dimensions. The resulting basis is indeed very close to what we expect. Table 2.4 shows the error rates for the three types of classification rules under comparison. Since the first cluster overlaps with the other clusters, the errors listed in Table 2.4 in the first row and column are larger than other rows and columns. The error rates from the proposed algorithm are comparable with those that use direct Bayesian optimality.

2.5.2 Experiments using real image data

Since our primary interest is in images which have a high dimensionality, we applied the new algorithm to appearance-based face image retrieval tasks. The first two experiments used face images from Weizmann Institute and FERET face database, respectively. The third experiment was conducted on OCR images from digitized car license plates.

Face recognition on Weizmann dataset

The first experiment used face images from the Weizmann Institute at Israel. The image data base were constructed from 28 human subjects, each having thirty images with all possible combinations of two different expressions under three different lighting conditions with five different orientations. An example of the face images from one human subject is shown in Fig. 2.7.

We applied leave-one-out cross validation method to test this image data set. For each trial, a total of 840 images were used for the testing. Table 2.5 compares different appearance-based methods. For the principal component analysis (PCA), the number of eigenvectors used is determined by keeping 95% of the total sample variance. This gives 127 eigenvectors for PCA. A PCA tree is a binary classification tree where each node uses PCA.

Further, we compared the error rate of the proposed HDR algorithm with some major tree algorithms. CART[2] and C5.0 are among the best known classification trees. However, like most other decision trees, they are univariate trees in that each internal node used only one input component to partition the samples. This means that the partition of samples is done using hyperplanes that are orthogonal to one

[2]We have experimented the same data set using CART implemented by OC1. The performance is far worse than those reported in the Table 2.5. See CART for FERET set in Table 2.6.

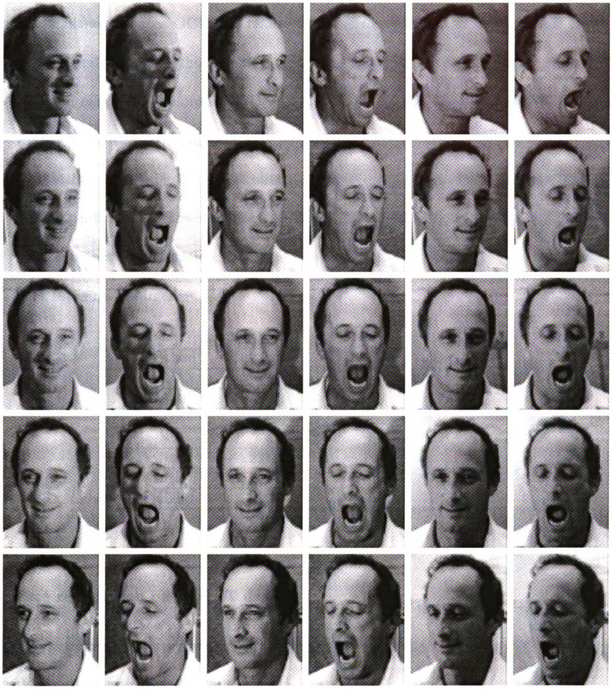


Figure 2.7: Face images from Weizmann Institute, all the combination of 3 lightings, 2 expressions, and 5 orientations.

axis. We do not expect this type of tree can work well in a high dimensional space for highly correlated multimedia data like images. Thus, we also tested a more recent multivariate tree OC1. We realize that these trees were not designed for high dimensional spaces like those from images. We also tested the corresponding versions by performing PCA before using CART, C5.0, and OC1 and call them CART with PCA, C5.0 with PCA, and OC1 with PCA, respectively.

As shown in Table 2.5, LDA shares the best performance with our new HDR method in this test. However, the new HDR method is faster than LDA and has a more compact representation. The speed difference will be more significant when the data set is much larger.

Table 2.5: The performance for Weizmann face dataset

Method	Error rate	Avg. testing time (msec)
PCA	0.95%	290
PCA tree	1.79%	76
LDA	0.00%	110
NN	1.31%	370
C5.0 with PCA	27.98%	197
OC1 with PCA	37.62%	203
HDR	0.00%	82

Face recognition on FERET dataset

We performed two experiments using the FERET face dataset [71] [72] [81]. We used the frontal views from the data set. There are 457 persons with frontal views, thirty three of whom with four frontal images, one with six images, and the remaining 423 persons having only two images each.

A face normalization program was used to translate, scale, and rotate each face image into a canonical image of 88 rows and 64 columns [2] so that eyes are located at prespecified positions as shown in Fig. 2.8. To reduce the effect of background and non-facial areas, image pixels are weighted by a function of the radical distance from the image center. Further, the image intensity is masked by a linear function so that the minimum and maximum values of each image are 0 and 255, respectively. Fig. 2.8 shows the effect of such a series of transformations.

Thirty four human subjects were involved in the first experiment for the FERET data set. Each person had three face images for the purpose of training. The other face image was used for testing. We compare different options of the proposed algorithms. First, we used Euclidean distance in the discriminating subspace instead of SDNLL distance. With different choices of number of x-clusters (q), we found that the performance does not significantly increase with the increase of q. Then we used SDNLL distance and the result is shown in Fig. 2.9. From the figure, we found that the best q ($q = 18$ and beyond) resulted in 100% recognition rate.

To give an intuitive display about what are the centers of the x-clusters, we show in Fig. 2.10 the mean face images at the root with $q = 15$. The dimensionality of the discriminating subspace is then 14.

Fig. 2.11 (a) shows the depth of the HDR trees with different q's and distance metrics. All the options resulted in a similar tree height. The tree constructed using Euclidean distance has the most shallow depth. Figs. 2.11 (b) and (c) give the nodes counts at every level of the trees for $q = 2$. It is worth noting that the structure of the trees affects the speed of the algorithm. As shown in Fig. 2.12 (b), a deeper tree results in a faster tree retrieval because it works on a lower dimensional space

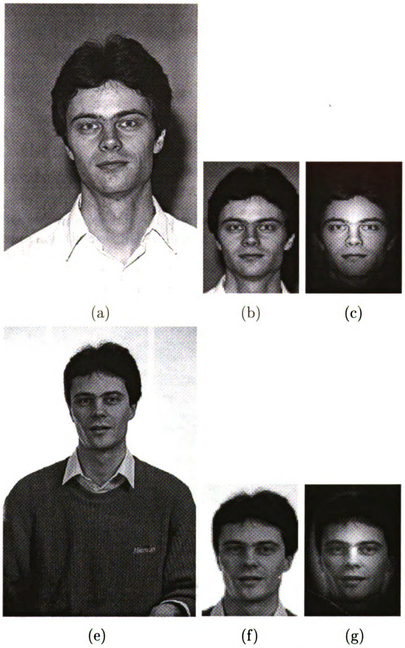


Figure 2.8: The demonstration of the image normalization process. (a) and (e): The original image from the FERET data set. (b) and (f): The normalized image. (c) and (g): The masked image.

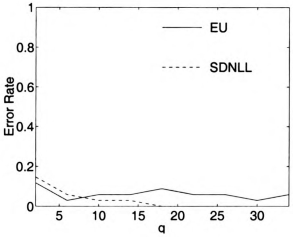


Figure 2.9: The plot of error rate vs. number of x-clusters for FERET face test 1 using Euclidean distance and the new SDNLL distance.

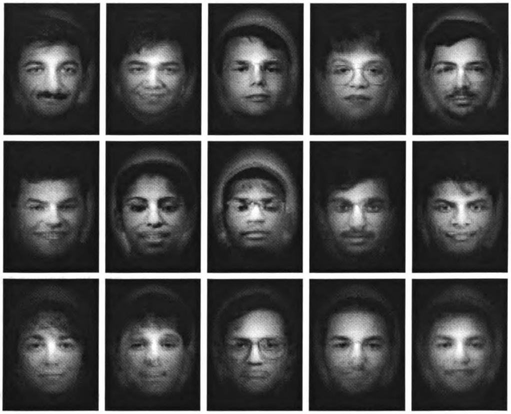


Figure 2.10: mean images in the root.

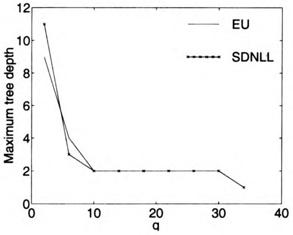
Table 2.6: The performance comparison of decision trees for the FERET test 1

Method	Error rate		Time (sec)	
	Training	Testing	Training	Testing
CART	10%	53%	2108.00	0.029
C5.0	1%	41%	21.00	0.030
OC1	6%	56%	2206.00	0.047
CART with PCA	11%	53%	10.89	0.047
C5.0 with PCA	6%	41%	9.29	0.047
OC1 with PCA	5%	41%	8.89	0.046
HDR	0%	0%	12.25	0.027

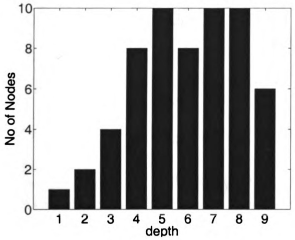
at each level. Fig. 2.12 indicates that the SDNLL distance metric does not require significantly more time to compute.

A summary of performance comparison with some existing major tree classifiers is listed in Table 2.6. Notice that the training time is measured for the total time to train the corresponding system. The testing time is the average time per query. To make a fair comparison, the computation time for PCA is included in C5.0 with PCA, OC1 with PCA, and CART with PCA. As shown, none of the existing decision trees can deal with FERET set acceptably well, not even the versions that use PCA as a preprocessing step.

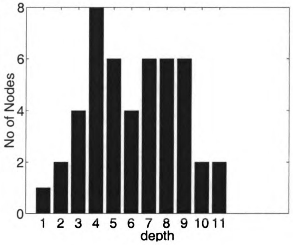
The second experiment for the FERET data set used all the available data (Experiment 1 did not use all the available FERET data because capacity-limitation of these existing decision trees compared). As described before, most of subjects have only two views. We used leave-one-out cross validation method. For each trial, one image was selected for each person for testing and the remaining images were used for training. The number of samples for each cluster thus is not equal. We use a similar



(a)



(b)



(c)

Figure 2.11: The tree structures of FERET face test 1. (a) The plot of depth of the tree vs q for different distance options. (b) and (c): The plots of tree structures for different options with $q = 2$ for EU and SDNLL, respectively. EU: Euclidean Distance.

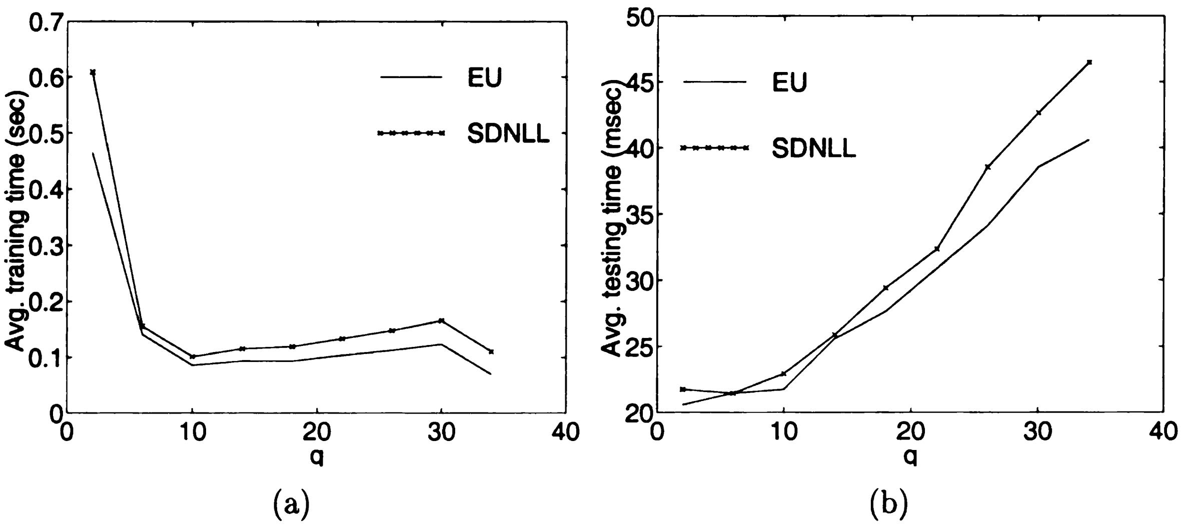


Figure 2.12: The timing data of FERET face test 1. (a) The plot of the average training time vs q. (b) The plot of the average testing time vs q. EU: Euclidean Distance.

analysis to experiment one. The results are shown in Figs. 2.13, 2.14, and 2.15.

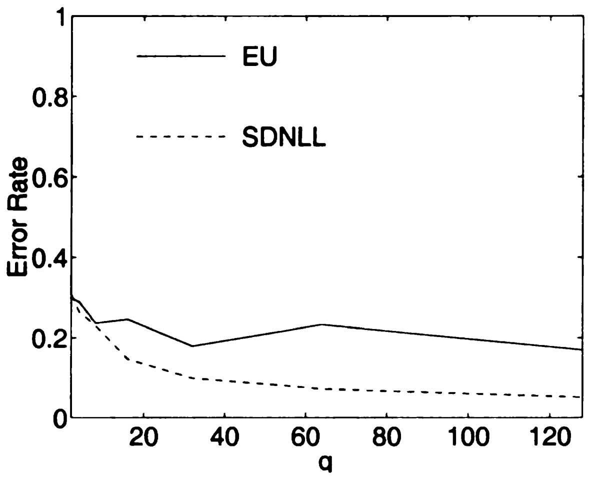


Figure 2.13: The performance plots of FERET test 2. The plots of error rate vs. number of x-clusters. EU: Euclidean Distance.

Optical character recognition from car license plates

We also applied our method on the OCR problem. Images of characters extracted from car license plates were used in the experiment. Fig 2.16 shows the examples of the character 'A.' A total of 23 different characters appeared in the plates. Totally 222

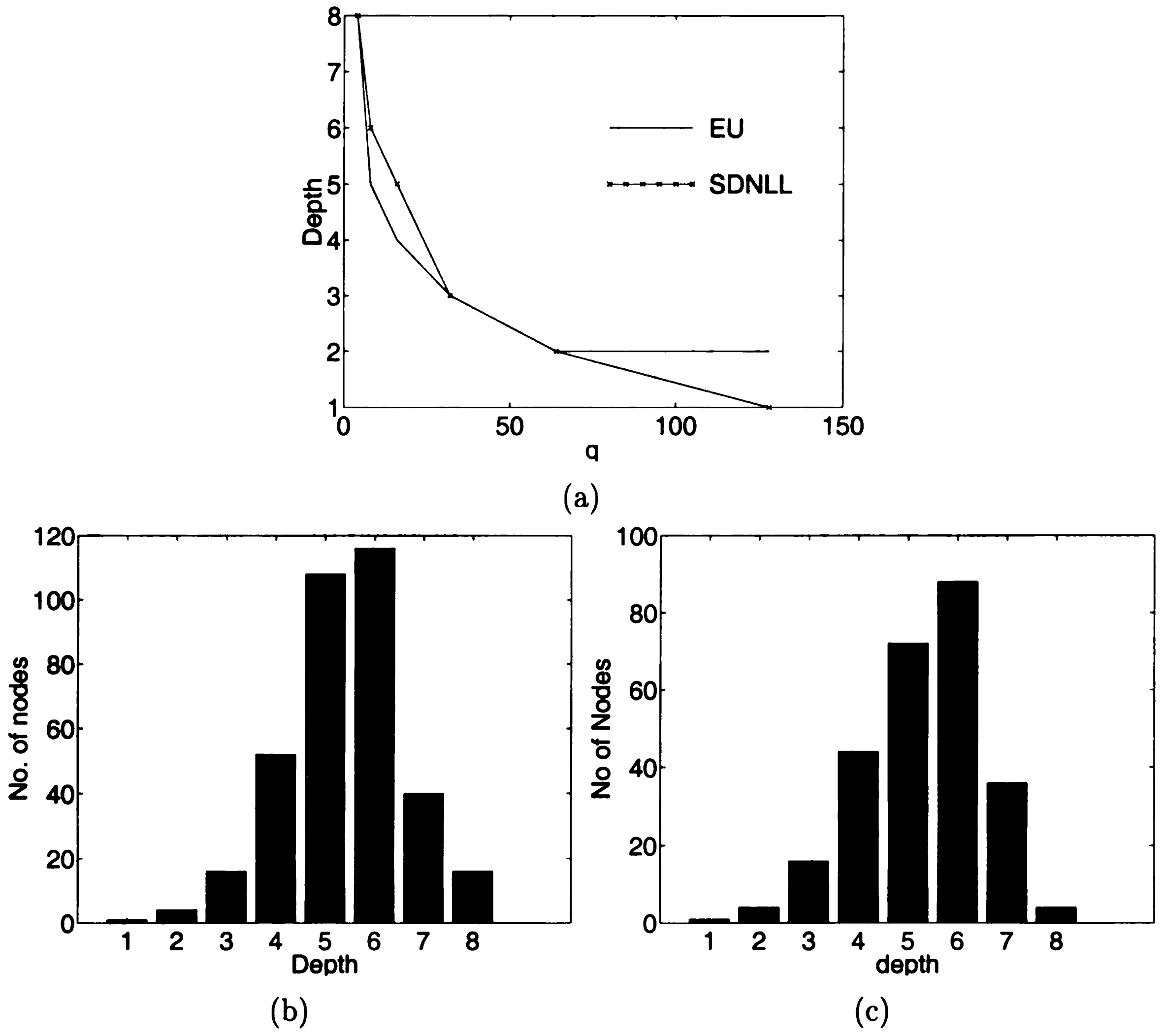


Figure 2.14: The tree structures of FERET face test 2. (a) the plot of depth of the tree vs q for different options. (b) and (c) are the plots of tree structures for Euclidean Distance and SDNLL distance with $q = 2$, respectively.

Table 2.7: The performance for FERET face dataset II

Method	Error rate	Avg. testing time (msec)
PCA	4.38%	203
PCA tree	7.66%	34
LDA	Segmentation fault	Nil
NN	3.50%	270
HDR (q=64)	8.32%	132
HDR (q=2)	12.91%	37

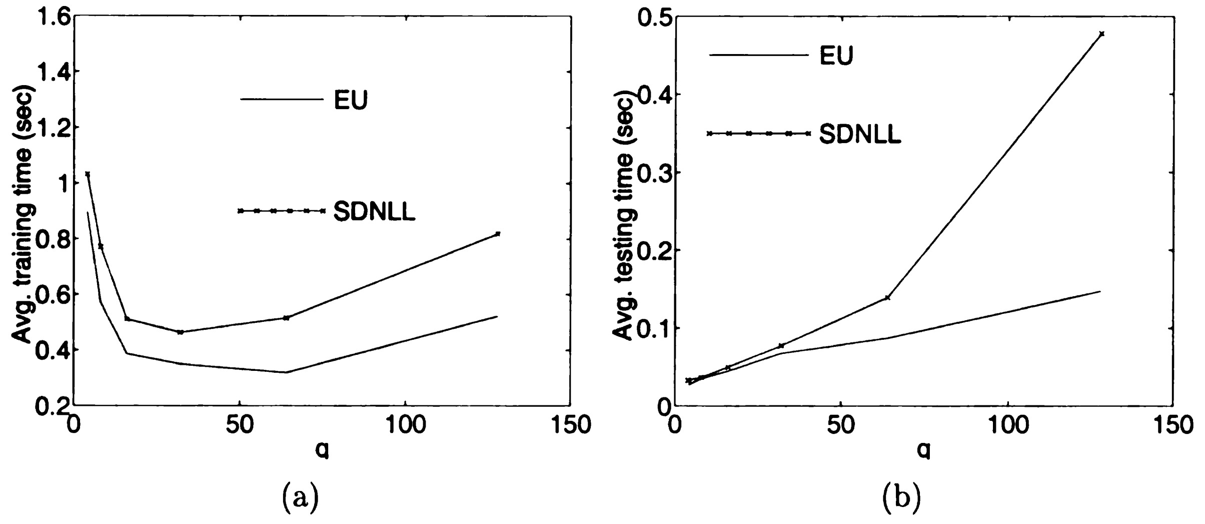


Figure 2.15: The timing data of FERET face test 2. (a) The plot of the average training time vs q. (b) The plot of the average testing time vs q. EU: Euclidean Distance.

Table 2.8: The performance for character images from car license plates

Method	Error rate	Avg.testing time (msec)
PCA	4.28%	2.0
PCA tree	4.53%	1.5
LDA	0.50%	1.6
NN	3.78%	3.0
C5.0 with PCA	17.50	1.7
OC1 with PCA	17.04%	1.9
HDR	0.00%	1.7

images were used for training and 397 images were used for testing. We summarized the results in Table 2.8. The new HDR method has the best accuracy and the PCA tree method is the fastest algorithm.

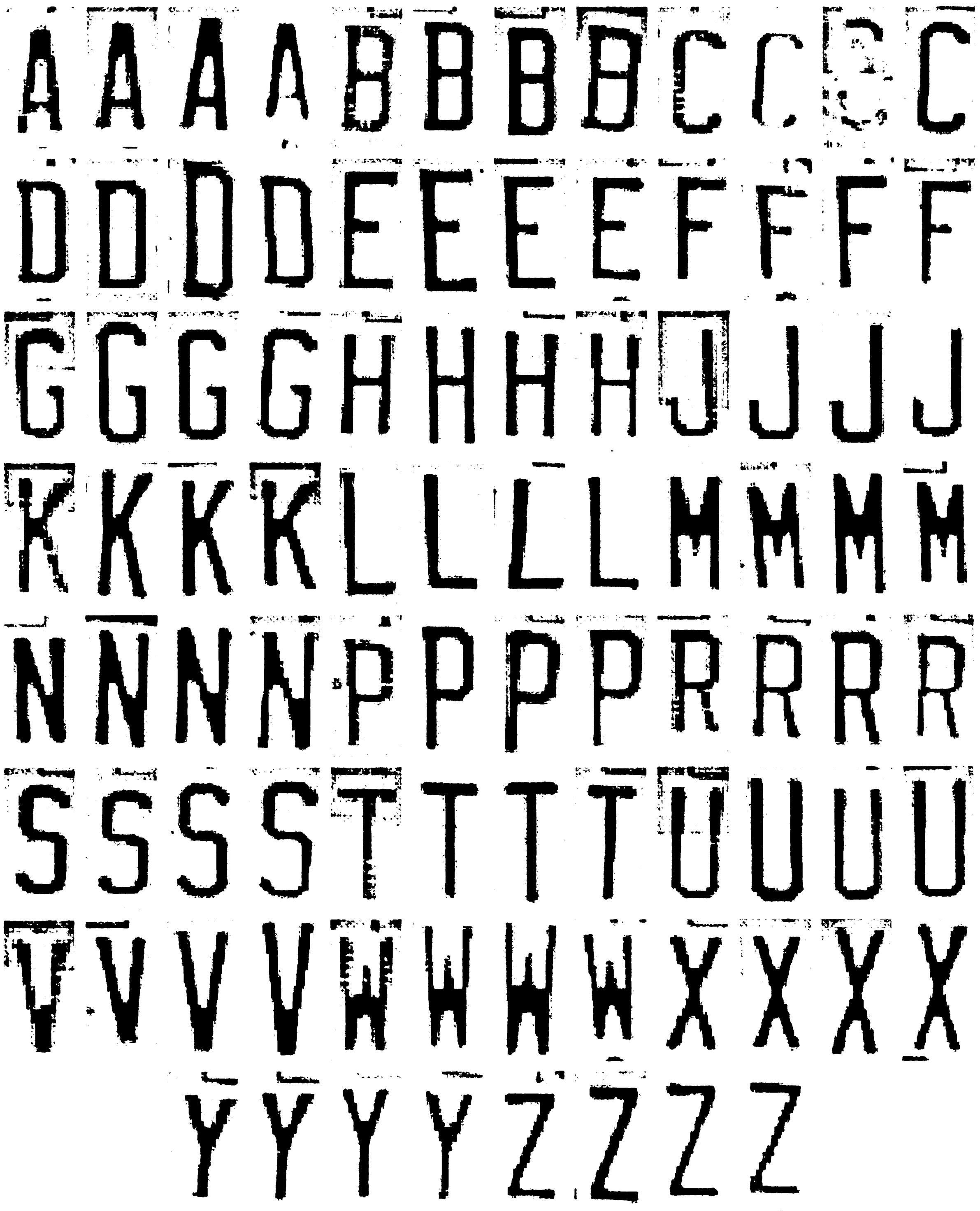


Figure 2.16: Letter images from car license plates.

2.5.3 Experiments using data with manually extracted features

We further investigated how our HDR algorithm performs on lower dimensional real data, such as those publically available data sets that use human defined features. Thus, we also tested our algorithm on some data available publically. These data used manually selected features. We reported comparison results for three data sets.

1. Letter image recognition data: There are 26 classes which corresponding to 26 capital letters. Each sample has 16 numeric features. 15000 samples were used for training and 5000 samples were used for testing.

2. Satellite image dataset: There are six decision classes representing different types of soils from satellite image. Each sample has 36 attributes. Training set includes 4435 samples and testing set includes 2000 samples.

3. shuttle dataset: There are seven classes. The number of attributes is 7. 43500 samples were used for training and 14500 samples were used for testing.

We inserted the performance of the new HDR tree algorithm as well as the IHDR tree algorithm described in Chapter 2 to the results which were published in the StatLog project [57] as shown in Tables 2.9, 2.10, and 2.11. For these lower dimensional data sets, the performance of HDR tree algorithm is comparable with other best existing ones.

Table 2.9: Test results on letter image recognition data

Algorithm	Error rate		Time (sec)	
	training	testing	training	testing
Alloc80	0.065	0.064	39575	?
KNN	0	0.068	15	2135
* HDR tree	0	0.070	212.7	30
IHDR tree	0	0.072	1150	41
LVQ	0.057	0.079	1487	48
QuaDisc	0.101	0.113	3736	1223
Cn2	0.021	0.115	40458	52
BayTree	0.015	0.124	276	7
NewId	0	0.128	1056	2
IndCart	0.010	0.130	1098	1020
C4.5	0.042	0.132	309	292
Dipol92	0.167	0.176	1303	80
Radial	0.220	0.233	?	?
LogDisc	0.234	0.234	5062	39
Ac2	0	0.245	2529	92
Castle	0.237	0.245	9455	2933
Kohonen	0.218	0.252	?	?
Cal5	0.158	0.253	1033	8
Smart	0.287	0.295	400919	184
Discrim	0.297	0.302	326	84
BackProp	0.323	0.327	277445	22
Bayes	0.516	0.529	75	18
Itrule	0.585	0.594	22325	69
Default	0.955	0.960	?	?
Cascade	1.0			
Cart	1.000			

Table 2.10: Test results on satellite image dataset

Algorithm	Error rate		Time (sec)	
	training	testing	training	testing
KNN	0.089	0.094	2105	944
LVQ	0.048	0.105	1273	44
* HDR tree	0	0.108	2.36	0.41
Dipol92	0.051	0.111	746	111
Radial	0.111	0.121	564	74
Alloc80	0.036	0.132	63840	28757
IHDR tree	0	0.135	220	0.85
IndCart	0.023	0.138	2109	9
Cart	0.079	0.138	330	14
BackProp	0.112	0.139	72495	53
BayTree	0.020	0.147	248	10
NewId	0.067	0.150	226	53
Cn2	0.010	0.150	1664	36
C4.5	0.040	0.150	434	1
Cal5	0.125	0.151	764	7
QuaDisc	0.106	0.155	157	53
Ac2	?	0.157	8244	17403
Smart	0.123	0.159	27376	11
LogDisc	0.119	0.163	4414	41
Cascade	0.112	0.163	7180	1
Discrim	0.149	0.171	68	12
Kohonen	0.101	0.179	12627	129
Castle	0.186	0.194	75	80
Bayes	0.308	0.287	75	17
Default	0.758	0.769		
Itrule	?	100.00		

Table 2.11: Test results shuttle dataset

	Error	Rate	TIME	
Algorithm	Train	Test	Train	Test
** HDR tree	0	0.0021	41.9	3.76
NewId	0	0.01	6180	?
BayTree	0	0.02	240	17
Cn2	0	0.03	11160	?
Cal5	0.03	0.03	313	10
IHDR tree	0	0.04	820.7	6.55
Cart	0.04	0.08	79	2
IndCart	0.04	0.09	1152	16
C4.5	0.04	0.10	13742	11
Ac2	0	0.32	2553	2271
Itrule	?	0.41	91970	?
BackProp	4.50	0.43	5174	21
KNN	0.39	0.44	32531	10482
LVQ	0.40	0.44	2813	84
Dipol92	0.44	0.48	2068	176
Smart	0.61	0.59	110010	93
Alloc80	0.95	0.83	55215	18333
Radial	1.60	1.40	?	?
Castle	3.70	3.80	461	150
LogDisc	3.94	73.83	6946	106
Bayes	4.60	4.50	1030	22
Discrim	4.98	4.83	508	102
QuaDisc	6.35	6.72	709	177
Default	21.59	20.84		
Cascade	?	100		
Kohonen	?	100		

Chapter 3

Hierarchical Discriminant Regression Tree: Incremental Learning Mode

3.1 introduction

As far as we know, there is no published incremental statistical method for constructing a regression tree for high dimensional input space based on discriminant analysis. By high dimensional space we mean that the dimensionality is above a few thousands and the number of samples can be smaller than the dimensionality [1].

This high dimensional issue becomes increasingly important with increased use of high dimensional digital multimedia data such as images and video where each pixel

[1]When the number of samples is smaller than the input dimensionality (i.e., the number of features), Brieman et al. [11] and Murthy [40] called the situation data underfits the concept and thus disregard the situation.

value is a component of the input vector [2]. These applications present a high degree of correlation among components of high-dimensional input, not matched by typical lower-dimensional human-prepared feature data. None of CART, C5.0, OC1 or other published tree classifiers that we know was designed for this high-dimensional, highly correlated input situation. SHOSLIF tree is for high input dimensionality and it has an incremental version [97], but it uses PCA only for splits. It is technically challenging to incrementally construct a classification or regression tree that uses discriminant analysis, due to the complex nature of the problem involved. Why is discriminant analysis important? Discriminant analysis uses information of the output space in addition to the information in the input space to compute the splits. PCA uses only information in the input space and cannot use information in the output space. Consequently, variations in the input space that is totally useless for output (e.g., pure noise components) will also be captured by PCA. On the other hand, various neural networks are incremental in training. They tend to build a network with some discriminating power. However, since neural networks typically do not use a statistical model, they suffer from local minima problem and thus give poorer performance as we will show later in this chapter.

We present an incremental way of constructing a regression tree that uses discriminant analysis. Further, we deal with the unbalanced sample problem in that some regions of input space may have a very large number of samples while other regions have only very sparse samples. A sample-size dependent likelihood measure

[2]This corresponds to a now well-accepted and highly successful approach called appearance-based approach, with which the human system designer does not define features at all but rather applies statistical methods directly to high-dimensional, preprocessed image frame [42] [92] [61] [88].

is proposed to make suboptimal decision for different sample sizes, which is also very critical for an incremental algorithm which receive training data incrementally. We also require real-time speed of the regression system which is essential for many interactive applications and thus a hierarchical data pruning structure such as a tree is a must. We present experimental result to demonstrate the performance of the new technique and compare it with some major published methods.

3.2 Incremental Hierarchical Discriminant Regression

The algorithm incrementally builds an IHDR tree from a sequence of training samples. The deeper a node is in the tree, the smaller the variances of its x-clusters are. When the number of samples in a node is too small to give a good estimate of the statistics of q x-clusters, this node is a leaf node. The following is the outline of the incremental algorithm for tree building (also tree retrieval when y is not given).

Procedure 4 `Update-node`*: Given a node N and (x, y) where y is either given or not given, update the node N using (x, y) recursively. Output: top matched terminal nodes. The parameters include: k which specifies the upper bound in the width of parallel tree search; δ_x the sensitivity of the IHDR tree in $\mathcal{X}$ space as a threshold to further explore a branch; and c representing if a node is on the central search path. Each returned node has a flag c. If $c = 1$, the node is a central cluster and $c = 0$ otherwise.*

1. *Find the top matched x-cluster in the following way. If* $c = 0$ *skip to step (2). If* y *is given, do (a) and (b); otherwise do (b).*
   (a) *Update the mean of the y-cluster nearest* y *in Euclidean distance by using amnesic averages. Update the mean and the covariance matrix of the x-cluster corresponding to the y-cluster by using amnesic average.*
   (b) *Find the x-cluster nearest* x *according to the probability-based distances. The central x-cluster is this x-cluster. Update the central x-cluster if it has not been updated in (a). Mark this central x-cluster as active.*
2. *For all the x-clusters of the node* N*, compute the probability-based distances for* x *to belong to each x-cluster.*
3. *Rank the distances in increasing order.*
4. *In addition to the central x-cluster, choose peripheral x-clusters according to increasing distances until the distance is larger than* δ_x *or a total of* k *x-clusters have been chosen.*
5. *Return the chosen x-clusters as active clusters.*

From the above procedure, we can observe the following points. (a) When y is given, the corresponding x-cluster is updated, although this x-cluster is not necessarily the one on the central path from which the tree is explored. Thus, we may update two x-clusters, one corresponding to the given y, the other being the one used for tree exploration. The update for the former is an attempt to pull it to the right location. The update for the latter is an attempt to record the fact that the central x-cluster has hit this x-cluster once. (b) No matter y is given or not, the x-cluster along the central path is always updated. (c) Only the x-clusters along the central path are updated, other peripheral x-clusters are not. We would like to avoid, as much as possible, storing the same sample in different brother nodes.

Procedure 5 `Update-tree`: *Given the root of the tree and sample (x, y), update the tree using (x, y). If y is not given, estimate y and the corresponding confidence. The parameters include: k which specifies the upper bound in the width of parallel tree search.*

1. *From the root of the tree, update the node by calling* `Update-node` *using (x,y).*
2. *For every active cluster received, check if it points to a child node. If it does, mark it inactive and explore the child node by calling* `Update-node`. *At most q^2 active x-clusters can be returned this way if each node has at most q children.*
3. *The new central x-cluster is marked as active.*
4. *Mark additional active x-clusters according to the smallest probability-based distance d, up to k total if there are that many x-clusters with $d \leq \delta_x$.*
5. *Do the above steps 2 through 4 recursively until all the resulting active x-clusters are all terminal.*
6. *Each leaf node keeps samples (x_i, y_i) that belong to it. The output is y_i if x_i is the nearest neighbor among these samples.*
7. *If the current situation satisfies the spawn rule, i.e. the number of samples exceeds the number required for estimating statistics in new child, the top-matched*

x-cluster in the leaf node along the central path spawns a child which has q new x-clusters. All the internal nodes are fixed in that their clusters do not further update using future samples so that their children do not get temporally inconsistent assignment of samples.

The above incrementally constructed tree gives a coarse-to-fine probability model. If we use Gaussian distribution to model each x-cluster, this is a *hierarchical version* of the well-known mixture-of-Gaussian distribution models: the deeper the tree is, the more Gaussians are used and the finer are these Gaussians. At shallow levels, the sample distribution is approximated by a mixture of large Gaussians (with large variances). At deep levels, the sample distribution is approximated by a mixture of many small Gaussians (with small variances). The multiple search paths guided by probability allow a sample x that falls in-between two or more Gaussians at each shallow level to explore the tree branches that contain its neighboring x-clusters. Those x-clusters to which the sample (x, y) has little chance to belong are excluded for further exploration. This results in the well-known logarithmic time complex for tree retrieval: $O(\log m)$ where m is the number of leaf nodes in the tree, assuming that the number of samples in each leaf node is bounded above by a constant.

3.3 Amnesic average

In incremental learning, the initial centers of each state clusters are largely determined by early input data. When more data are available, these centers move to more appropriate locations. If these new locations of the cluster centers are used to judge the boundary of each cluster, the initial input data were typically incorrectly classified. In other words, the center of each cluster contains some earlier data that do not belong to this cluster. To reduce the effect of these earlier data, the amnesic average can be used to compute the center of each cluster. The amnesic average can also track dynamic change of the input environment better than a conventional average.

The average of n input data $x_1, x_2, ..., x_n$ is given by

$$\bar{x}^{(n)} = \frac{1}{n}\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} \frac{1}{n} x_i. \tag{3.1}$$

In the above expression, every x_i is multiplied by a weight $1/n$ and the product is summed up together. Therefore, each x_i receives the same weight $1/n$. This is called an equally weighted average. If x_i arrives incrementally and we need to compute the average for all the inputs received so far, it is more efficient to recursively compute the current average based on the previous average:

$$\bar{x}^{(n+1)} = \frac{n\bar{x}^{(n)} + x_{n+1}}{n+1} = \frac{n}{n+1}\bar{x}^{(n)} + \frac{1}{n+1}x_{n+1}. \tag{3.2}$$

In other words, the previous average $\bar{x}^{(n)}$ gets a weight $n/(n+1)$ and the new input x_{n+1} gets a weight $1/(n+1)$. These two weights sum to one. The recursive equation Eq. (3.2) gives an equally weighted average. In amnesic average, the new input gets more weight as given in the following expression:

$$\bar{x}^{(n+1)} = \frac{n-l}{n+1}\bar{x}^{(n)} + \frac{1+l}{n+1}x_{n+1} \tag{3.3}$$

,where l is a parameter.

The amnesic average can also be applied to the recursive computation of a covariance matrix Γ_x from incrementally arriving samples: $x_1, x_2, ..., x_n, ...$ where x_i is a column vector for $i = 1, 2, ...$ The unbiased estimate of the covariance matrix from these n samples $x_1, x_2, ..., x_n$ is given in a batch form as

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T \tag{3.4}$$

with $n > 1$, where $\bar{x}$ is the mean vector of the n samples. Using the amnesic average, $\bar{x}^{(n+1)}$, up to the $(n+1)$-th sample, we can compute the amnesic covariance matrix up to the $(n+1)$-th sample as

$$\Gamma_x^{(n+1)} = \frac{n-1-l}{n}\Gamma_x^{(n)} + \frac{1+l}{n}(x_{n+1} - \bar{x}^{(n+1)})(x_{n+1} - \bar{x}^{(n+1)})^T \tag{3.5}$$

for $n > l+1$. When $n \leq l+1$, we may use the batch version as in expression (3.4). Even with a single sample x_1, the corresponding covariance matrix should not be estimated as a zero vector, since x_1 is never exact if it is measured from a physical event. For example, the initial variance matrix $\Gamma_x^{(1)}$ can be estimated as $\sigma^2 I$, where σ^2 is the expected digitization noise in each component and I is the identity matrix of the appropriate dimensionality.

3.4 The experimental results

The proposed algorithm was tested with the data which was used in the batch mode. First, we present the experimental results using synthetic data. Then we show the the power of the method use real face images as high dimensional input vectors for classification. For the regression problem, we demonstrated the performance of our algorithm for autonomous navigation where input is current image and output is the required steering signal.

3.4.1 Experiments using synthetic data

The first experiment used the same 3-clusters data set as used in the batch-mode testing. The training set is shown in Fig. 2.2 (c) with 500 samples per class. The bases derived from the IHDR tree algorithm are $(-0.88, 0.44), (0.45, 0.92)$. The error rate is estimated on the basis we derived. Table 3.1 shows the classification errors from (1) ground truth of distribution using Bayesian optimality (Bayesian-GT), (2) the parameters estimated from the training data based on Bayesian optimality (Bayesian-Sample), and (3) the proposed new algorithm (IHDR).

Table 3.1: Error rates for 2-D Synthetic Data

	Bayesian-GT	Bayesian-Sample	IHDR
class 1	7%	7.2%	7.6%
class 2	6.8%	8.2%	5.6%
class 3	1.6%	2.0%	2.0%

The second experiment is for 3-D. The discriminating space $\mathcal{D}$ is 2D – x-y plane.

There were 3 clusters, each being modeled by a Gaussian distribution with means, respectively, $(0,0,0)$, $(5,0,0)$, $(0,5,0)$ and covariance matrices

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 4 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 2.25 \end{bmatrix}$$

. There are 500 samples per class for training and testing, respectively. The basis derived from the proposed algorithm is: $(0.43, 0.92, -0.021)$, $(-0.89, 0.44, 0.001)$ which are very close to x-y plane as we expected. The error rate is estimated on the basis we derived. Table 3.2 shows the classification from ground truth Bayesian rule, sample-based Bayesian rule and our new algorithm, like the case of 2D.

Table 3.2: Error rates for 3-D Synthetic Data

	Bayesian-GT	Bayesian-Sample	IHDR
class 1	7%	7.2%	4%
class 2	6.8%	8.2%	4.6%
class 3	1.6%	2.0%	4.8%

The third experiment used a high-dimensionality and six clusters. Each class was modeled by a Gaussian distribution in 100 dimensions with means $(0,\ldots,0)$, $(0,5,0,\ldots,0)$, $(0,0,5,0,\ldots,0)$, $(0,0,0,5,0,\ldots,0)$, $(0,0,0,0,5,0,\ldots,0)$, $(0,0,0,0,0,5,0,\ldots,0)$, respectively. The covariance matrix for class 0 is an identity matrix. The covariance matrix for the class $i, i > 0$ is an identity matrix except that the (i,i) element is equal to 2.25. There were 500 samples per class for training and the other 500 samples per class for testing. We expect the basis for discriminating subspace $\mathcal{D}$ is in the first six dimensions. Table 3.3 shows the error rates for the three types of classification rules under comparison. Since the first cluster overlaps with the other clusters, the errors listed in Table 3.3 in the first row and column are larger than other rows and columns. The error rates from the proposed algorithm are comparable with those that use Bayesian optimality.

Table 3.3: Error rates for 100-D Synthetic Data

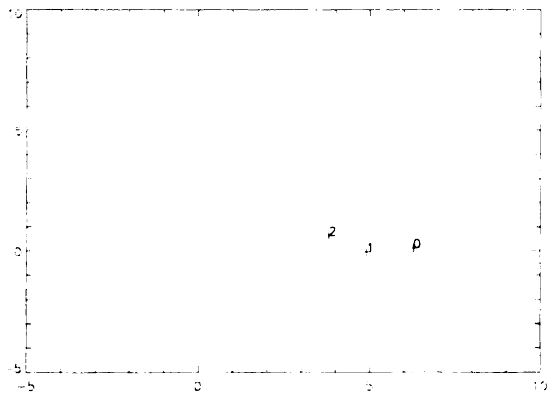
	Bayesian-GT	Bayesian-Sample	IHDR
class 1	9.6%	11.2%	18%
class 2	2.6%	2%	2%
class 3	2.8%	3.2%	4%
class 4	4.4%	3.8%	5%
class 5	2.8%	2.6%	2.8%
class 6	2.8%	3%	3.8%

The experimental results for these three synthetic data set show that the IHDR algorithm has similar performance to the batch version – HDR algorithm. Its classification rates are also comparable to that of Bayesian decision rule for these tests.

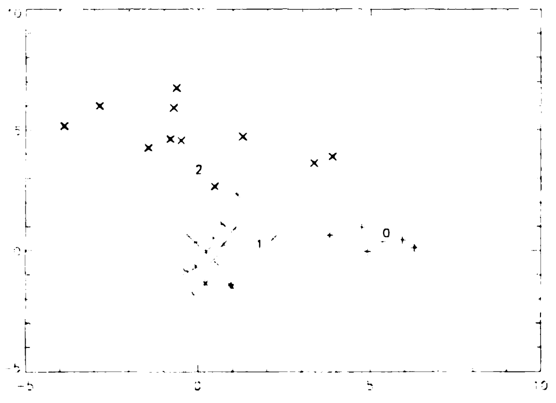
The last experiment for the synthetic data is to show the effect of the y-clustering algorithm. We used the 2-D synthetic data to show how the y means are clustered. As shown in Fig. 3.1 and Fig. 3.2, the y-means are gradually converge to the correct cluster centers.

3.4.2 Experiments using real face data

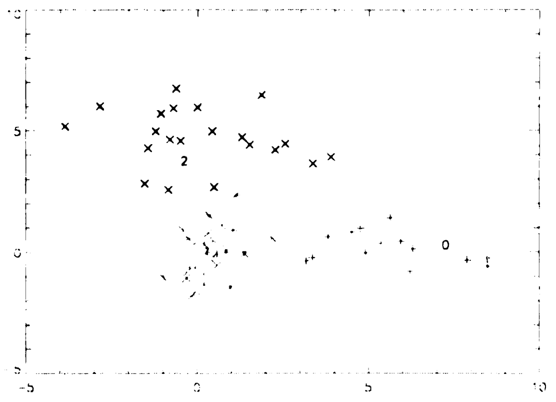
A major application of the presented algorithm is to directly deal with high-dimensional multi-media data such as images, video, or speech. We present our



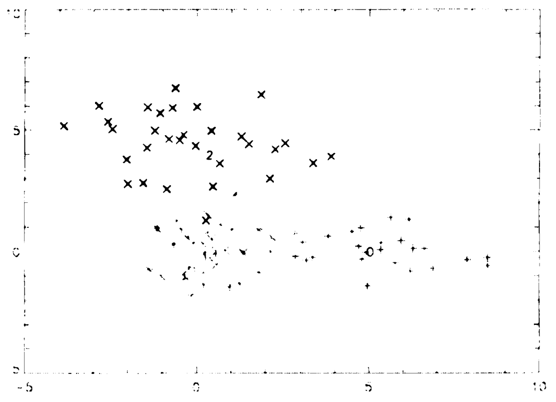
(a)



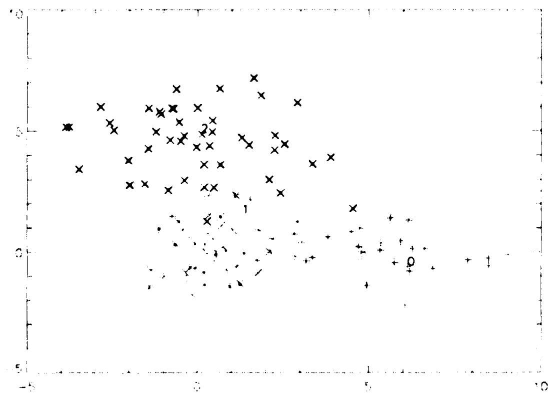
(b)



(c)

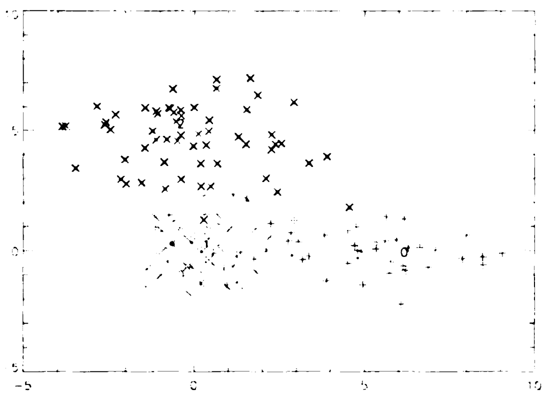


(d)

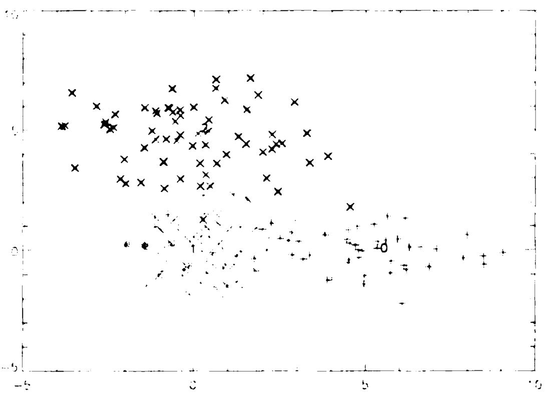


(e)

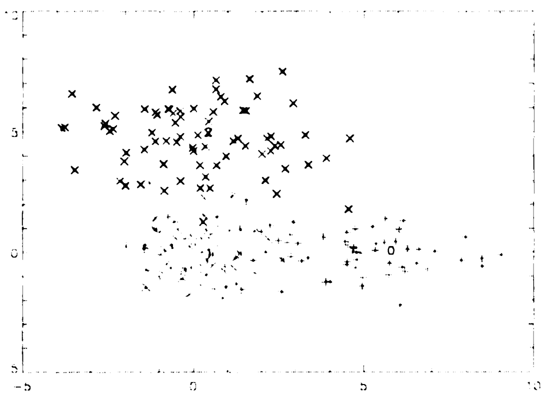
Figure 3.1: The effect of y-clustering algorithm. The number of samples for (a), (b), (c), (d), and (e) are 3, 33, 63, 93, and 123, respectively.



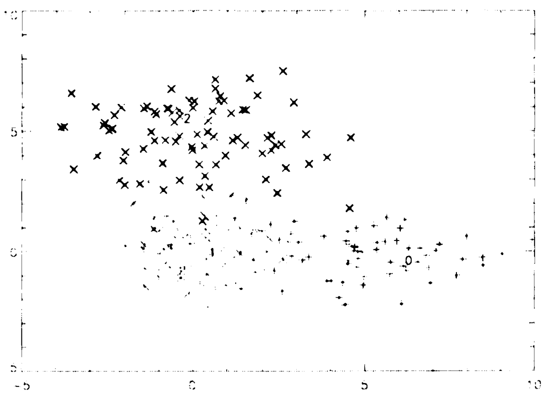
(f)



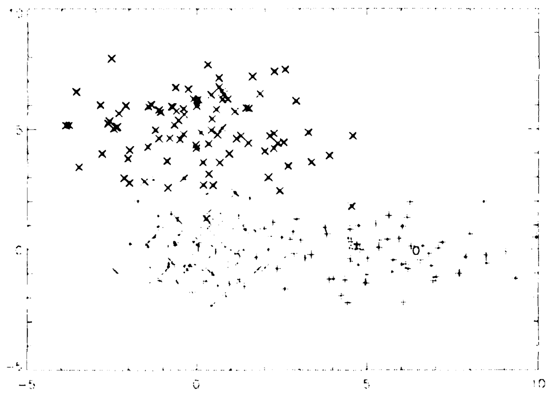
(g)



(h)



(i)



(j)

Figure 3.2: The effect of y-clustering algorithm (Continue from Fig. 3.1.) The number of samples for (f), (g), (h), (i), and (j) are 153, 183, 213, 243, and 273, respectively.

experiments with images here. We treat each image of m rows and n columns as a mn-dimensional vector, where each component of the vector corresponds to the intensity of each pixel. Statistical methods have been applied directly to these vectors of high dimensionality. This type of approaches has been very successful in the field of computer vision and now has been commonly called appearance-based methods [92] [60] [88]. Although appearance-based methods themselves do not have invariance in position, size, and orientation when applied to appearance-based object recognition, they have been well-accepted for their superior performance when input images are pre-processed images with roughly normalized position and size.

The first experiment used face images from the Weizmann Institute at Israel. The image database were constructed from 28 human subjects, each having thirty images all combinations of two different expressions under three different lighting conditions with five different orientations. An example of the face images from one human subject is shown in Fig 3.3 3.4. The preprocessed images have a resolution of 88×64, resulting an input space of 5632. The task here is to classify images into person's ID as class label. We used the mean of all training images of each person as the corresponding y vector.

The data set was divided into two groups: training set and testing set. The training set contains 504 face images. Each subject contributed 18 face images in the training set. The 18 images include three different poses, three different lightings, and two different expressions. The remaining 336 images were used for the testing set. Each subject had 12 images for testing, which include two different poses, three different lightings, and two expressions. In order to present enough training samples

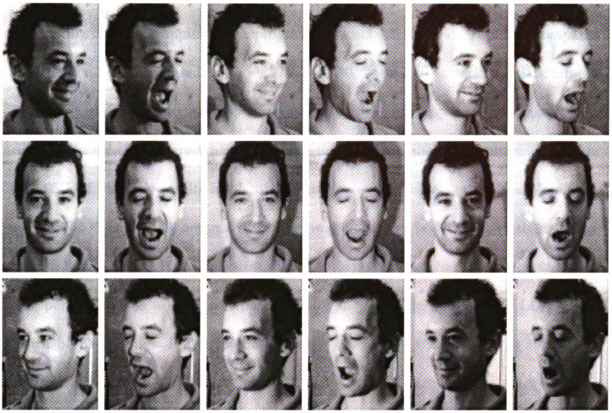


Figure 3.3: Face images from Weizmann Institute. The training images of one subject. 3 lightings, 2 expressions, and 3 orientations are included in the training set.

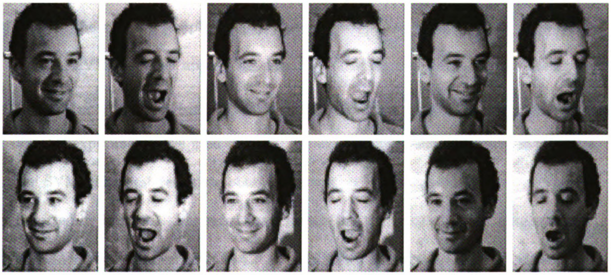


Figure 3.4: Face images from Weizmann Institute. The testing images of one subject. 3 lightings, 2 expressions, and 2 orientations are included in the testing set.

for the IHDR algorithm to build a stable tree, we artificially increase the samples by presenting training samples to the program 20 times (20 epochs). Table 3.4 compares different appearance-based methods. We used 95% sample variance in determining the number of basis vectors (eigenvectors) in the principal component analysis (PCA). PCA is faster than nearest neighbor (NN) and shares a similar accuracy. However, the 95% of variance results in about 98 eigenvectors which are much less than that of NN (5632-D!). PCA organized with a binary tree was faster than straight NN as shown in the Table. It is the fastest algorithm among all the methods we tested but the performance is worse than those of PCA and NN. The accuracy of LDA is the third best. Our new IHDR method is faster than LDA and resulted in the lowest error rate.

We also applied support vector machines (SVM) [15] to this image set to compare the performance. Support vector machines utilizes the structural risk minimization principle [93]. It results in a maximum separation margin and the solution depends only on the training samples (support vectors) which are located on the supporting planes. SVM has been applied on both classification and regression problems. We used the SVM software obtained from Royal Holloway, University of London [83] for this experiment. We used the PCA of the face images as the input features for the SVM[3] The best result we obtained by tuning the parameters of the software is reported in Table 3.4. The recognition rate of the SVM with PCA is similar to that of PCA alone. However, SVM with PCA is faster than PCA. This is because SVM has more compact representation and PCA alone needs to conduct linear search for

[3]The software failed when we used the original image input with dimensionality 5362.

every training sample.

We compared the error rate of the proposed IHDR algorithm with some major tree classifiers. CART [11] and C5.0 [76] are among the best known classification trees[4] . However, like most other decision trees, they are univariate trees in that each internal node used only one input component to partition the samples. This means that the partition of samples is done using hyperplanes that are orthogonal to one axis. We do not expect this type of tree can work well in a high dimensional space. Thus, we also tested a more recent multivariate tree OC1 [62]. We realize that these trees were not designed for high dimensional spaces like those from images We also tested the corresponding versions by performing PCA before using CART, C5.0, and OC1 and call them CART with PCA, C5.0 with PCA, and OC1 with PCA, respectively.

We have also compared the batch version of this algorithm [34] [98]. The batch version, named hierarchical discriminating regression (HDR) tree, computes statistics of training samples in a batch fashion. We expect the batch method out-perform the incremental one. However, the error rate of IHDR tree is lower than that of HDR tree for this set of data. The reason is that the same training samples might distribute in different leaf nodes for the IHDR tree because we run several iterations during training. For batch version, each training sample will only be allocated in one leaf node.

Then we performed an experiment on FERET face data set. We used the frontal views from the data set. Thirty four human subjects were involved in this experiment

[4]We have experimented the same data set using CART implemented by OC1. The performance is significantly worse than those reported in the Table 3.4.

Table 3.4: The performance for Weizmann face data set

Method	Error rate	Avg. testing time (msec)
PCA	12.8%	115
PCA tree	14.58%	34
LDA	2.68%	105
NN	12.8%	164
SVM with PCA	12.5%	90
C5.0 with PCA	45.8%	95
OC1 with PCA	44.94%	98
HDR tree	1.19%	78
IHDR tree	0.6%	74

with four face images per subject. Each person had three face images for the purpose of training. The other face image was used for testing.

A summary of comparison is listed in Table 3.5. Notice that the training time is measured for the total time to train the corresponding system. The testing time is the average time per query. To make a fair comparison, the computation time for PCA is included in C5.0 with PCA, OC1 with PCA, and CART with PCA. As shown, none of the existing decision trees can deal with FERET set well, not even the versions that use PCA as a preprocessing step. The batch version of the proposed algorithm (HDR) tree shares the same error rate as the IHDR tree. The HDR tree is faster than the IHDR in both training and testing. This is because we ran several epochs for the IHDR tree and the IHDR tree has more redundant information inside.

In order to display how the IHDR tree converges, Fig 3.5 shows the error rates vs. epoch plot. As can be seen, the resubstitution error rate converges to zero at the 5th epoch. The testing error rate reaches zero at 6th epoch.

Table 3.5: The performance of decision tree for FERET test

Method	Error rate		Time (sec)	
	Training	Testing	Training	Testing
CART	10%	53%	2108.00	0.029
C5.0	41%	41%	21.00	0.030
OC1	6%	56%	2206.00	0.047
CART with PCA	11%	53%	10.89	0.047
C5.0 with PCA	6%	41%	9.29	0.047
OC1 with PCA	5%	41%	8.89	0.046
IHDR tree	0%	0%	12.25	0.027
IHDR tree	0%	0%	23.41	0.041

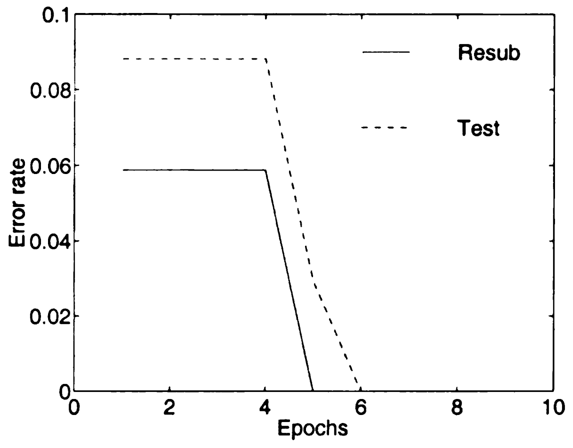


Figure 3.5: The performance plot of FERET face test 1. The plot of error rate vs. number of epochs. The "Resub" line means the resubstitution error rate. The "Test" line represents the testing error rate.

3.4.3 Experiments with autonomous navigation problem

All the tasks above are classification tasks. We present the result for a regression task. A vision-based navigation system accepts an input image X and outputs the control signal C to update the heading direction of the vehicle. The navigator can be denoted by an function f that maps the input image space $\mathcal{X}$ to control signal space $\mathcal{C}$. The learning process of the autonomous navigation problem then can be realized as a function approximation. This is a very challenging task since the function to be approximated is for a very high dimensional input space and the real application requires the navigator to perform in real time.

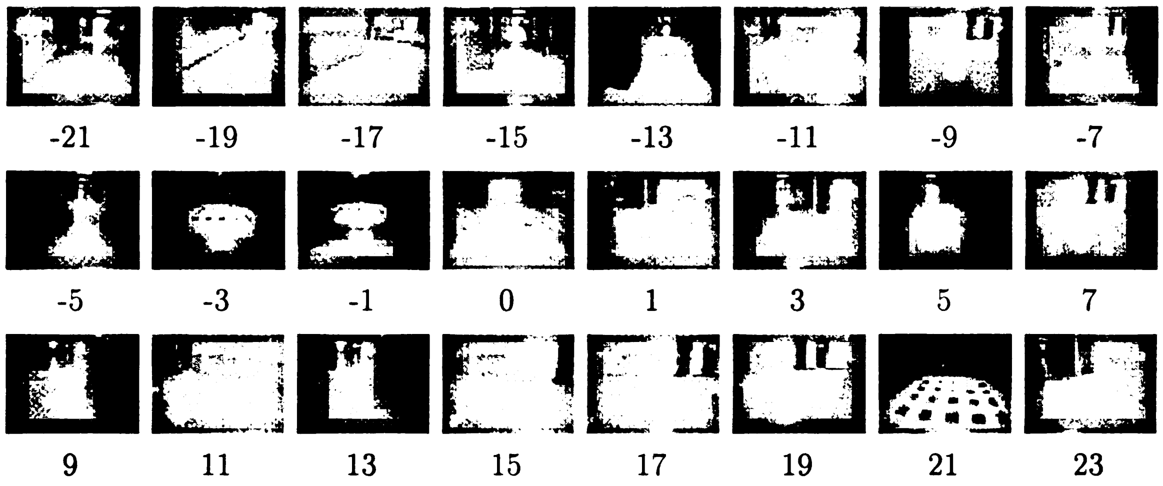


Figure 3.6: A subset of images used in autonomous navigation problem. The number right below the image shows the heading direction associated with that image.

We applied our algorithm to this challenging problem. Some of the example input images are shown in Fig 3.6. Totally 318 images with the corresponding heading directions were used for training. The resolution of each image is 30 by 40. We used the other 204 images to test the performance of the trained system. Fig 3.7 shows the maximum error rates and the mean error rates versus the number of training epochs. Both maximum error and mean error converge around the 15th epoch. Fig 3.8 gives plots of the histograms of the error rates at different epochs. As shown even after the first epoch, the performance of the IHDR tree is already reasonably good. With the increase of the epochs, we observed the improvement of the maximum error and mean error. The improvement stopped at the 15th epoch because we did not use any new training samples in each epoch and the system has perfectly fit the existing training data set. our test on real mobile robot has shown that a system of such an error level can navigate the robot very reliably for hours until the batteries are exhausted.

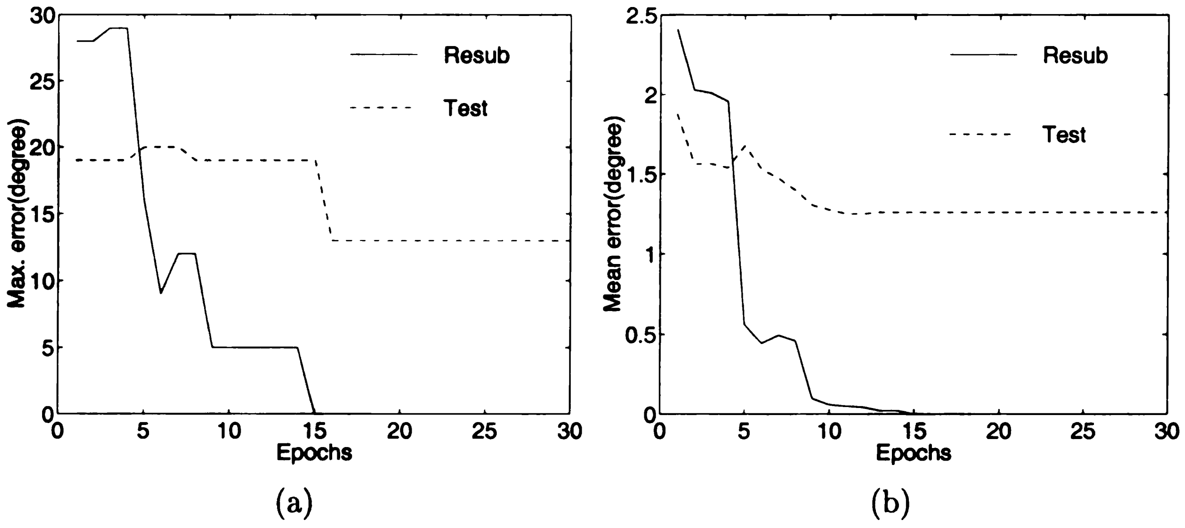


Figure 3.7: The performance of the autonomous navigation. (a) The plot for maximum error rates vs. epochs. (b) The plot for mean error rates vs. epochs. The solid line represents the error rates for resubstitution test. The dash line represents the error rates for the testing set.

We also compare our experimental results with two artificial neural networks (ANN) as reported in [99] with a consideration that the pattern-by-pattern training mode of artificial neural networks is also an incremental learning method. A two-layer feed-forward (FF) network and a radial basis function (RBF) network were used to train and test for the mapping from the image space to control signal space using the same data set as used in our IHDR tree algorithm. The results are listed in Table 3.6 which shows that our algorithm outperforms these two ANN methods.

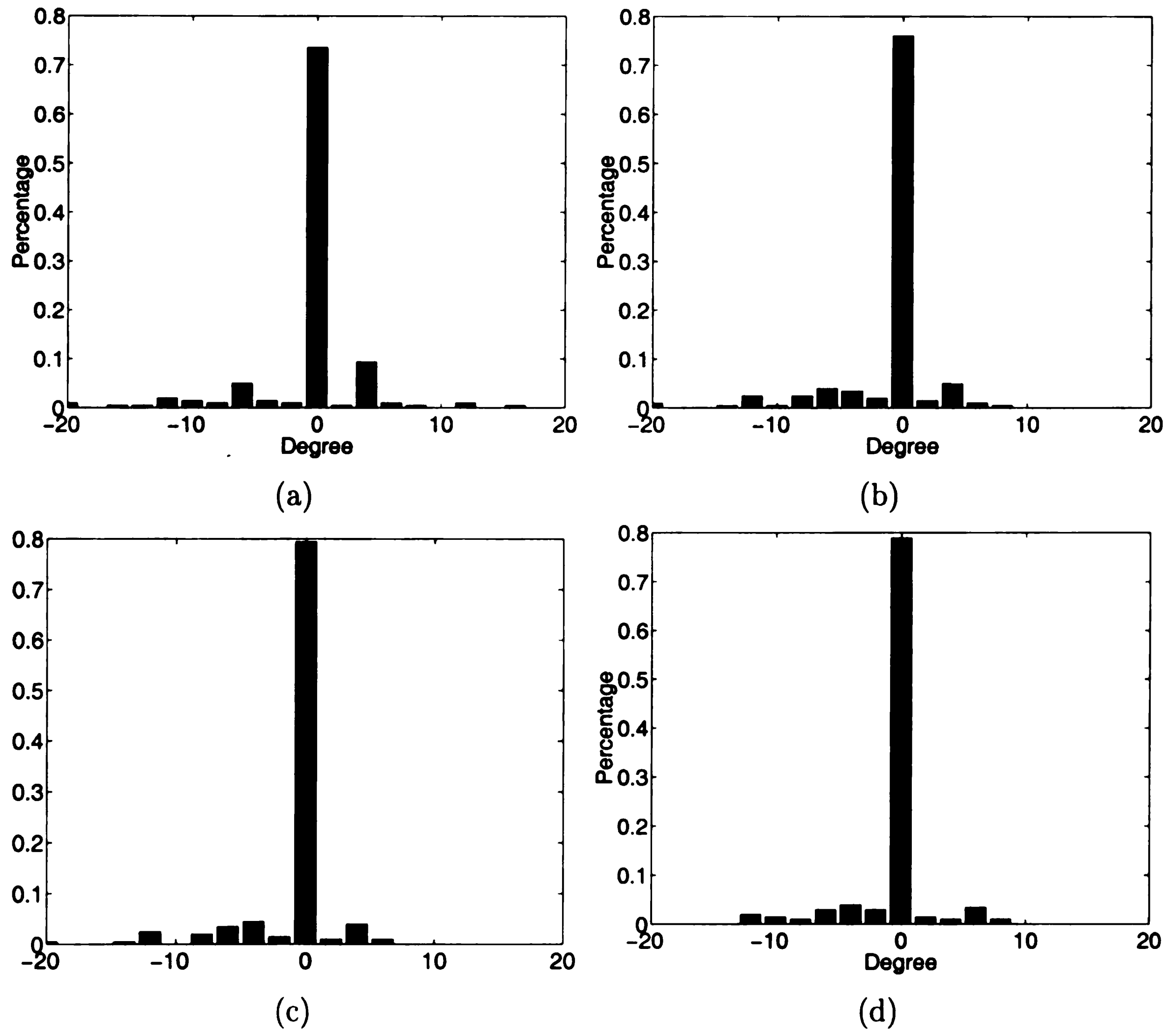


Figure 3.8: The histograms of the error rates. Plot (a), (b), (c), and (d) correspond to the histograms at epoch 1, 6, 11, 20, respectively.

Table 3.6: The performance for vision-based navigation

Algorithm	Mean error (degree)		Max. error (degree)	
	Resubstitution	Testing set	Resubstitution	Testing set
FF	1.02	2.00	10	12
RBF	1.53	1.84	12	12
IHDR tree	0.00	1.25	0	13

3.4.4 Experiments using data with manually extracted features

We further investigated how our IHDR algorithm performs on lower dimensional real data. These data used manually selected features. The same data sets from StatLog project [57] were adopted in the experiments in order to have a fair comparison with the batch HDR tree algorithm. We inserted our performance to the results in Table 2.9 and Table 2.10. The performance is slightly worse than the batch one while still comparable to all the other classifiers.

Chapter 4

Learning from Continuous Input Stream

4.1 System Overview

For the purpose of learning automation, the program-level representation should not be constrained by, or embedded with, handcrafted knowledge-level world models or system behaviors. It is very difficult to manually build a sufficient set of rules or behaviors that is general and complete enough to handle the challenging recognition tasks we have to deal with. Fig. 4.1 shows the relation between the learner and the trainer. Let X_i be the image input at time i, where X_i is with dimensionality $r \times c$, representing an image of r rows and c columns. Z_i denotes the numerical sensory input from the trainer at time i. When there is no numerical input from the trainer, $Z_i = 0$ (default value). The human trainer decides when to impose the numerical inputs. A_i denotes the action performed by the learner at time i. When $A_i = 0$, no action is performed. I_i denotes the action imposed by the trainer on the learner. When there is no action to be imposed, the trainer set $I_i = 0$. Otherwise $I_i \neq 0$ and $A_i \leftarrow I_i$, meaning the imposed action is performed immediately.

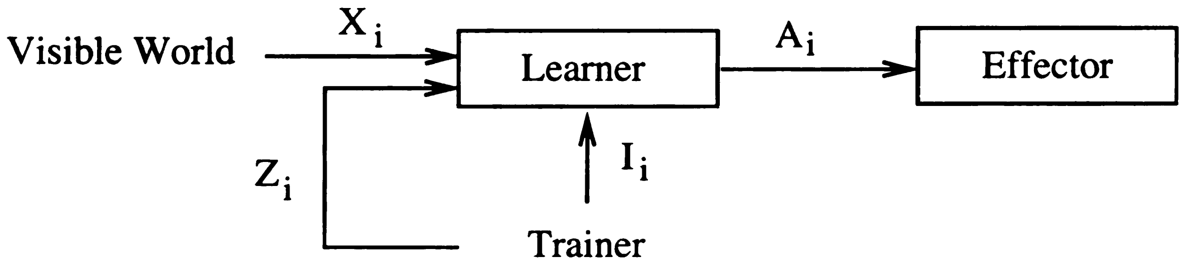


Figure 4.1: Learner and trainer. X_i: visual input. Z_i: numerical sensory input from the trainer. I_i: action to be imposed. A_i: action to be performed.

At each time instant i, the learner performs the following steps:

- Receives (X_i, Z_i, I_i).
- If I_i is on (non-zero), then $A_i \leftarrow I_i$. Otherwise, the learner uses its current state S_i to query the prediction tree and get the predicted action Ap_i. Let $A = Ap_i$.
- Use (X_i, Z_i, A_i) as input to query and update decision tree. This gives a primitive cluster P_i.
- Use (P_i, P_{i-1}, S_i) to query and update a association tree. This gives a state S_{i+1}.
- Use (S_i, S_{i+1}) to update the prediction tree. Let the current state be S_{i+1}.

The following sections explain how these are realized.

4.2 The Split SHOSLIF

SHOSLIF [96] [87] is a general approximator for supervised or unsupervised learning. It approximates a high dimensional function $f : X \mapsto Y$, by incrementally creating a tree from training samples. SHOSLIF uses the principal component analysis (PCA) and linear discriminant analysis (LDA) to recursively build a feature space for every internal node of the tree. The basis vectors from PCA are called most expressive features (MEF) vectors and those from LDA are called the most discriminating feature (MDF) vectors. Each leaf stores the sample x_i it represents. Given any input $x \in X$, the tree finds the top k matched leaf nodes. For supervised learning, each leaf node stores the desired output $y_i = f(x_i)$. The output $f(x)$ is approximated by a weighted average among the output vectors of top k leaf nodes. For unsupervised learning, it reports the top k matched leaf nodes with a confidence measure. The time complexity for finding top k (k is a constant) matched leaf nodes is $O(\log(n))$ for a tree with n leaf nodes.

An incremental version of SHOSLIF is described in [97] whose task is to build a tree incrementally. It uses a scheme that involves microtrees and macrotrees. The required computation time for learning each sample is not uniform. In order for each machine cycle to be finished within a roughly constant time, we introduce a new scheme here. We do not evolve the node, unless it becomes a leaf after the deleting process has deleted the subtree rooted from it. The new update-node algorithm is as follow:

Procedure 6 `Update-node (N, X)`*: Given a node N and the input X, update the* *subtree rooted from node N.*

```
1. if (node is a leaf)
    (a) if input-sensitive(X, node-¿mean) // if X is not the same as mean with noise considered
        • split-node(node, X); // node is split into two nodes
    (b) else
        • matched(node, X); // no need to learn
2. else if (MEF-projection(X - mean) < 0 ) // decide which half-space X falls into
    • update-node(node-¿leftchild, X);
3. else
    • update-node(node-¿rightchild, X);
```

In order to avoid storing unnecessary samples, especially those that are very similar, a learning sample x does not cause a new leaf node to be created if x is within δ distance from the top matched node x_i and only x_i is updated by x to record the mean and variance of the samples falling into the leaf node. Thus, each leaf stores a prototype sample with the pre-defined input-sensitivity δ.

In the procedure split-node, we do not change the X-field of the existing node. The X-field (mean) of a node B is fixed as soon as it is created. However, we keep a reserved splitter inside B as the mean vector $m^{(n)}$ and an estimate of the MEF vector $v^{(n)}$ as more inputs pass through B, where n is the number of visits to B. As soon as B becomes a leaf node (by e.g. the forgetting process described in section 4.3 that deletes nodes), we replace $m^{(n)}$ and $v^{(n)}$ for its current splitter. When B node spawns new children, this new splitter is used to split its children population into left and right subtrees.

Now, consider how to update the reserved splitter. Suppose that a node B with $m^{(n)}$ and $v^{(n)}$ receives a new input x. The new mean is updated as

$$m^{(n+1)} = \beta \frac{nm^{(n)} + x}{n+1} + (1-\beta)x$$

where $0 < \beta \leq 1$ is a depreciation factor for the history so that very old samples will be disregarded faster than the standard case with $\beta = 1$. When $n = 2$, the vector $v^{(2)}$ is a unit vector aligned with $m^{(1)} - x$. When $n > 2$, vector is updated by x using an appropriate method, so that it aligns roughly with the principal component. For example, we can use either the power method or a probably more space saving method (no need to store covariance matrix):

$$v^{(n+1)} = \gamma v^{(n)} + (1-\gamma)(m^{(n)} - x)$$

where $0 \leq \gamma < 1$ determines how much the old $v^{(n)}$ should be retained as opposed to the simple difference method where $\beta = 1$ for $n = 2$. Note that this vector can happened to be zero, in which case, $(m^{(n)} - x)$ can be directly used. The vector $(m^{(n)} - x)$ can never be zero because the input sensitivity test guarantees $||m^{(n)} - x|| > 0$.

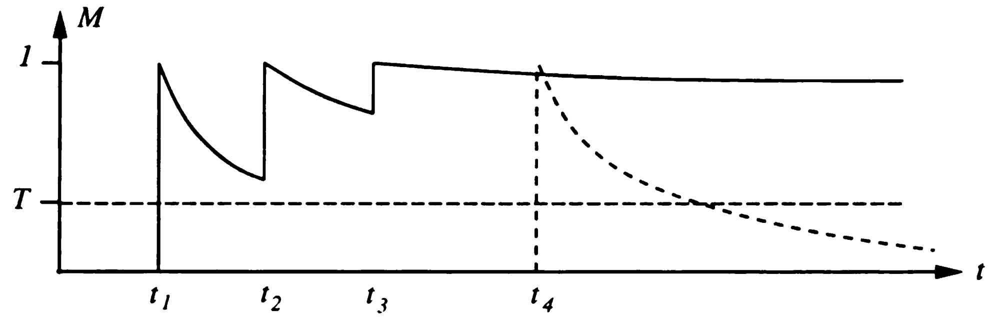


Figure 4.2: Update of memory trace (strength) M through time t. The solid curve represents an element which is visited often enough to be kept. The dashed curve indicates an element that is not visited often enough and thus, it falls below the threshold T before being visited again.

4.3 Forgetting

Due to a finite memory space, the system cannot remember all the associations that it has come across. In fact, it should forget for generalization and for space limitation. Many associations must be forgotten. The utilization of association is indicated by *co-occurrence frequency* and *occurrence intervals.*

Let us consider a memory element, a tree node, which is used in our system for memorizing an association it represents. Each element has a memory residual register whose updating curve is shown in Fig. 4.2 which resembles what we know about human memory characteristics [6] [36].

Each visit to the same element makes the trace to be reset to 1 and then the curve declines using a next slower speed. For example, we can define a series of memory *fade factors* $\alpha_1 < \alpha_2 < ... < \alpha_m \approx 1$. α_i is used for an element that has been visited i times. The memory trace r can be updated by $r \leftarrow r\alpha_i^t$ where t is the number of system cycles (refresh) elapsed since the last visit to the node. Thus, we do not need to visit all the elements at every system cycle. When an element is visited, its memory trace is updated first from what remains from the last visit. If the memory trace falls below the designated threshold, the memory element should be deleted and so it is marked as to-be-deleted. If what is deleted is more than a single element, the deleting process will not delete it right away to avoid consuming too much CPU time in a real time process. Instead, it puts those elements in a garbage buffer which is to be cleaned when the learner is "sleeping."

4.4 Spatiotemporal clustering

The objective of spatiotemporal clustering is to form primitive clusters (P-clusters) using not just spatial information, but also temporal information. A P-cluster consists of a number of prototypes each being represented by a leaf node. Each P-cluster corresponds to a state of the system.

Fig. 4.3 shows a schematic illustration of the temporal-adjacency cluster. The cluster represents a mapping $g : X \mapsto Y$ which maps from input space X to output space Y. X is the sensory input space of a particular sensor and Y is the output space containing P-clusters. Given x, $y = g(x)$ is the image of x, representing the corresponding P-cluster. In order to preserve the necessary topology in X, Y has the same dimensionality as X or has a reduced dimensionality. If X has a dimensionality $n \times n$ corresponding to an image space of $n \times n$ pixels, then $Y = R^{m \times m}$, where R is the set of all real numbers. The ratio of dimensionality difference $(n \times n)/(m \times m)$, is called the factor of dimensionality reduction (FDR). This reduction of dimensionality is reasonable because all we need is to have Y space to roughly keep the topology of

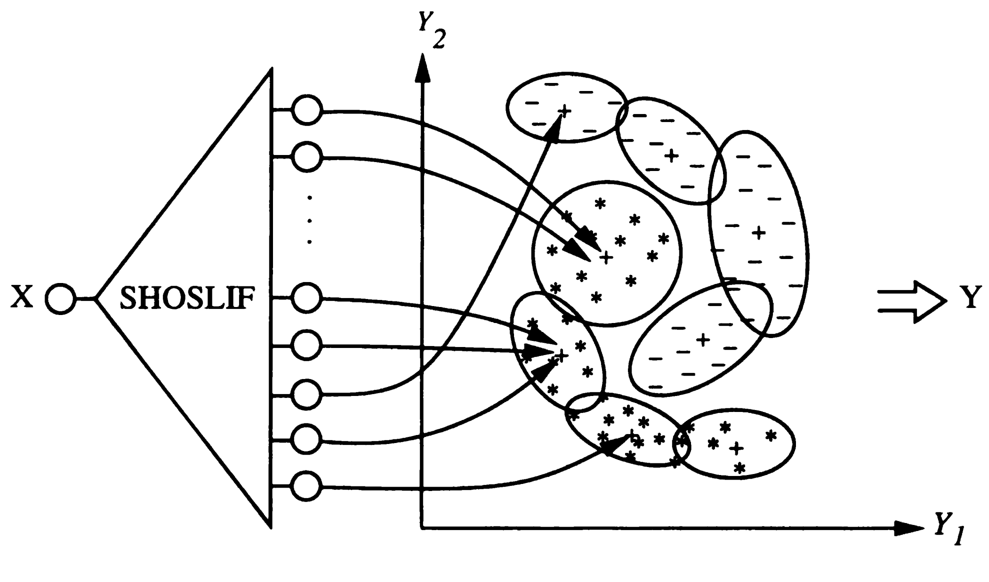


Figure 4.3: A schematic illustration of the temporal adjacency cluster. Each region in Y space represents a P-cluster. A dashed curve is used to roughly indicate the samples that fall into the cluster. The circles to the right of the SHOSLIF tree are the leaf nodes of the tree. Each $-$,$=$ or $*$ sign indicates the relative positions of leaf nodes (δ-prototype) in the input space to the SHOSLIF tree. Two different signs representing two semantically different input sequences. Each leaf node in the SHOSLIF tree has a pointer to the Y space, which points to the center of the corresponding P-cluster or close to it after a significant amount of gravity pulling and merging. Y is shown here as 2-D for visualization, but it is typically of the same dimensionality as X or with a reduced dimension by a factor of dimension reduction.

X space. When a δ-prototype is first created, it is represented by a single training sample x_i. We let its image y be the same as x_i, $y_i = g(x_i) = x_i$ (with FDR taken into account if $FDR > 1$.) Later successful matching with x_i will cause y_i to be pulled toward the Y vectors of its neighbors in X.

Consider an input stream $x = \{x(0), x(2), x(3), ..., x(t), ...\}$. where each frame $x(i)$ is in X and i is the time index. The stream x forms a trajectory in $X \times T$ space, where $T = \{0, 1, 2...t...\}$ represents the set of time indexes. We need to chunk the trajectory into segments of some appropriate length of events. Each segment corresponds to a P-cluster. Two frames $x(l)$ and $x(m)$ are *temporally adjacent* if one is followed immediately by another within a time window w, i.e., $0 < |l - m| \leq w$, where w

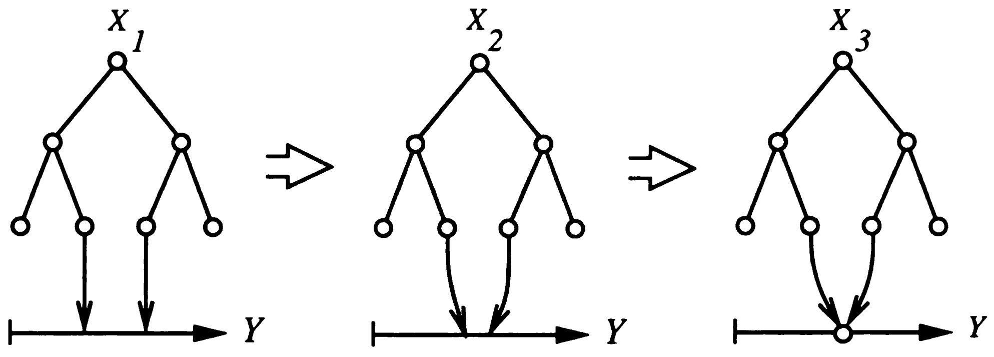


Figure 4.4: The effect of pulling in spatiotemporal clustering. The SHOSLIF tree is repeatedly visited by $x_1 \approx x_2 \approx x_3$ that are nearby in spatiotemporal domain. The two leaf nodes are among the top k matched leaf nodes, although they are separated early in the tree. Consequently, their Y vectors are merged when they are sufficiently close and the corresponding prototypes of the two leaf nodes belong to the same P-cluster.

is the adjacency window (default is 1). Given each frame $x(t)$ at time t, its top k matched prototypes are considered. The top match gives its image y. Among top k matched δ prototypes with a sufficient matching confidence, if two δ-prototypes are less than σ ($\sigma > \delta$) distance apart in X space and they are temporally-adjacent, the images of these two δ-prototypes are pulled together using a simulated force, where the number of visits of each prototypes is the "mass". Thus, among the temporally adjacent δ-prototypes with a σ radius in X, those mostly often visited prototypes tends to become the center of the cluster. When the y images of two prototypes are very close, their y images become one. Merged y vectors form the center of the P-cluster. Merged centers of y vectors have a higher mass. It can quickly pull nearby y vectors and merge them. This speeds up the clustering speed. To avoid two neighboring P-clusters to be slowly pulled together, a maturity schedule is applied so that P-clusters that are mature enough will no longer be moved — an effect of long term memory. Fig. 4.4 shows that the pointers emitting from leaf nodes gradually point to the center of the P-cluster in Y space.

Note that two inputs x_i and x_j will not have the same image (state) if they are within σ distance but never occur one after another. In other words, the P-clusters are clusters of spatiotemporal clusters, but δ-prototypes in SHOSLIF leaf nodes are spatial-only prototypes and their spatial radius are typically much smaller than that of a spatiotemporal cluster.

The output from the spatiotemporal cluster can be regarded as the lowest-level discrete state. The techniques used here have a close relationship with learning vector quantizer (LVQ) [43] [44] and k-mean clustering [38]. The differences are (1) temporal concept; (2) unknown number of clusters; (3) the nearest neighbors must be found quickly and thus the SHOSLIF tree is used; (4) learning is incremental.

4.5 The experiments

4.5.1 The system flow

Depending on whether the action is imposed or not, the learning can be classified into *action-imposed* learning and *action-autonomous* learning. *Action-imposed* learning is such that the extro-effector part of $a(t+1)$ is supplied by the trainer. For example, hand-in-hand learning can be used by human adult to teach a child how to use a pen. Otherwise, the learning is *action-autonomous* learning.

Fig. 4.5 illustrates a flow chart of action-imposed learning. If the trainer imposes an action on an effector at any time through, e.g., through a joystick, the system performs action-imposed learning for that effector. Otherwise, the system performs

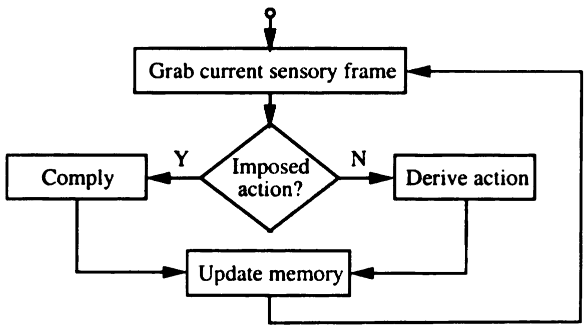


Figure 4.5: The flowchart for action-imposed learning. The system learns while performing.

action-autonomous learning.

Suppose that the machine agent M has recorded in memory $B = \{(x(i), s(i), a(i)) \mid i = 0, 1, ..., t-1\} \cup \{s(t), a(t)\}$. Note that $s(t), a(t)$ are the result from sensory input $x(t-1)$. According to the flow diagram in Fig. 4.5, M grabs the current sensory frame $x(t)$. Then, M computes the next state $s(t+1)$. If an action is imposed, $a(t+1)$ is supplied by a human being (or environment) and thus M complies by sending $a(t+1)$ to the effector and then updates its memory by replacing B by $B \cup \{x(t), s(t+1), a(t+1)\}$. If an action is not imposed, M derives action $a(t+1)$ based on the past experience using a simplified g in Eq. (1.2) as follows. First, M finds the best matched state:

$$j = \text{argmin}_{0 \leq i \leq t} ||s(t+1) - s(i)||. \tag{4.1}$$

Then, the output action is determined as the action associated with the best matched $s(j)$: $a(t+1) = a(j)$. The memory update is done as before. After $x(t+1)$ is grabbed in the next machine cycle, which may include the resulting reward sensed by the biased sensor, the system memory becomes $B = \{(x(i), s(i), a(i)) \mid i = 0, 1, ..., t+1\}$.

As can be seen, this oversimplified version of action-imposed learning can do only a little generalization by extending the action learned by the nearest neighbor $s(j)$ (or multiple neighbors with action interpolation) to the current new state $s(t+1)$, whenever no action is imposed by the human.

4.5.2 Experimental results

Although the method does not restrict the content domain in which the system learns, we apply it to face recognition training, with supervised learning. The hypothetical theme is to train the learner to learn "Tell the name of the face". The interactive training was conducted on-line. Several persons entered the scene one by one, stayed for a while, and left. During the stay, the system trainer gives a coded numerical sensor signal Q. In the mind of the trainer, it means question "who is it?" because the trainer immediately imposes a numerical action which is regarded as the name of that person by the trainer. First, we present the result of experiment using synthetic data. The goal is to see how the system works. Data for person A (P_A) and person B (P_B) are simulated. To simplify the notation of later presentation, we use the symbols summarized in the following tables.

Table 4.3 explains how the training session was conducted through continuous machine cycles, where each T_i covers several image fresh cycles (i.r.c.) as listed below: $T_1 : 0-19$, $T_2 : 20-29$, $T_3 : 30-39$, $T_4 : 40-49$, $T_5 : 50-69$, $T_6 : 60-79$, $T_7 : 80-89$, $T_8 : 90-119$, $T_9 : 120-129$, $T_{10} : 130-139$, $T_{11} : 140-149$, $T_{12} : 150-169$, $T_{13} : 160-179$, $T_{14} : 180-189$, $T_{15} : 190-199$. The appropriate action were imposed

Table 4.1: The representation for image input

Symbol	Meaning
b	Background
A_e	P_A entering
A_s	P_A stay
A_l	P_A leaving
B_e	P_B entering
B_s	P_B stay
B_l	P_B leaving

Table 4.2: The representation for numerical sensor input and action output

Symbol	Meaning
S	Silence
Q	Asking question
N	No action
A	Answering "A"
B	Answering "B"

by the human in the training session. Since this is a noise-free case, the resulting primitive clusters are combinations of images, numerical sensors, and actions. These clusters are the inputs to level-one in the automatic spatiotemporal level building. Notice that the recurrence input to the level-one tree helps to remember the history of training experience. If we do not use the recurrence input, an ambiguous prediction will occur. In this case, P_2 will go to both P_3 and P_5.

The training session is followed by the testing session. A question is asked after presenting a face. The results are summarized in Table 4.4. The results show that both faces can be recognized correctly because the correct reply "A" or "B" was received. The time periods correspond to different image refresh cycles in the test

Table 4.3: The Training and the resulting primitive clusters

Period	See	Hear	Action	P-Clusters
T_1	b	S	N	P_0
T_2	A_e	S	N	P_1
T_3	A_s	S	N	P_2
T_4	A_s	Q	N	P_3
T_5	A_s	S	A	P_4
T_6	A_s	S	N	P_2
T_7	A_l	S	N	P_5
T_8	b	S	N	P_0
T_9	B_e	S	N	P_6
T_{10}	B_s	S	N	P_7
T_{11}	B_s	Q	N	P_8
T_{12}	B_s	S	B	P_9
T_{13}	B_s	S	N	P_7
T_{14}	B_l	S	N	P_{10}
T_{15}	b	S	N	P_0

session were listed as follows. $T_1 : 0 - 14$, $T_2 : 15 - 20$, $T_3 : 21 - 25$, $T_4 : 26 - 27$, $T_5 : 28 - 30$, $T_6 : 31 - 35$, $T_7 : 36 - 52$, $T_8 : 53 - 58$, $T_9 : 59 - 66$, $T_{10} : 67 - 68$, $T_{11} : 69 - 70$, $T_{12} : 71 - 75$, $T_{13} : 76 - 80$.

In the next experiment, we added some noise to the synthetic input data (both training and testing sessions). Different signal noise ratios (SNR) resulted in different primitive clusters at level zero. The number of states at level zero increased when the SNR decreased. Since we do not manually define the state representation, the number of states in level zero does not affect the performance much. Because of the application of nearest neighbors, very similar prototypes are used as inputs at level one even if the matches are not exactly the same. However, if SNR was too small, the retrieval was not perfect. The test results are shown in Table 4.5 and 4.6, where each period T_i' corresponds to almost the same but somewhat different image refresh

Table 4.4: Testing with noise-free images

Period	See	Hear	Action
T'_1	b	S	N
T'_2	A_e	S	N
T'_3	A_s	S	N
T'_4	A_s	Q	N
T'_5	A_s	S	A
T'_6	A_l	S	A
T'_7	b	S	N
T'_8	B_e	S	N
T'_9	B_s	S	N
T'_{10}	B_s	Q	N
T'_{11}	B_s	S	B
T'_{12}	B_l	S	B
T'_{13}	b	S	N

cycles as that for Table 4.4. The system failed to reply correctly when SNR went down about 2db as shown in Table 4.6. However, this does not reflect the system's noise immunity because the data are simulated and the task in the simulation is relatively simple.

Next, we present the experiment results for real images. Fig. 4.6 shows some frames that involve one person. The weight of the center pixels of an image is larger than that of the boundary as indicated by the first image in Fig. 4.6 so that a scene element at center will have a larger effect than one at the periphery. The system autonomously grabbed images frames from the incoming video stream and self-organized its internal representation. The resolution of an input image was 50 by 50 pixels. Fig. 4.7 shows the resulting states of level zero that correspond to the segment in Fig. 4.6 after forgetting, pulling, and merging. The number of level zero states were reduced to 18. Since the input of the level zero contains numerical input, image in-

Table 4.5: Result when inputs are noisy with SNR = 14db

Period	See	Hear	Action
T'_1	b	S	N
T'_2	A_e	S	N
T'_3	A_s	S	N
T'_4	A_s	Q	N
T'_5	A_s	S	A
T'_6	A_l	S	A
T'_7	b	S	N
T'_8	B_e	S	N
T'_9	B_s	S	N
T'_{10}	B_s	Q	N
T'_{11}	B_s	S	B
T'_{12}	B_l	S	B
T'_{13}	b	S	N

put, and numerical action, the state representation is a combination of these three components. In Fig. 4.7 we display the states by showing the image and the values of numerical sensor and numerical action as a pair (S, A). These automatically derived states approximate the prototypes of sample population with detail forgotten (merged).

The system automatically built level one. The input to level one includes states from level zero (primitive clusters) and the current state of level one. The state representation in level one is recursive and it is difficult to give an intuitive visual display. In the training session of this experiment, eight people were presented to the system one by one. When a person stayed in front of the camera, the trainer asked the question by using numerical sensor input and imposed action (the label of that person) after the question. Later, the person entered the scene again for test. No action was imposed in the testing phase. The system performed according to its

Table 4.6: Result when inputs are noisy with SNR = 2db

Period	See	Hear	Action
T'_1	b	S	N
T'_2	A_e	S	N
T'_3	A_s	S	N
T'_4	A_s	Q	N
T'_5	A_l	S	N
T'_6	b	S	N
T'_7	B_e	S	N
T'_8	B_s	S	N
T'_9	B_s	Q	N
T'_{10}	B_l	S	N
T'_{11}	b	S	N

Table 4.7: Real Images: The Resubstitution Test

Person	Ask at (i.r.c.)	Answer at (i.r.c.)	Answer
1	27-36	32-44	1
2	137-146	145-154	2
3	237-246	243-254	3
4	335-344	346-352	4
5	432-441	437-450	5
6	530-539	536-547	6
7	637-646	642-654	7
8	743-752	754-760	8

predicted action. Some image frames seen by the system during testing were shown in Fig. 4.8. During the entire test session for all eight persons, the system replied correctly except for one person at roughly the right time (after being asked) and did not "speak" when it was not supposed to. The results are summarized in Table 4.7 and 4.8. The test result shown in Table 4.7 uses video sequence for training. That in Table 4.8 uses new video stream for testing and thus involved various noise and view variation.

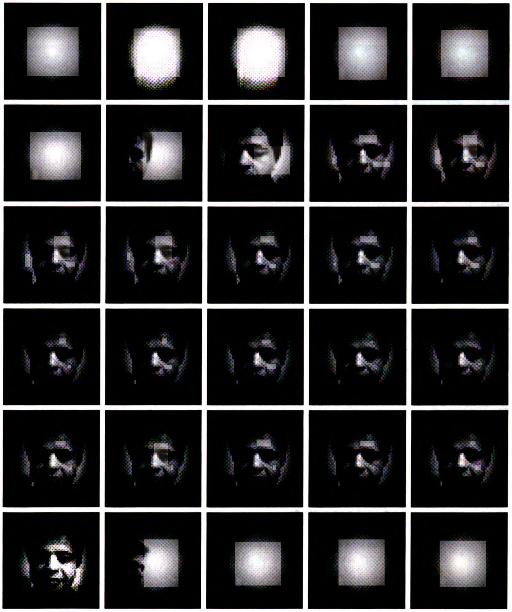


Figure 4.6: A temporally subsampled segment of the real-time video stream during which eight persons were presented to the system for training. Only one person is involved in this segment. The images are shown in the English reading order.

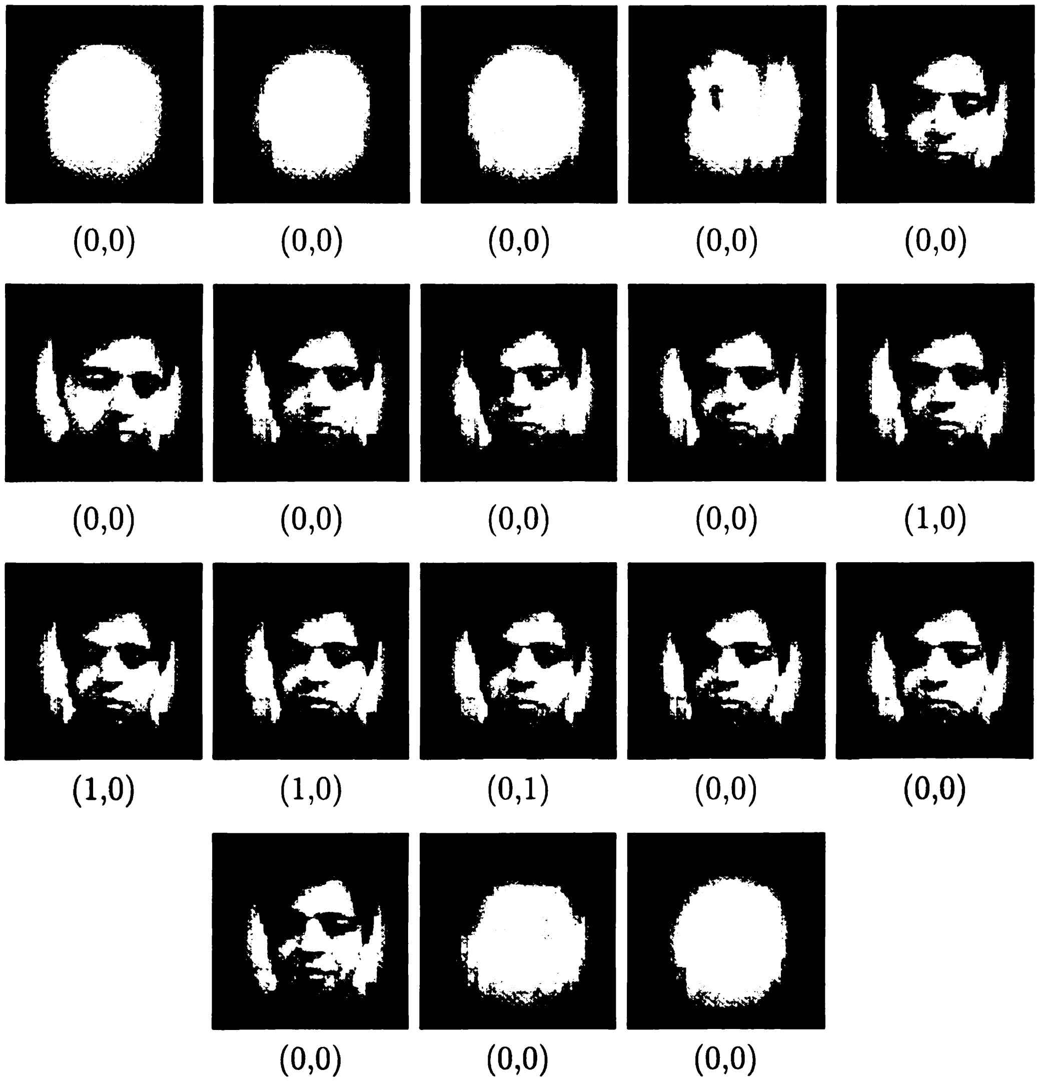


Figure 4.7: State representations at level zero. The values of numerical sensor and numerical action are represented as (S, A). Where $S = 0$ means "silence" and $S = 1$ means "asking question". $A = 0$ means "no action" and $A = 1$ means "person 1".

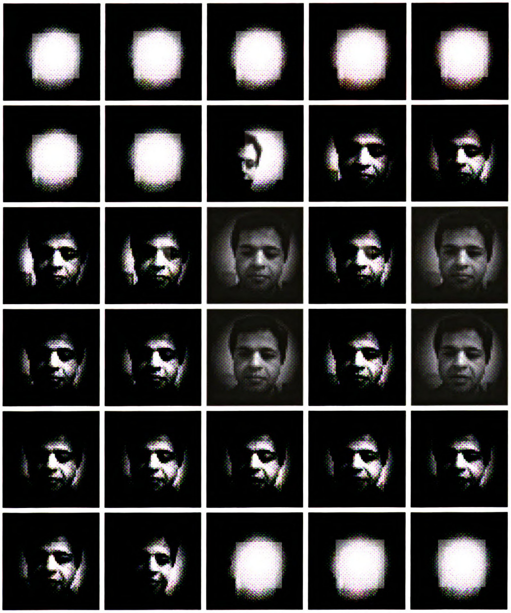


Figure 4.8: A temporally subsampled segment of the real-time video stream during the test session, corresponding to the person shown in Fig. 4.6.

Table 4.8: Real Images: The Disjoint Test

Person	Ask at (i.r.c.)	Answer at (i.r.c.)	Answer
1	37-46	38-45	1
2	137-146	147-162	2
3	237-246	256-263	3
4	335-344	354-357	4
5	425-434	445-451	5
6	537-546	556-563	4
7	632-641	651	7
8	744-753	763-770	8

The attention mechanism is then applied to improve the performance. A typical attention gates for both channels are shown in Fig. 4.9. The attention mechanism selects the modalities according to the novelty of the signals. The results are shown in Fig. 4.10. All eight persons got right response at roughly appropriate time during the testing phase, except the single incorrect short transition shown in Fig. 4.10.

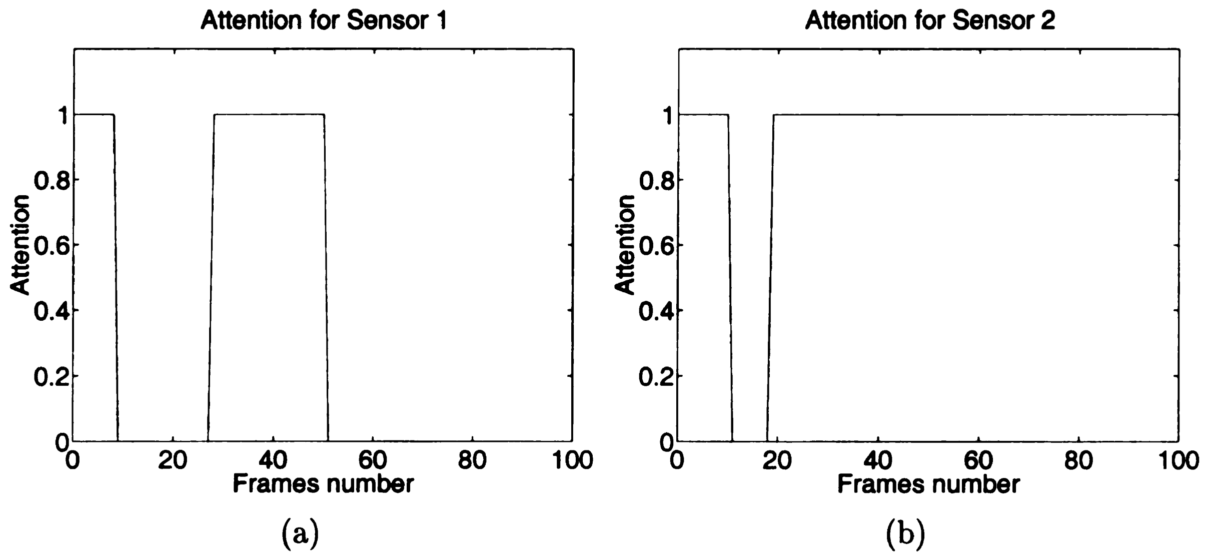
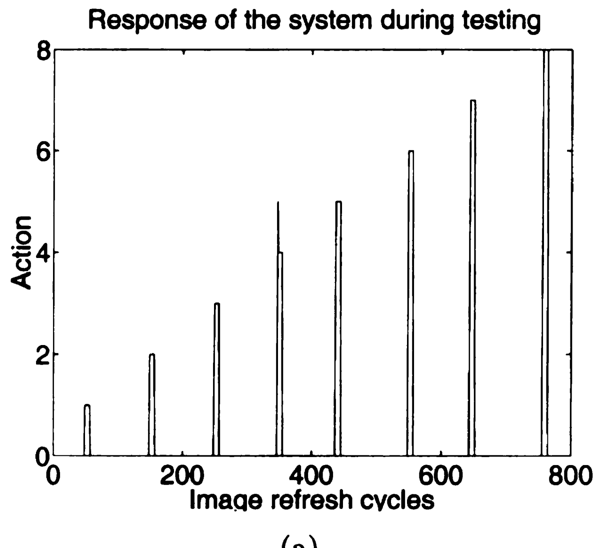
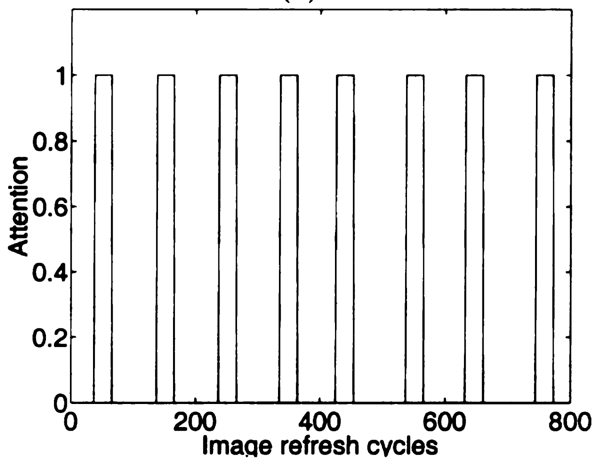


Figure 4.9: Attention gates for two channels in 100 i.r.c. (a) is for numerical input and (b) is for image input. When the attention is 1, the gate is open. Otherwise, the gate is closed.



(a)



(b)

Figure 4.10: The testing results which uses attention selection mechanism. (a) the system response during the testing phase. (b) the attention for the numerical channel. The peak at 347 i.r.c in (a) indicates a wrong answer at the beginning of person 4 entering. The wrong response is corrected by the latter signals.

Chapter 5

Application to the Robotics

5.1 Introduction

The conventional mode of developmental process for a robot is not automatic — the human designer is in the loop. Given a robotic task, it is the human designer to understand the task. Based on his understanding, he comes up with a representation, chooses a computational method, and writes a program that implements the method for the robot. During this developmental process, some machine learning may be used, during which some parameters are adjusted according to the collected training data. However, these parameters are defined by the human designer for the task given. The resulting program is for this task only, not for any other tasks. This manual development paradigm has met tremendous difficulties for tasks that are too muddy for human to provide an effective representation. For example, tasks require perception of environment but the environment is totally unknown at the time of programming.

Due to the tremendous difficulties in sensing and reconstructing unknown scenes, the behavior methods, such as those by Brooks [13], aim to avoid the drawbacks of the conventional sense-model-plan-act (SMPA) framework. With a behavior-based method, the programmer converts hand-designed behavior rules for the robot into the control program. Although such an approach has successfully created robot's behaviors that have been explicitly programmed in, the huge number of sensed conditions and the desired behaviors requires very complex behavior rules, which tends to be difficult to design and verify, especially when the number of behaviors become large. In the work reported here, we investigate the possibility of letting a system learn without explicitly programming behavior rules into a system. In particular, the system has not been calibrated, because it is not a trivial task to calibrate a dynamically changing hand-eye-head system. The task of system design is mainly to develop a self-organization scheme for the system. The system acquires task execution capability through interactive learning process, in which it learns by sensing and doing.

To work in an unstructured environment, robots need sensors to sense the environments and translate what they sensed to what they can relate to their actions. The basic issue is the calibration issue, calibrating the relationship between the sensor and the other parts of the robot. For vision-guided robotic system, the calibration is not a trivial task. Two types of approaches exist. The first type requires the human designer to explicitly model the relationships between sensors and actuators. The result is a set of equations with a number of parameters that need to be estimated. Most of the existing works belong to this category (e.g., [85], [53], [94], [68], [17], [21], [26]).

The second type of approach does not employ explicit parameterized models. But rather, a general approximator is used to map the sensor space to the actuator space. A common tool to accomplish this is the artificial neural network [41], [53], [94]. The first type of approach is effective when the system can be accurately modeled with a few parameters, but has problems when the system configuration is time varying or has a lot of degree of freedoms. The second type does not have the latter limitation due to the generality of the neural network approximator. However, the neural networks require a huge number of iterations and thus, not suited for interactive learning or task learning where each case presented must be learned immediately on line. The system must remember each training example presented and generalize to other location based on relatively few examples learned.

For vision tracking, positioning control such as in [9], they do not use a learning scheme. Some of them focus on the moving objects as in [52]. For learning by watching such as in [101] and [102], they generate a sequence of robot control language as final outputs. Some researches are in the area of learning task level as in [56] and [1]. Many approaches place a constraint on the positioning and(or) movement of the target as in [69], which assumes that the depth of the plane on which objects travel is known *a priori*. In [3], it is assumed that the target will maintain a regular and repetitive movement that can be analyzed and intercepted. In [84], the three-dimensional movement of the manipulator is decomposed into movements which are at most two-dimensional. In [31] [32] [33], a recursive partition tree (RPT) is used to approximate the mapping from the input to the output and applied to stereo calibration and sensor-based action sequence learning and execution.

The work reported here is to study a mechanism which enables a hand-eye-head system to perform some task sequences with explicitly hand-crafting task rules into the control program. With this goal in mind, two criteria are important. The first is the generality — the system should not be imposed with certain rules that prevent it from learning various tasks. The second is the scalability — the system can not slow down quickly when the number of learned cases or tasks increase. The scalability is important since the system needs to perform in real time eventually.

Specifically, we use the incremental hierarchical discriminant regression (IHDR) tree described in Chapter 3 to organize the input space into a hierarchy of coarse-to-fine partitions which approximate the mapping between the sensing-task space and the temporal action space. This tree shares the common characteristic with the well known decision trees [76], regression trees [11] and clustering trees [38] in the sense that uses a tree to organize information in a hierarchical manner. Moreover, the building procedure of the IHDR tree is not iterative. This allows the system to learn in real time. The incremental construction capabilities of the IHDR tree allows the tree to grow according to the performance of the current tree. We also perform inter-leaf interpolation for the multiple path probing, which results in a good approximation and generalization performance without need for time-consuming batch analysis of the entire set of data during tree construction. The method has been tested on a hand-eye-head system for performing vision-guided temporal tasks, such as vision tracking and reaching objects.

5.2 System Architecture

5.2.1 The Hardware Setup

A real robot called SAIL was assembled, as shown in Fig. 5.1. We enumerate the hardware components as follow:

1. A Pentium II 333 dual-processor dual-bus CPU with 512 MB RAM and an internal 27 GB three-drive disk array. This allows a real-time memory recall and update as well as real-time effector controls.

2. ViewSonic ViewPanel VPA138 14 inches LCD display. The display is both flag and light. These features make the robot platform easy to carry the display.

3. Two Panasonic GP-KS152 industrial Color 1/2 inch CCDs with auto gain control and auto white balance.

4. A Matrox video card with two-way splitter for image capture and digitization.

5. Two Citizen M938 3.8 inches LCD Color video monitors.

6. An AccelePort 8e multiport serial adapter from Digi International Inc.. This allows the host extend extra 8 serial communication ports.

7. Two Pan-Tilt Units (PTU) from Directed Perception, Inc. to control pan and tilt motion of two eyes (CCD cameras). A C Programming Interface (PTU-PCI) is used for fast binary communication between the PTU controller and the host computer.

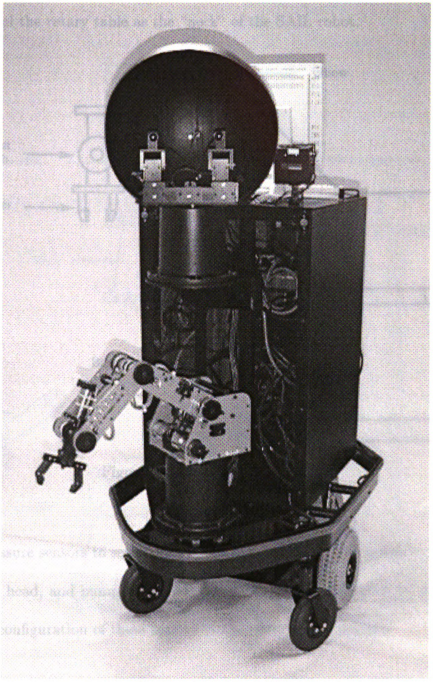


Figure 5.1: The SAIL robot built at the Pattern Recognition and Image Processing Laboratory at Michigan State University.

8. A five-joint SCORBOT-ER III Robot Arm from Eshed Robotec, as shown in Fig. 5.2. The controller can control two additional motors. We use one to control the rotary table as the "neck" of the SAIL robot.

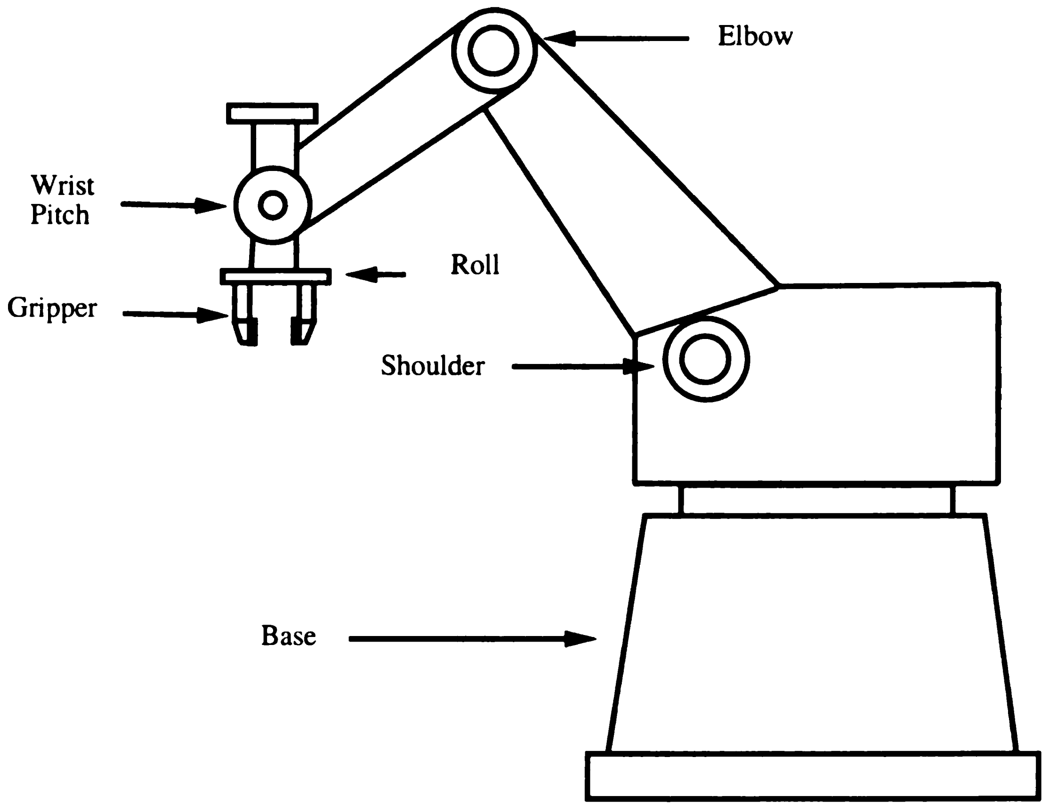


Figure 5.2: The Eshed robot arm.

9. 4 pressure sensors to sense push actions and force. 28 touch sensors on its arm, neck, head, and bumper to allow human to teach how to act by direct touch. The configuration of these sensors is described in Appendex C.

10. An automatic power system APS 750 from TRIPP LITE power protection to switch between heavy-duty batteries and line power.

The system diagrams are shown in Fig. 5.3, 5.4, and 5.5.

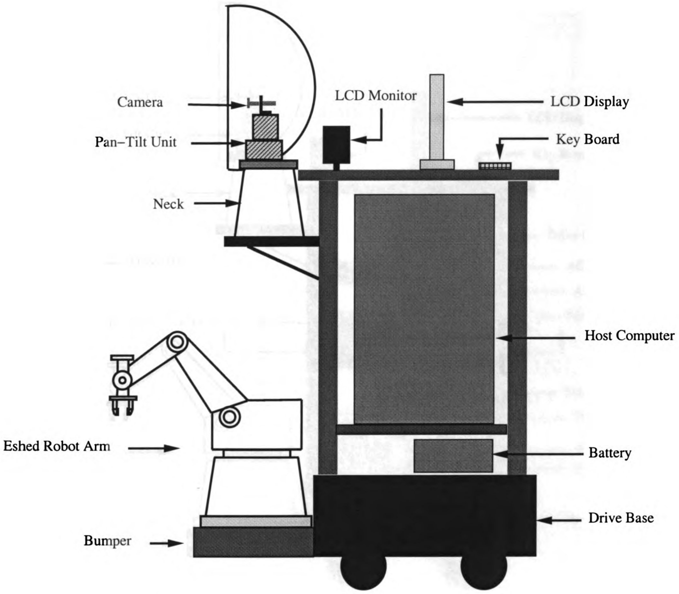


Figure 5.3: The system diagram of the SAIL robot (right side view).

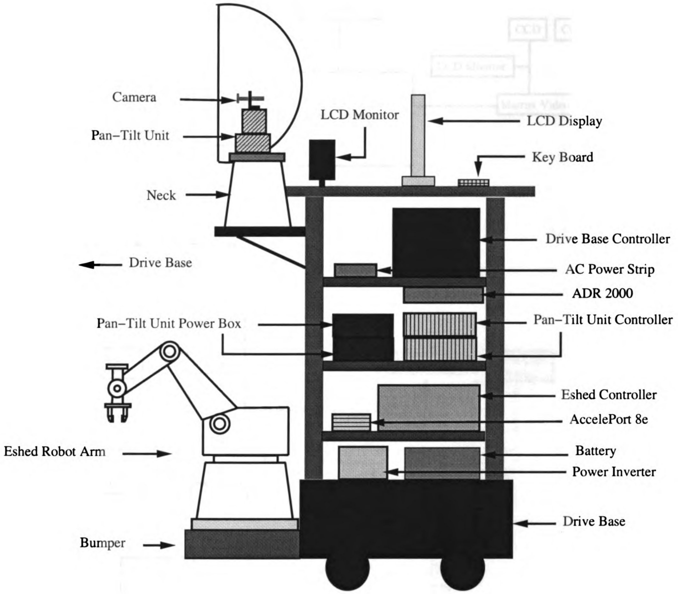
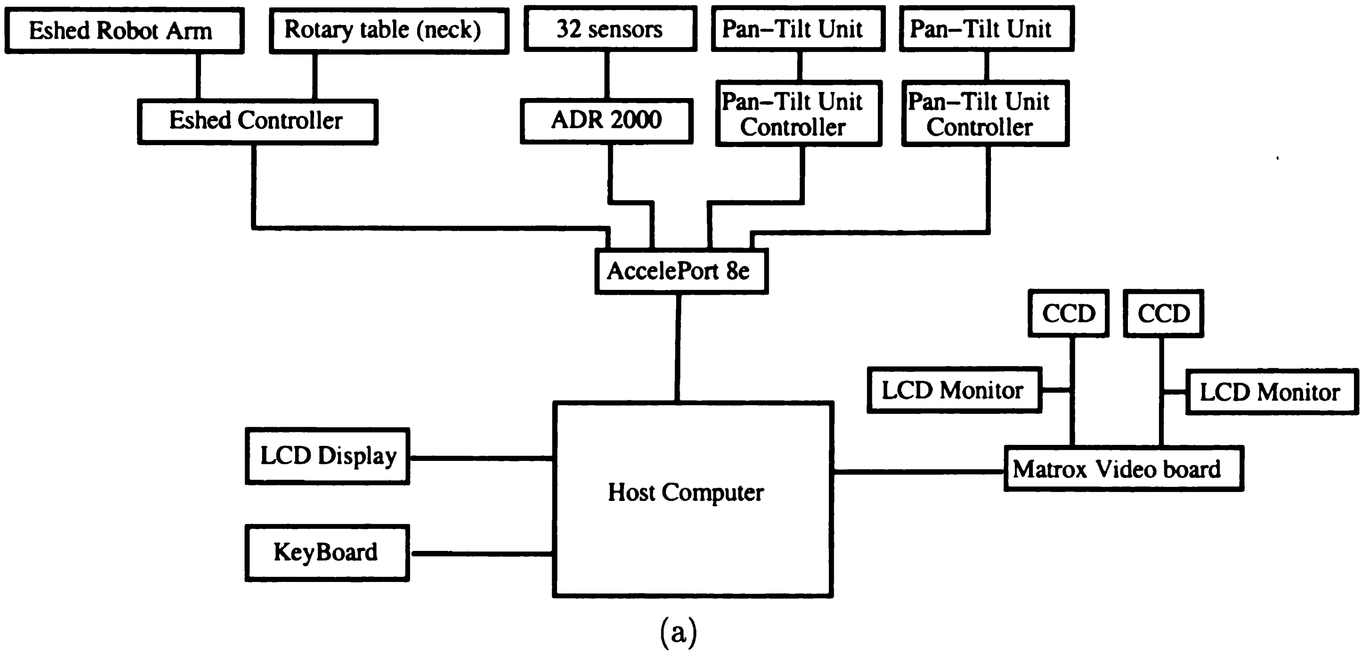


Figure 5.4: The system diagram of the SAIL robot (left side view).



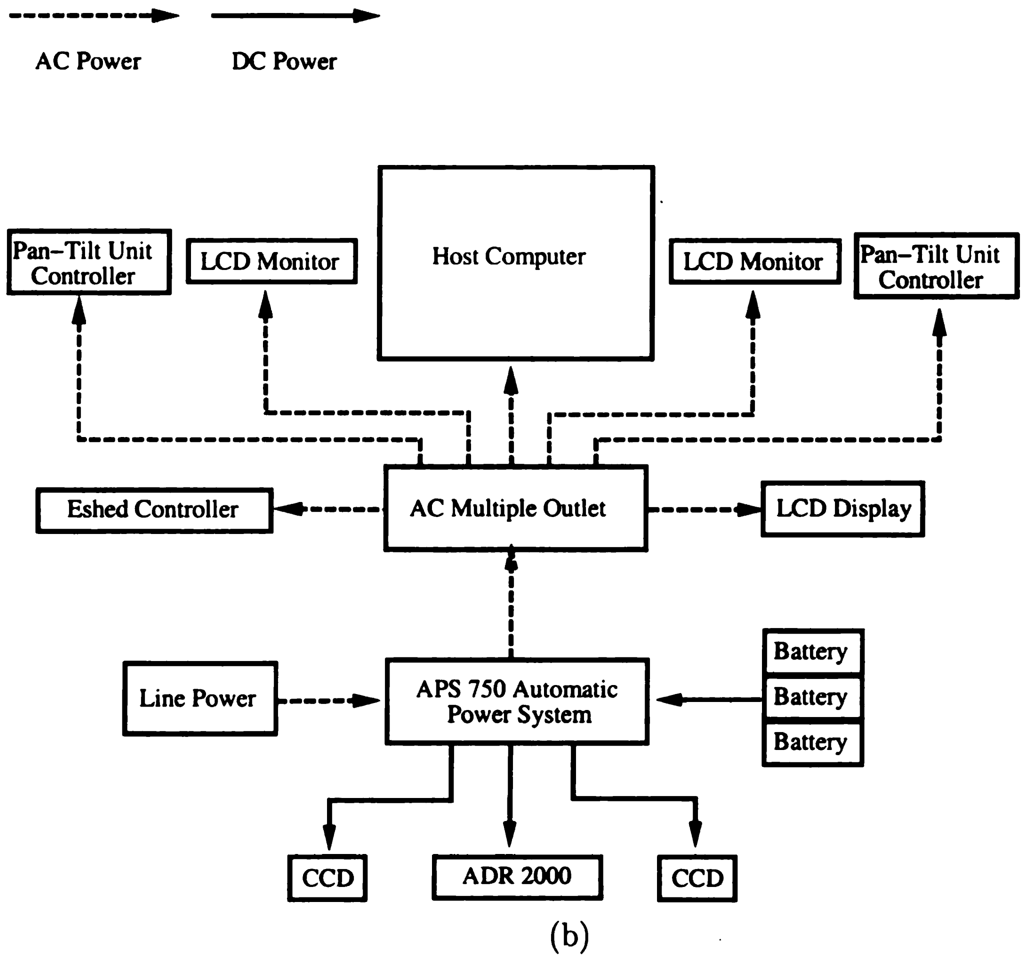


Figure 5.5: The system diagram of the SAIL robot. (a) hardware connection. (b) power connection.

5.2.2 The Software Setup

The programs were developed using Microsoft Foundation Class (MFC) for Graphic User Interface (GUI) and multiple threads processes. The multiple thread processes are illustrated in Fig. 5.6.

Instead of having four threads continuously querying the status of sensory inputs, our system collects the information whenever is needed. It first spawns four threads to acquire 1) the current positions of the Pan-Tilt Units, 2) the 28 touch sensors and four pressure sensors, 3) the current joint positions of the arm, and 4) the stereo images. The main loop of the program then starts from checking the completion of these threads. Whenever a thread from one sensory input is finished, the main loop then spawns a thread for that device. When all the information is acquired, the brain process runs and then goes to the other iteration. Currently the output of the brain process consists of two actions: the movement of the robot arm and the PTU. This is trig from the brain process by spawning the robot arm action thread and PTU action thread. This design not only saves the system resources by query from demand but also solves the synchronization problem of multiple threads process for a real time system.

The brain process takes four sensory inputs and two action feedback inputs. It generates two action outputs. The original image input gives 480×640 resolution with three color bands. The input dimension of the original stereo images is as high as $480 \times 640 \times 3 \times 2 = 184320$ bytes. This is a very high dimensional input and is not feasible for a real time system under the current CPU speed. Instead, we only use

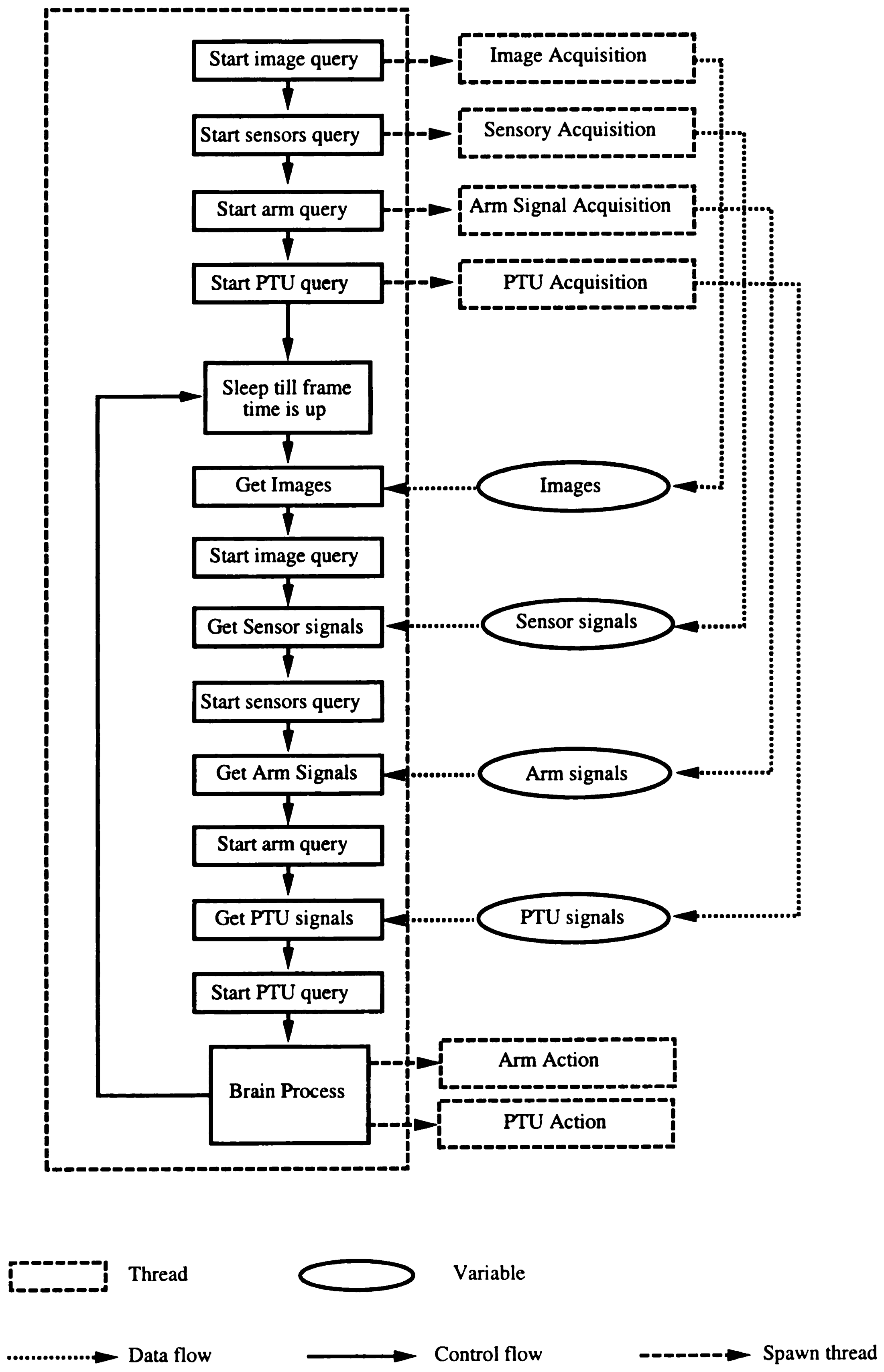


Figure 5.6: The flow diagram of the system. Dashed line box indicates a separate thread. Oval box means thread. Solid line is for control flow and dashed line represents data flow. Dot line indicates spawning a thread. Frame time is up every 100 ms.

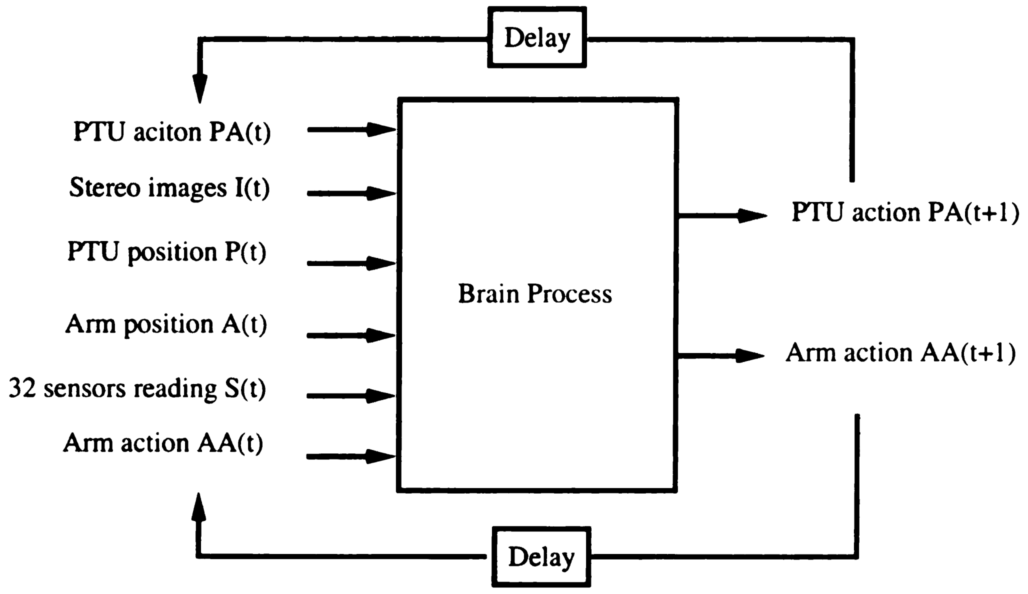


Figure 5.7: The inputs/outputs of the brain process.

region of interest (ROI) of the stereo images I_L and I_R for the left and right images with $r \times c$ resolution, respectively. The ROI is further down-scaled to a factor f, $f \leq 1.0$. The dimension of the image input then is $DIM(I) = r \times c \times 3 \times 2 \times f$. The choice of the system parameters (r,c,p) depends on the tasks given. For example, if the robot arm wants to pick up a cup in front of it, it needs $f = 1$ with small (r, c). If we want to do navigation, we need to have $r = 480$ and $c = 640$ with a small f to cover the whole views and the attention might need to take care of part of the images with a large f for locations like doors or crossings.

We have two PTUs to control the motion of two "eyes". Each PTU reports its current pan and tilt position. The inputs from PTUs are $P = (P_L, P_R)$ with $P_L = (pan_L, tilt_L)$ and $P_R = (pan_R, tilt_R)$, where pan_L and $tilt_L$ are the pan and tilt position of the left PTU. The input dimension from PTU thus is four.

The robot arm has five joints and one gripper. The inputs from the position of the robot arm are $A = (j_1, j_2, \ldots, j_6)$, where $j1$ is the base position, $j2$ is the shoulder position, $j3$ is the elbow position, $j4$ is the roll position, $j5$ is the pitch position, and $j6$ is the gripper position.

There are 28 touch sensors and 4 pressure sensors for the sensory input. The usage of these sensory data is task dependent. In the current design, 20 touch sensors are used. There are four touch sensors $T_{lp} = (T_{lpl}, T_{lpr}, T_{lpu}, T_{lpd})$ to control the left PTU. The touch sensors to control the left PTU to turn left and right is called T_{lpl} and T_{lpr}. The touch sensors to control the left PTU to tilt up and down is called T_{lpu} and T_{lpd}. Similar to left PTU, the right PTU has four touch sensors to control the motion: $T_{rp} = (T_{rpl}, T_{rpr}, T_{rpu}, T_{rpd})$. Each robot arm joint has two touch sensors to control the motion. Thus the touch sensors for the arm joints are: $T_{aj} = (T_{j1l}, T_{j1r}, T_{j2u}, T_{j2d}, T_{j3u}, T_{j3d}, T_{j4u}, T_{j4d}, T_{j5l}, T_{j5r})$. Like the naming of the PTU touch sensors, u is for "up", d is for "down", l is for "left", and r is for "right". For the gripper, (T_{j6c}, T_{j6o}) are the touch sensors to control the open and close motion. The left front pressure sensor T_{lf} is used to specify the task type for the current implementation. The left rear pressure sensor T_{lr} tell the system in the learning mode and the right rear pressure sensors T_{rr} is used to trig the system in the testing mode. The sensory reading for the brain process thus is $S = (T_{lp}, T_{rp}, T_{aj}, T_{j6c}, T_{j6o}, T_{lf}, T_{lr}, T_{rr})$.

The brain process generates two action outputs for the current implementation. One for the robot arm $AA(t)$ and the other for the PTUs $PA(t)$. The values of these actions do not represent the amounts of the control signals. Every action only takes one of the three values $(-1, 0, 1)$. For example, the action of the joint angle 1 (base) of the robot arm is 1 if we want it to move left. A value -1 for the base action means turn to right and 0 for stop motion. Such a design enables us to perform real time training and testing. This is very different from the "snap shot" approach which trains the system to achieve the goal state with the current observation without any feedback control signals sent in between.

Besides the inputs from the hardwares, the brain process also takes action feedbacks from PTU $PA(t-1)$ and arm $AA(t-1)$ as inputs. These feedbacks can both make the input features more distinguishable and stabilized the action sequences, as shown in Fig. 5.8.

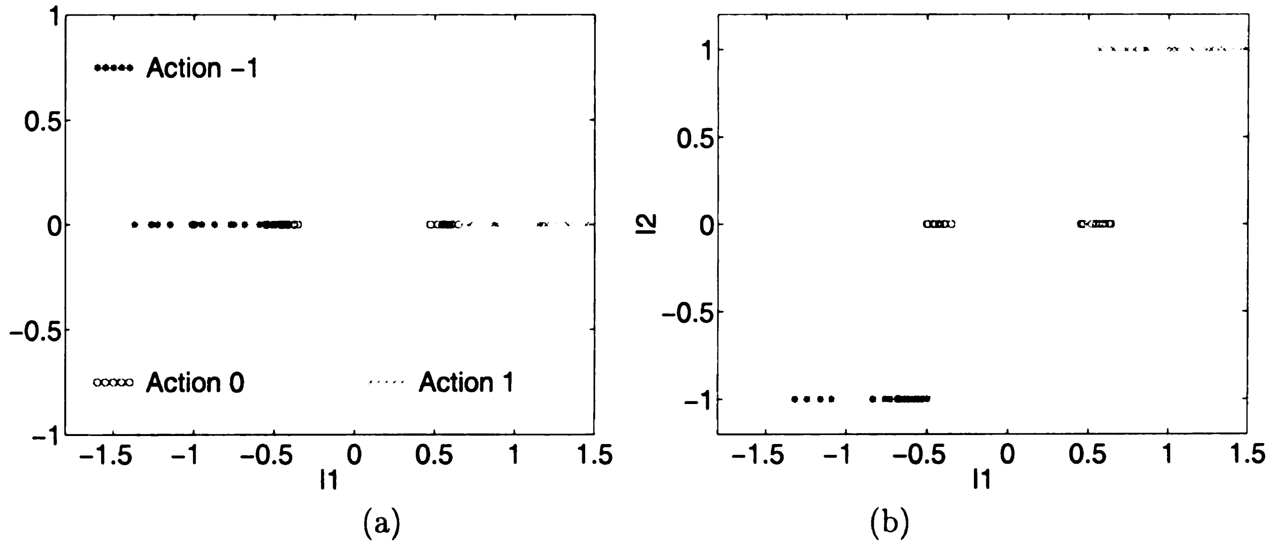


Figure 5.8: (a) A system without feedback. The x axis means the input I_1. The training samples are mixed up on the boundary of the action 0 the other two actions. The system will behave dangling between action 0 and the other two actions. (b) Use feedback as input to the system. The x axis means the input I_1 and y axis is the feedback input I_2. The distance between action 0 and the other two actions are pulled away. When the system outputs action 0 from action 1, it will remain in action 0 unless I_1 changes a lot.

The brain process has a general IHDR core with a task interpreter module. The IHDR tree incrementally mapping from high dimensional input space to action space as described in Chapter 3. The task interpreter module specifies the meaning of the sensor inputs. For example, a touch on the front left pressure sensor means the robot is doing task one. A touch on the rear left pressure sensor means start to train. In the training mode, the robot will takes both the input $In(t) = (TSK(t), I(t), A(t), P(t), PA(t-1), AA(t-1))$ and the corresponding output $Out(t) = PA(t), AA(t))$ to build the IHDR tree, where TSK(t) is the type of the task interpreted by the task interpreter module. In the test mode, the robot gives the current input $In(t) = (TSK(t), I(t), A(t), P(t), PA(t-1), AA(t-1)).)$ to query the IHDR tree and outputs the actions $Out(t) = PA(t), AA(t))$.

Since each input component has different dimensionality ranging from hundreds (image input) to one (e.g. task type), we need to normalize each component so that each component is comparable. The input to IHDR tree thus is a long vector :

$$X(t) = (w_{tsk}TSK(t), w_I I(t), w_A A(t), w_P P(t), w_{PA} PA(t-1), w_{AA} AA(t-1))$$

.

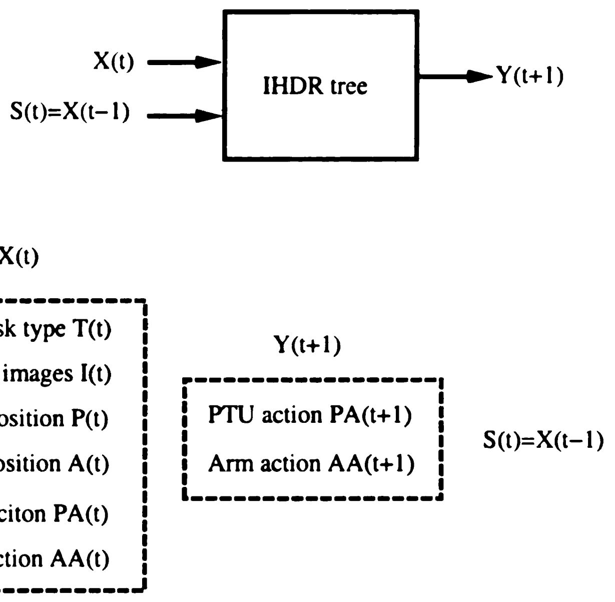


Figure 5.9: The input/output of the IHDR tree.

5.3 Experimental Results

We performed several experimental on the SAIL robot. Since tracking objects and reaching objects are sensorimotor behaviors first developed in early infants, we trained our SAIL robot for two tasks. In the first task, called finding-ball task, we trained the SAIL robot to find a nearby ball and then turn eyes to it so that the ball is located on the center of sensed image. In the second task, called pre-reaching task, we trained the SAIL robot to reach for the object once it has been located and the eyes fixate on it.

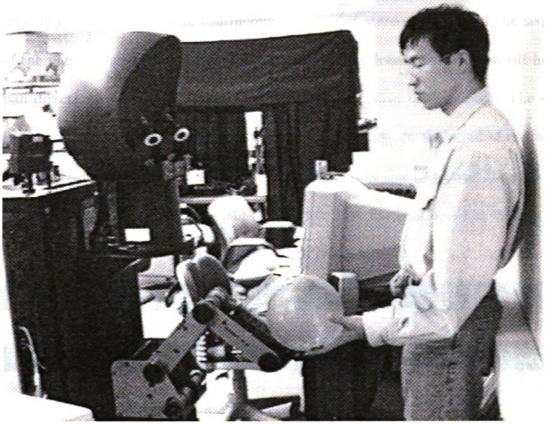


Figure 5.10: Real-time training and testing for SAIL robot: finding the nearby ball and then reaching for it.

Existing studies on visual attention selection are typically based on low-level saliency measures, such as edges and texture [8]. In Birnbaum's work [10], the visual attention is based on the need to explore geometrical structure in the scene. In our case, the visual attention selection is a result of past learning experience. Thus, we do not need to define any task-specific saliency features. It is the SAIL robot that automatically derive most discriminating features for the tasks being learned. At the time of learning, the ball was presented in the region of interest (ROI) inside the stereo images. The human trainer interactively pulls the robot's eyes toward the ball (through the touch sensors for the pan-tilt heads) so that the ball is located on the center of the region of ROI (fixating the eyes on the ball). The inputs to the developmental algorithm are the continuous sequence of stereo images and the sequence of the pan-tilt head control signal. Three actions are defined for the pan-tilt head in pan direction: 0 (stop), 1 (move to the left), or -1 (move to the right). The size of ROI we chose for this experiment is defined as 120×320. In the mind of trainer, the ROI is divided into five regions so that each region is of size 120×64. The goal of the finding-ball task, is to turn the pan-tilt head so that the ball is at the center region. Fig. 5.11 shows some example images for the tracking task.

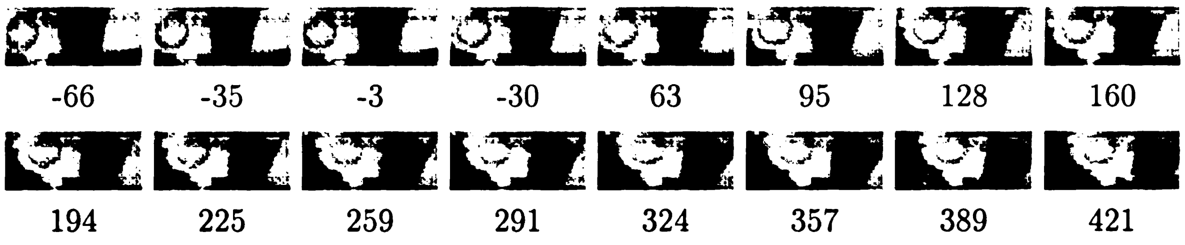
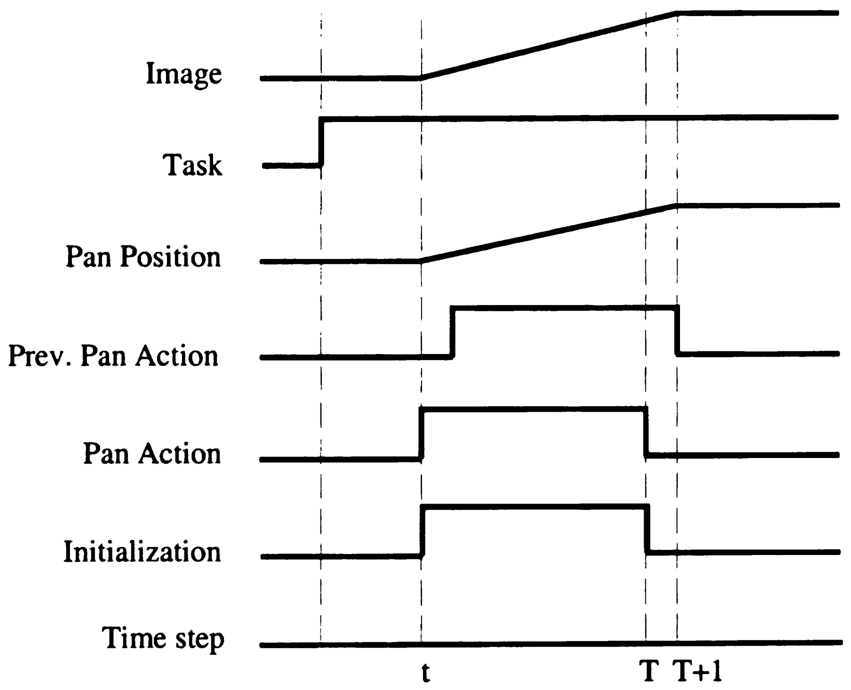
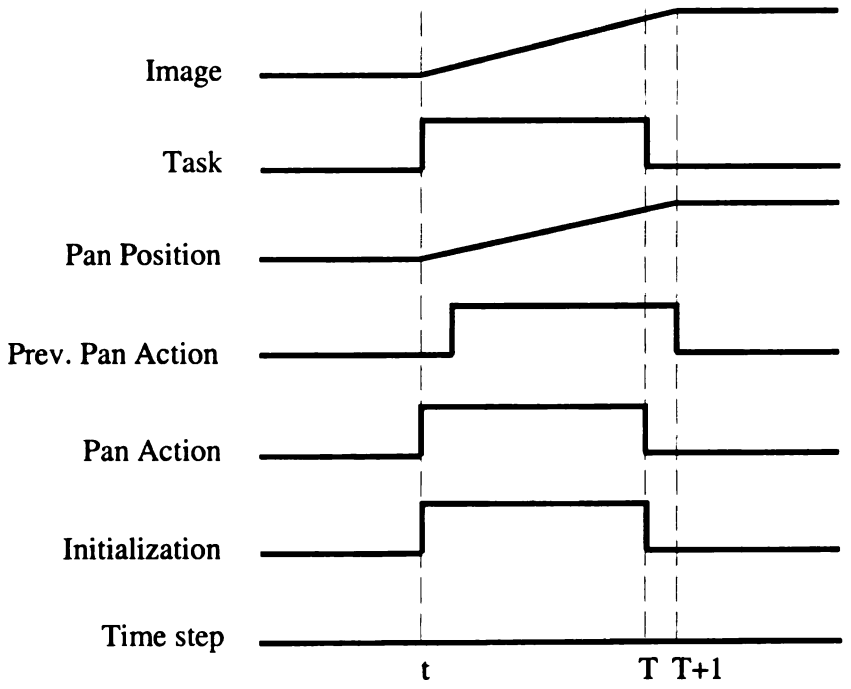


Figure 5.11: A subset of images used in the tracking problem. The number right below the image shows the PTU position associated with that image. From left to right, one image sequence of ball-tracking is shown.

The scenarios for training and testing are shown in Fig. 5.13 and Fig. 5.12. The



(a)



(b)

Figure 5.12: The transitions of sensors/effectors for ball tracking problem. (a) The training session. (b) The testing session.

transitions during the training session are described below:

1. The task input is initiated by pushing a pressure sensor of the robot (or key in a letter using keyboard) before imposing action to pan the camera. The action of the pan is zero at this time since there is no action is imposed.

2. The action of the pan is imposed at time t. The initialization flag is on at the same time. The main program issues a pan thread to pan the camera.

3. The PTU starts to pan and the pan position as well as the image change. Note at t+1 the previous pan action is zero.

4. When the ball is at the fixation of the view at time T, we stopped impose action by issuing a stop thread to the PTU and the initialization flag is off.

5. At time T+1, the PTU stopped moving the image did not change anymore. It is worth noting that the pan action are all zero after time T-1.

Similarly, the testing session can be explained at the follow:

1. We push a pressure sensor to give the task input as well as turn the initialization flag on at time t.

2. The action of the pan is given by query the IHDR tree. A non-zero action is expected according to the trained examples.

3. The PTU and the image starts to move.

4. When the ball is at the fixation of the view at time T, the query result of the IHDR is a zero action. The program issues a stop thread to the PTU and the initialization flag is off.

5. At time T+1, the PTU stopped moving the image did not change anymore.

If we did not use the context (previous pan action) as input, the image and the pan position will be very similar at the point which the action should stop. This will make the PTU stop and go at the boundary point. As explained in Fig. 5.8, the additional context is necessary to make the system stable.

To better understand the task performed, we first used the off-line training-testing mode to analyze the behavior of the algorithm. In the training mode, we positioned the ball in five different position in the ROI and turned the PTU to locate the ball in each session. Two sessions were trained for the system. In the test mode, we presented the ball in 7 different position in the ROI. We trained the system first used two training sessions and then followed by a test session. The training and testing were ran for 10 epochs.

Fig. 5.14(a) shows the memory usage of the program. In the first stage, the tree grows since the samples are accumulated in the root node. When the sample buffer is saturated, the memory curve goes to flat. At the end of the second epoch (runs 19 and 20), the tree started to grow child nodes and the memory start to increase again. In the run 21 and 22, the forgetting process was initiated and the memory still slowly grows because the second level nodes were accumulating samples. Note at this point the tree had one root node and two child nodes. From run 23 to run 28, the second

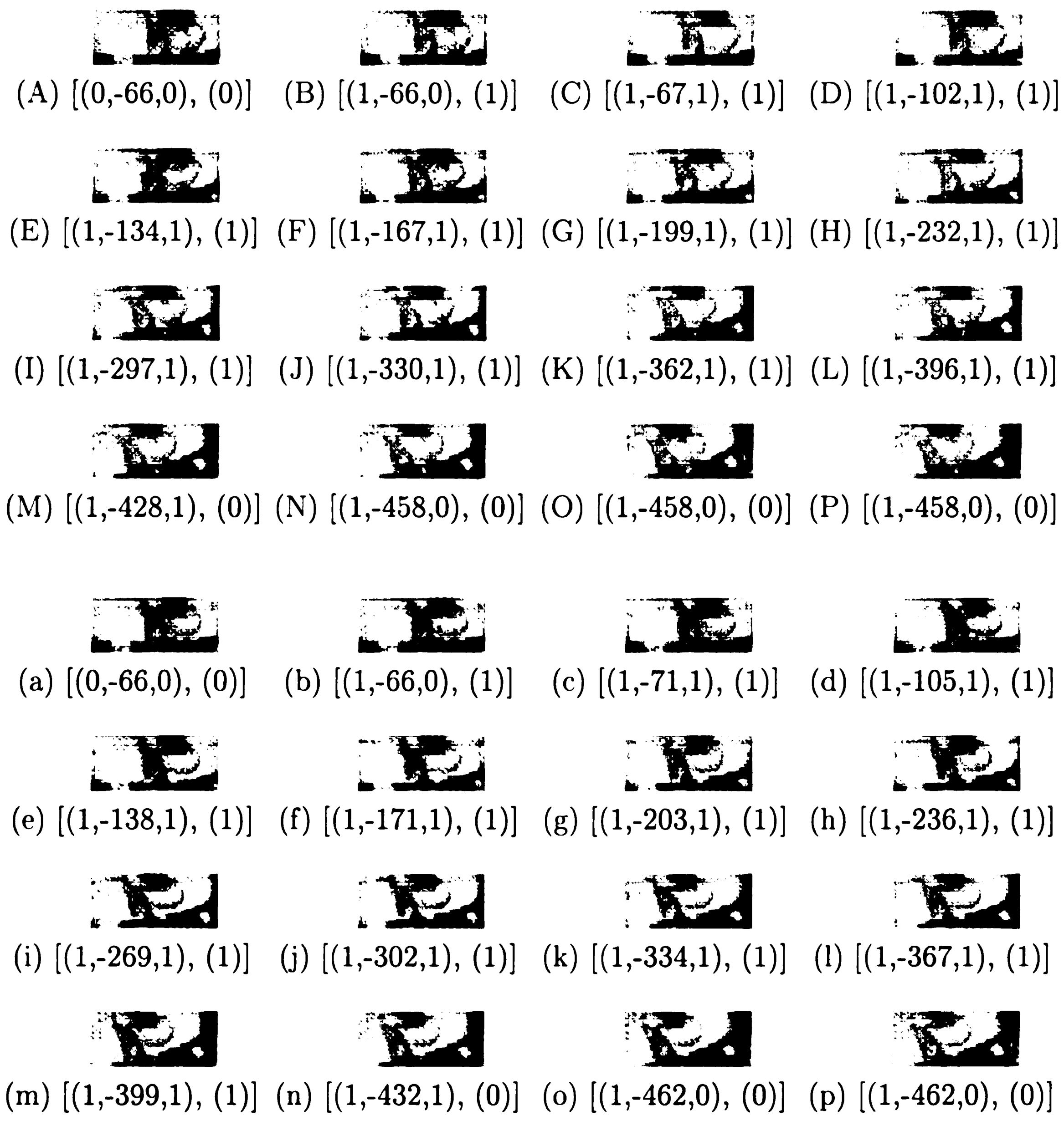


Figure 5.13: An illustration of how the training and testing sequence match. The first four rows show a sequence of training samples, from (A) to (P). The related sensors/effector is shown for each sample. The sensory inputs shown are the image I(t) and In(t)=(TSK(t), P(t), PA(t-1)). The output Out(t) is the action of the pan PA(t). Under each image shown the associated inputs and outputs [In(t), Out(t)]. The last four rows shows a sequence of testing samples.

level nodes are accumulating samples. Run 29 to run 36 the samples are saturated and the memory curve stay flat. At run 37, the third node of the second level were spawned. The memory grows till the run 48. The memory then saturated and keep flat afterward.

Fig. 5.14(b) shows the performance of the program for the test session. After the 3-rd epoch, the systems can achieve the desired performance to locate the ball in the center of ROI. Fig. 5.14(c) gives the plot of the average CPU time for each action sequence. The average CPU time for each action sequence is within the mental cycle time: 100 millisecond. Since the IHDR tree is dynamically updated, all the forgetting, updating, and rejecting processes occurred during each sequence. The average CPU time for each sequence is up and down, depending on the given training examples.

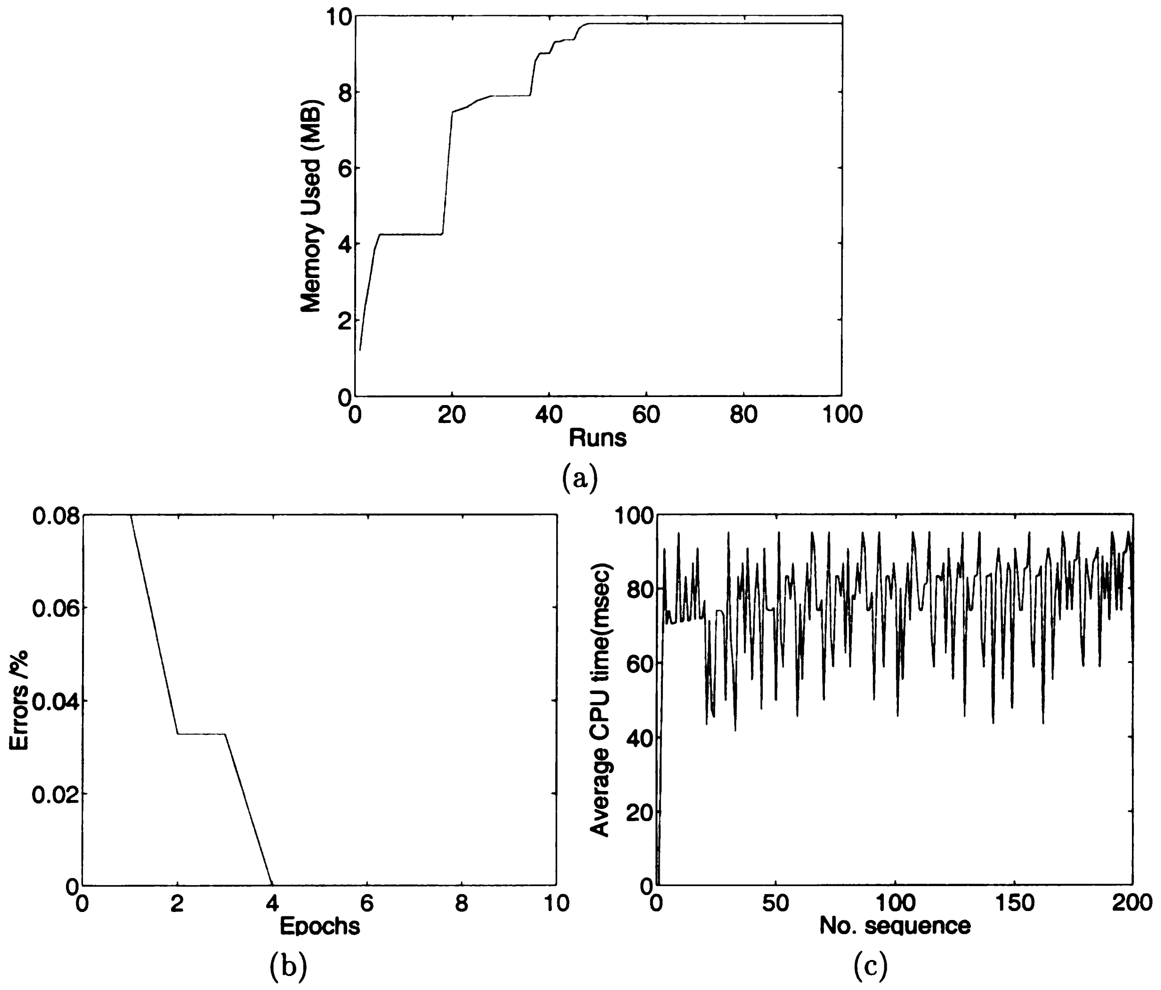


Figure 5.14: (a) The memory usage for the off-line simulation of ball tracking task. (b) The performance of the testing. (c) The average CPU time for each sequence.

Chapter 6

Conclusions

6.1 Summary

A new algorithm is first introduced to cast both classification and regression problems into a unified regression framework. This allows us to design the new doubly-clustered method. Clusters in output space provide coarse-to-fine virtual class labels from clusters in the input space. To deal with high-dimensional input space, a different discriminating subspace is automatically derived at each internal node of the tree. A size-dependent probability-based distance metric SDNLL is proposed to deal with large sample cases, small sample cases, and unbalanced sample cases. Our experimental study with synthetic data showed that the method can achieve near-Bayesian optimality for both low dimensional data and high dimensional data with low dimensional data manifolds. With the help of the new decision tree, the retrieval time for each sample is of a logarithmic complexity. The output of the system can be both class label or numerical vectors, depending on how the system trainer gives the training data. The experimental results have demonstrated that the algorithm can deal with a wide variety of sample sizes with a wide-variety of dimensionality.

The incremental building of the HDR tree opens up the possibility of real-time interactive training where the number of training samples can be extremely large but the resulting tree does not need to store all the training samples. Thanks to the amnesic average and the forgetting process, the computational complexity is very low compared with the previous batch version of HDR tree. We have also demonstrated the performance of the incremental version of the HDR tree is comparable to the batch version. The experiments on the vision-based navigation show the feasibility of the IHDR tree algorithm applied on the regression problem.

A new way of dealing with recognition tasks is proposed. The recognition of an object is treated as a dynamic, online, incremental, continuous learning process. The system learns while performing. The programming stage does not require humans to specify content-specific rules. The goal is to relieve humans from the tedious task of directly controlling the internal representation. No reprogramming is required to train the system for more cases, more types of input, and even new tasks for new domains. This is because our framework does not require humans to provide a problem space. Of course, the idea of active vision [4] [18] calls for the dynamic aspect. The autonomous learning concept introduced here is a further advance.

The application on the robotics develops an robot which can interacts with the environment in real-time. A technical challenge to perform such a task is the mapping engine that scalable — keeping real-time speed and a stable performance for a very large number of high dimensional vector data. With the proposed powerful mapping engine, the robot is able to operate in real time. A new style of programming multi-sensory multi-effector real-time system is implemneted. Every thread is finished within each memory cycle. The proposed algorithm has successfully run on the SAIL robot for real-time interactive training and testing for two sensori-motor tasks: finding ball and reaching the centered ball, two early tasks that infants often learn to perform.

6.2 Future Work

The proposed HDR tree algorithm provide an efficient way to deal with both classification and regression problems. However, there are several issues remain to be explored:

- The theoretic proof of the algorithm. A formal proof of the algorithm is desired to better understand the behavior and the limit of the algorithm. Although the synthetic data simulation in Section 2.5 shows the HDR tree algorithm reaches near optimality of Bayesian bound, a further study of the theoretic bound can give more precise insight of the algorithm and help to choose the suitable applications.

- More applications and larger data set to test the algorithm. Several experiments have bee conducted for the proposed algorithm including synthetic data, real image data, data with manually extracted features. Since what we proposed in a general framework for both classification and regression problem, we would like to test more applications of the algorithm. To make others easy to try our algorithms for their own application, the software can be accessed in the web page: *http : //www.cse.msu.edu/rgroups/amdl/hdrt.html*. In the face recognition problem, we would like to apply our algorithm to larger databases.

- Improvement of the decision tree. The proposed HDR tree algorithm does not handle problem with missing data attribute values. Methods in [64] [90] [89] can be used to deal with this problem. Besides, if the cost matrix is provided for a classification problem, the current implemented HDR tree did not utilize this information to obtain the discriminating features. Some data mining techniques such as boosting [20] can also be incorporated into the HDR tree algorithm.

Similar to the HDR tree, the future research directions for the incremental HDR tree are:

- A more comprehensive study of the dynamic behaviors of the proposed algorithm. We would like to see further analysis of the parameters used in the algorithm and the theoretic bound for the method.

- We have tested the algorithm in the simulation of the vision-based navigation problem. It is interesting to see the algorithm applied to the real robot and compare with other existing systems [99] [73].

- Improve the decision tree by using other data mining techniques.

For the robotic application, the future directions are as follow:

- Longer context: The current tasks trained does not need much context. We would like to have a longer context for the future work.

- Integration of short and long temporal context: We would like to have a mechanism for automatic spatiotemporal level building.

- Non-monolithis sensory processing: All vision, speech, and touch sensory inputs have similar receptive field mechanisms.

- Intramodal and intermodal attention: We would like to deal with both intramodal (e.g., visual attention) and intermodal attention (selection which sensory inputs are important.)

- Mechanisms of self-generated intention: This is accomplished in motivational system in the human central nervous system.

APPENDICES

Appendix A

Linear manifold

Given a set of vectors $V = \{v_1, v_2, ..., v_n\}$ which is a subset of a vector space $\mathcal{X}$. We want to express the subspace that passes the head tips of the vectors in V.

For numerical stability, we use the center of the vectors in V,

$$\bar{v} = \frac{1}{n}\sum_{i=1}^{n} v_i$$

and define the set of scatter vectors from their center: $s_i = v_i - \bar{v}$, $i = 1, 2, ..., n$. These n scatter vectors are not linearly independent because their sum is equal to a zero vector. Let S be the set that contains these scatter vectors: $S = \{s_i \mid i = 1, 2, ..., n\}$. The subspace spanned by S, denoted by span(S), consists of all the possible linear combinations from the vectors in S.

A translation of a subspace is called a linear manifold [66] (also called linear variety [51]). The subspace M translated to vector v_0 is denoted by $v_0 + M$: $v_0 + M = \{v_0 + m \mid m \in M\}$. Thus, the subspace that passes the head tips of the vectors in S

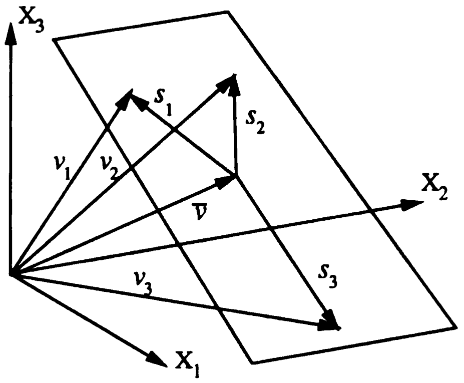


Figure A.1: The linear variety (hyper plane) that passes through the head points of the vectors. It can be represented by $\bar{v}$ + span(S), the spanned space from scatter vectors translated by the center vector $\bar{v}$.

can be represented by the linear manifold $\mathcal{D} = \bar{v} + \text{span}(S)$, as shown in Fig. A.

The orthonormal basis $a_1, a_2, ..., a_{n-1}$ of the subspace span(S) can be constructed from the radial vectors $s_1, s_2, ..., s_n$ using the *Gram-Schmidt Orthogonalization* (GSO) procedure:

Procedure 7 GSO Procedure: *Given vectors $s_1, s_2, ..., s_{n-1}$, compute the orthonormal basis vectors $s_1, s_2, ..., s_{n-1}$.*

1. $a_1 = s_1/||s_1||$.
2. *For $i = 2, 3, ..., n-1$, do the following*
   (a) $a'_i = s_i - \sum_{j=1}^{i-1}(s_i^T a_j)a_j$.
   (b) $a_i = a'_i/||a'_i||$.

In the above procedure, a degeneracy occurs if the denominator is zero. In the first step, the generacy means s_1 is a zero vector. In the remaining steps, it means that the corresponding vector s_i is a linear combination of the previous radial vectors. If a degeneracy occurs, the corresponding s_i should be discarded in the basis computation.

The number of basis vectors that can be computed by the GSO procedure is the number of linearly independent radial vectors in S.

Given a vector $x \in \mathcal{X}$, we can compute its scatter part $s = x - \bar{v}$. Then compute the projection of x onto the linear manifold. It's i-th coordinates in the orthonormal basis is given by $\beta_i = s^T a_i$, $i = 1, 2, ..., n-1$. We call the vector $f = (\beta_1, \beta_2, ..., \beta_{n-1})^T$ the feature vector of x in the linear manifold S.

Appendix B

Cholesky Decomposition

Procedure 8 Cholesky factorization: *Given an* $n \times n$ *positive definite matrix* $A = [a_{ij}]$, *compute lower triangular matrix* $L = [l_{ij}]$ *so that* $A = LL^T$.

```
For $i = 1, 2, ..., n$ do
1. For $j = 1, 2, ..., i - 1$ do
        $l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}) / l_{jj}$
2. $l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$
```

Appendix C

Sensor System of SAIL Robot

C.1 Communication specification

The serial A/D board can be connected to the serial port of the host computer through the following protocol:

1. Baud rate: 38,000.
2. parity: No.
3. data bits: 8.
4. stop bits: 1.
5. No software or hardware handshaking is needed.

C.2 The A/D board

The sensor system consists of (32) 12 bit analog input channels with a range of 0 to 5 Volts. This corresponds to a digital range of 0X000 to 0XFFF in hexadecimal, or 0 to 4095 in decimal notation.

The 32 inputs are multiplexed into an 8 channel A/D converter using hardwired analog multiplexers. Each of the 8 lines has a 50 K ohm pull up resistor to +5V DC. A short on any line should produce an analog reading between 0.5 volts that gives a decimal reading below 400. An open should produce a voltage above 4.5 volts which gives a decimal reading above 3600.

Four of the analog inputs are connected to resistive force sensors which have a static resistance around 50K ohms. This produces a voltage input on these analog channels of about 2.5 volts which corresponds to a decimal reading around 2000. Pressure applied to the resistive force sensor will cause its resistance to change about 10% to 20%, giving a decimal reading change of about 300.

The command sent to the serial A/D board to cause it to scan all 32 ports is a "q" followed with a carriage return and line feed. The serial A/D will scan all 32 ports and return 32 4 character decimal values with space separators and a carriage return plus line feed terminator string. The output is arranged to two rows with each row consists of 16 port readings, as listed in the Table C.1.

Table C.1: The output of the SAIL sensor system

port	first row	second row
1	left eye left	left eye down
2	neck left	neck right
3	right eye left	right eye down
4	case left rear sensor	case right rear sensor
5	arm switch 1 (gripper) down	arm switch 1 (gripper) up
6	arm switch 3 (pitch) down	arm switch 3 (pitch) up
7	arm switch 4 (elbow) down	arm switch 4 (elbow) up
8	arm switch 6 (base) down	arm switch 6 (base) up
9	left eye right	left eye up
10	bumper front	bumper right front
11	right eye right	right eye up
12	case right front sensor	case left front sensor
13	arm switch 2 (roll) down	arm switch 2 (roll) up
14	bumper right rear	bumper rear
15	arm switch 5 (shoulder) up	arm switch 5 (shoulder) down
16	bumper left rear	bumper left front

Bibliography

[1] Alan D. Christiansen, Mattew T. Mason, and Tom M. Mitchell. Learning Reliable Manipulation Strategies without Initial Physical Models. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 1224–1230. IEEE, 1990.

[2] Hamid Alavi. State self-organization for continuous video stream input, a master's project report. Technical report, Department of Computer Science, Michigan State University, 1997.

[3] Allen, P.K. and A. Timcenko and B. Yoshimi and P. Michelman. Automated Tracking and Grasping of a Moving Object with a Robotic Hand-Eye System. *IEEE Trans on Robotics and Automation*, 9(2):152–165, Apr 1993.

[4] J. Aloimonos. Purposive and qualitative active vision. In *Proc. 10th Int'l Conf. Pattern Recognition*, pages 346–360, Atlantic City, NJ, June 1990.

[5] J. R. Anderson. *Cognitive Psychology and Its Implications.* Freeman, New Worky, third edition, 1990.

[6] M. H. Ashcraft. *Human Memory and Cognition.* Harper Collins College Publishers, New Royk, NY, 1994.

[7] Martin Bichsel. *Strategies of Robust Object Recognition for the Automatic Identification of Human Faces.* PhD thesis, Eidgenössischen Technischen Hochschule Zürich, 1991. Diss. ETH Number 9467.

[8] Martin Bichsel. *Strategies of Robust Object Recognition for the Automatic Identification of Human Faces.* Swiss Federal Institute of Technology, Zurich, Switzerland, 1991.

[9] Billibon H. Yoshimi and Peter K. Allen. Visual Control of Grasping and Manipulation Tasks. In *ARPA Image UnderstandingWorkshop*, pages 1151–1157, Morgan Kaufmann, San Francisco, CA, Oct 1994.

[10] Lawernce Birnbaum, Matthew Brand, and Paul Cooper. Looking for Trouble: Using Causal Semantics to Direct Focus of Attention. In *Proc of the IEEE Int'l Conf on Computer Vision*, pages 49–56, Berlin, Germany, May 1993. IEEE Computer Press.

[11] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Chapman & Hall, New York, 1993.

[12] R. Brooks. Intelligence without reason. In *Proc. Int'l Joint Conf. on Artificial Intelligence*, pages 569–595, Sydney, Australia, August 1991.

[13] R. A. Brooks. Symbolic reasoning among 3-D models and 2-D images. *Artificial Intelligence*, 17(1-3):285–348, Aug. 1981.

[14] R. A. Brooks. A robots layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[15] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[16] S. Carey. *Conceptual Change in Childhood.* The MIT Press, Chambridge, MA, 1985.

[17] Castanõ, A. and S. Hutchinson. Hybrid Vision/Position Servo Control of a Robotic Manipulator. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 1264–1269, Los Alamitos CA, May 1992. IEEE Computer Society Press.

[18] T. Darrell and Alex Pentland. Active gesture recognition using partial observable Markov decision processes. In *Proc. 10th Int'l Conf. Pattern Recognition*, Vienna, Austria, 1996.

[19] G. R. Dattatreya and L. N. Kanal. Decision tress in pattern recognition. In L. Kanal and A. Rosenfeld, editors, *Progress in Pattern Recognition*, pages 189–239. Elsevier Science, New York, NY, 1985.

[20] H. Drucker, C. Cortes, L.D. Jackel, Y. Lecun, and V. Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301, 1994.

[21] Feddema, J.T. and O.R. Mitchell. Vision-Guided Servoing with Feature-Based Trajectory Generation. *IEEE Trans on Robotics and Automation*, 5(5):691–700, Oct 1989.

[22] S. Franklin. *Artificial Minds.* MIT Press, Cambridge, MA, 1997.

[23] J. H. Friedman. A recursive partition decision rule for nonparametric classification. *IEEE Trans. on Computers*, 26:404–408, April 1977.

[24] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, New York, NY, second edition, 1990.

[25] G. H. Golub and C. F. van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, 1989.

[26] Graf, D.H. and W.R. LaLonde. A Neural Controller for Collision-Free Movement of General Robot Manipulators. In *Int'l Conf on Neural Networks*, volume 1, pages 77–84, San Diego CA, Jul 1988. IEEE.

[27] W. Eric L. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints.* The MIT Press, 1990.

[28] I. Harvey. Evolutionary robotic and SAGA: The case for hill crawling and tournament selection. Technical Report CSRP 222, University of Sussex, Brighton, U.K., 1992.

[29] E. G. Henrichon, Jr. and K. S. Fu. A nonparametric multivariate partitioning procedure for pattern classification. *IEEE Trans. on Computers*, 18:614–624, July 1969.

[30] D. H. Hubel. *Eye, Brain, and Vision.* Scientific American Library, Distributed by Freeman, New York, 1988.

[31] W. Hwang, S. J. Howden, and J. Weng. Performing temporal action with a hand-eye system using the SHOSLIF approach. In *Proc. Int'l Conference on Pattern Recognition*, Vienna, Austria, Aug. 1996.

[32] W. Hwang and J. Weng. Vision-guilded robot manipulator control as learning and recall using SHOSLIF. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, Albuquerque, NM, April 1997.

[33] W. Hwang and J. Weng. Autonomous vision-guided robot manipulation control. In *Proc. of Asian Conference on Computer Vision*, pages 503–510, Hong Kong, China, January 1998.

[34] W.-S. Hwang, J. Weng, M. Fang, and J. Qian. A fast image retrieval algorithm with automatically extracted discriminant features. In *Proc. IEEE Workshop on Content-based Access of Image and Video Libraries*, pages 8–15, Fort Collins, Colorado, June 1999.

[35] K. Ikeuche and T. Kanade. Automatic generation of object recognition programs. *Proceedings of the IEEE*, 76(8):1016–1035, 1988.

[36] Jr. J. L. Martinez and R. P. Kessner (eds.). *Learning & Memory: A Biological View.* Academic Press, San Diego, CA, 2 edition, 1991.

[37] Jr. J. R. Deller, Jone G. Proakis, and John H. L. Hansen. *Discrete-Time Processing of Speech Signals.* Macmillan, New York, NY, 1993.

[38] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, New Jersey, 1988.

[39] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[40] S. K. Murthy S. Kasit and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence*, 2:1–33, Aug. 1994.

[41] Kieffer, S. and V. Morellas and M. Donath. Neural Network Learning of the Inverse Kinematic Relationships for a Robot Arm. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 2418–2425, Sacramental CA, Apr 1991. IEEE.

[42] M. Kirby and L. Sirovich. Application of the karhunen-loève procedure for the characterization of human faces. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(1):103–108, Jan. 1990.

[43] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, second edition, 1988.

[44] T. Kohonen. *Self-Organizing Maps.* Springer-Verlag, Berlin, second edition, 1997.

[45] D. J. Kriegman and J. Ponce. On recognizing and positioning curved 3-D objects from image contours. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(12):1127–1137, 1990.

[46] J. E. Laird, E. S. Yager, M. Hucka, and C. M. Tuck. Robo-soar: An integration of external interaction, planning, and learning using Spar. *Robotics and Autonomous Systems*, 8:113–129, 1991.

[47] Y. Lamdan and H. J. Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. In *Proc. 2nd Int'l Conf. Computer Vision*, pages 238–249, 1988.

[48] D. B. Lenat. Cyc: A large-cale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[49] D. B. Lenat, G. Miller, and T. T. Yokoi. Cyc, wordnet, and edr: Critiques and responses. *Communications of the ACM*, 38(11):45–48, 1995.

[50] E. Lloyd. *Handbook of Applicable Mathematics, Volume VI, Part A, Statistics.* J. W. Arrowsmith Ltd, Bristol, 1984.

[51] D. G. Luenberger. *Optimization by Vector Space Methods.* Wiley, New York, 1969.

[52] Marcos Salganicoff and Ruzena K. Bajcsy. Robotic Sensorimotor Learning in Continuous Domain. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, Los Alamitos CA, May 1992. IEEE.

[53] Martinetz, R.M. and H.J. Ritter and K.J. Schulten. 3D-Neural-Net for Learning Visuomotor-Coordination of a Robot Arm. In *Int'l Jt Conf on Neural Networks*, volume 2, pages 351–356, Jun 1989.

[54] M. J. Mataric. A distributed model for mobile robot envrionment - learning and navigation. Technical Report AI-TR-1228, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1990.

[55] A. K. McCallum. Reinforcement learning with selective perception and hidden state. Technical report, PhD thesis, University of Rochester, Rochester, New York, 1996.

[56] Michael S. Branicky. Task-Level Learning: Experiments and Extensions. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 266–271, Sacramental CA, Apr 1991. IEEE.

[57] D. Michie, D.J. Spiegelhalter, and C.C. Taylor (eds). *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[58] G. A. Miller. Worknet: A lexical basebase for English. *Communications of the ACM*, 38(11):39–41, 1995.

[59] H. P. Moravec. The Stanford Cart and the CMU Rover. *Proceedings of the IEEE*, 71(7):872–884, 1982.

[60] H. Murase and S. K. Nayar. Illumination planning for object recognition in structured environments. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, pages 31–38, Seattle, WA, June 1994.

[61] H. Murase and S. K. Nayar. Visual learning and recognition of 3-D objects from appearance. *Int'l Journal of Computer Vision*, 14(1):5–24, January 1995.

[62] S. K. Murthy. Automatic construction of decision trees from data: A multidisciplinary survey. *Data Mining and Knowledge Discovery*, 1998.

[63] S. K. Nayar, H. Murase, and S. A. Nene. Learning, positioning, and tracking visual appearance. In *Proc. IEEE Conf. on Robotics and Automation*, May 1994.

[64] M. Nunez. The use of background knowledge in decision tree induction. *Machine Learning*, 6(3):231–250, 1991.

[65] J. Yamato J. Ohya and K. Ishii. Recognizing human action in time-sequential images using hidden Markov model. In *IEEE Trans. Pattern Analysis and Machine Intelligence*, pages 379–385, 1992.

[66] E. Oja. *Subspace Methods of Pattern Recognition.* Research Studies Press, Letchworth, UK, 1983.

[67] Eitetsu Oomoto and Katsumi Tanaka. OVID: Design and implementation of a video-object database system. *IEEE Trans. Knowledge and Data Engineering*, 5(4):629ff, August 1993.

[68] Papanikolopoulos, N.P. and P.K. Khosla and T. Kanade. Adaptive Robotic Visual Tracking. In *Proc of the 1991 American Control Conference*, pages 962–967, Jun 1991.

[69] Papanikolopoulos, N.P. and P.K. Khosla and T. Kanade. Visual Tracking of a Moving Target by a Camera Mounted on a Robot: A Combination of Control and Vision. *IEEE Trans on Robotics and Automation*, 9(1):14–35, Feb 1993.

[70] Alex Pentland, Baback Moghaddam, and Thad Starner. View-based and modular eigenspaces for face recognition. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, pages 84–91, June 1994. Seattle, Washington.

[71] P. J. Phillips, H. Moon, P. Rauss, and S. A. Rizvi. The FERET evaluation methodology for face-recognition algorithms. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, pages 137–143, San Juan, Puerto Rico, June 1997.

[72] P. J. Phillips, H. Moon, P. Rauss, and S. A. Rizvi. The FERET september 1996 database and evaluation procedure. In *Proc. Int'l Conf. on Audio and Video-based Biometric Person Authentication*, Cranse-Montana, Switzerlan, March 1997.

[73] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky, editor, *Advances in Neural Information Processing*, volume 1, pages 305–313. Morgran-Kaufmann Publishers, San Mateo, CA, 1989.

[74] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes.* Cambridge University Press, New York, 1986.

[75] M. L. Puterman. *Markov Decision Processes.* Wiley, New York, NY, 1994.

[76] J. Quinlan. Introduction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[77] J. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufman, San Mateo, CA, 1993.

[78] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE*, 77(2):257–286, 1989.

[79] L. R. Rabiner, L. G. Wilpon, and F. K. Soong. High performance connected digit recognition using hidden Markov models. *IEEE Trans. Acoustics, Speech and Signal Processing*, 37:1214–1225, Aug. 1989.

[80] V. S. Ramachandran. Perceiving shape from shading. In I. Rock, editor, *The Perceptual World*, pages 127–138. Freeman, San Francisco, CA, 1990.

[81] P. Rauss, P. J. Phillips, M. K. Hamilton, and A. T. DePersia. FERET (face-recognition technology) recognition algorithms. In *Proc. of Automatic Target Recognizer System and Technology Symposium*, July 1996.

[82] S. R. Safavin and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Trans. Systems, Man and Cybernetics*, 21(3):660–674, May/June 1991.

[83] C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schlkopf, and A. Smola. Support vector machine reference manual. Technical Report CSD-TR-98-03, Royal Holloway, University of London, Egham, UK, March. 1998.

[84] Schrott, A. Feature-Based Camera-Guided Grasping by an Eye-in-Hand Robot. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, pages 1832–1837, Los Alamitos CA, May 1992. IEEE Computer Society Press.

[85] Sharma, R. and Y. Aloimonos. Visual Motion Analysis Under Interceptive Behavior. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 148–153, Champaign, IL, June 1992. IEEE.

[86] T. Starner and A. Pentland. Visual recognition of American sign language using hidden Markov models. In *Proc. Int'l Workshop on Automatic Face- and Gesture-Recognition*, pages 189–194, Zurich, Switzerland, June 1995.

[87] D. Swets and J. Weng. Discriminant analysis and eigenspace partition tree for face and object recognition from views. In *Proc. Int'l Conference on Automatic Face- and Gesture-Recognition*, pages 192–197, Killington, Vermont, Oct. 14-16 1996.

[88] D. L. Swets and J. Weng. Using discriminant eigenfeatures for image retrieval. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 18(8):831–836, 1996.

[89] M. Tan. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, 13(1):1–33, 1993.

[90] M. Tan and J. C. Schlimmer. Two case studies in cost-sensitive concept acquisition. In *AAAI-90*, 1990.

[91] P. Thompson. Margaret thatcher: a new illusion. *Perception*, 9:483–484, 1980.

[92] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[93] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, 1995.

[94] Walter, J.A. and K.J. Schulten. Implementation of Self-Organizing Neural Networks for Visuo-Motor Control of an Industrial Robot. *IEEE Trans on Neural Networks*, 4(1):86–95, Jan 1993.

[95] J. Weng. On comprehensive visual learning. In *Proc. NSF/ARPA Workshop on Performance vs. Methodology in Computer Vision*, pages 152–166, Seattle, WA, June 1994.

[96] J. Weng. Cresceptron and SHOSLIF: Toward comprehensive visual learning. In S. K. Nayar and T. Poggio, editors, *Early Visual Learning.* Oxford University Press, New York, 1996.

[97] J. Weng and S. Chen. Incremental learning for vision-based navigation. In *Proc. Int'l Conference on Pattern Recognition*, Vienna, Austria, Aug. 1996.

[98] J. Weng and W. Hwang. An incremental learning algorithm with automatically derived discriminating features. In *Proc. of Asian Conference on Computer Vision*, Taipei, Taiwan, January 2000. accepted and to appear.

[99] Juyang Weng and Shaoyun Chen. Vision-guided navigation using SHOSLIF. *IEEE Trans. Neural Networks*, 11:1511–1529, 1998.

[100] S. S. Wilks. *Mathematical Statistics.* Wiley, New York, NY, 1963.

[101] Yasuo Kuniyoshi, Masayuki Inaba, Hirockika Inoue. Seeing, Understanding and Doing Human Task. In *Proc of the IEEE Int'l Conf on Robotics and Automation*, volume 1, pages 2–9, Nice, France, May 1992.

[102] Yasuo Kuniyoshi, Masayuki Inaba, Hirockika Inoue. Learning by Watching: Extracting Reusable Task Knowledge from Visual Observation of Human Performance. *IEEE Trans on Robotics and Automation*, 10(6):799–822, Dec 1994.