This is to certify that the

dissertation entitled

Learning-Based Vision and Its Application to Autonomous Indoor Navigation

presented by

SHAOYUN CHEN

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science

Major professor

Date 12/14/98

Learning-Based Vision and Its Application to Autonomous Indoor Navigation

By

Shaoyun Chen

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

1998

Abstract

Learning-Based Vision and Its Application to Autonomous Indoor Navigation

By

Shaoyun Chen

Adaptation is critical to autonomous navigation of mobile robots. Many adaptive mechanisms have been implemented, ranging from simple color thresholding to complicated learning with artificial neural networks (ANN). The major focus of this thesis lies in machine learning for vision-based navigation. Two well known vision-based navigation systems are ALVINN and ROBIN developed by Carnegie-Mellon University and University of Maryland, respectively. ALVINN uses a two-layer feedforward neural network while ROBIN relies on a radial basis function network (RBFN).

Although current ANN-based methods have achieved great success in vision-based navigation, they have two major disadvantages: (1) Local minimum problem: The training of either multilayer perceptron or radial basis function network can get stuck at poor local minimums. (2) The flexibility problem: After the system has been trained in certain road conditions, it is hard to make the system adapt to new road conditions while retaining good performance for those road conditions that have already been learned. Sometimes this is termed a "memory loss" problem. As part of our SHOSLIF (Self-organizing Hierarchical Optimal Subspace Learning and Inference Framework) effort, SHOSLIF-N (SHOSLIF for Navigation) treats vision-based navigation as a content-based retrieval problem. Three major components of SHOSLIF-N are: (1) Automatic feature derivation: Instead of starting with random initial weights, the system employs either principle component analysis or linear discriminant anal-

ysis to derive features which are best suited for navigation tasks. (2) Nonparametric recursive partitioning regression, which is more flexible than global parametric regression used in either ALVINN or ROBIN, is employed in direct input-to-output mapping. Nonparametric recursive partitioning regression is realized with a recursive partition tree (RPT). (3) Self-organizing mechanism. (4) Low computational complexity: the recursive partition tree has a logarithmic retrieval complexity and can be used to address the complexity issue in learning a large number of scenes.

For a binary RPT, only the most dominant eigenvector of principle component analysis or linear discriminant analysis is needed for further partitioning of each inner node. This leads to an efficient online incremental learning algorithm: the system learns or rejects a learning sample "on-the-fly" with real time response. Similar to ALVINN and ROBIN, the basic SHOSLIF-N maps a single-framed retinal input into an output steering signal. The system was successfully tested but exhibited limited capability in handling more complicated situations. When the number of different turns or corners was increased to a certain extent, the system sometimes failed to make the turn. One way to tackle this problem is to incorporate state information, that indicates the relative position between the robot and the oncoming corner or intersection, into the system. Therefore, state-based SHOSLIF-N, a system that incorporates states and utilizes a simple yet efficient visual attention mechanism which is helpful in determining the correct state transitions, is proposed and tested. With a set of fewer than 300 learning samples, state-based SHOSLIF-N has been successfully tested in indoor navigation on the 2nd and 3rd floors of our Engineering Building. Using a SUN Sparc-I and a framegrabber, both online incremental learning and autonomous navigation were done in real-time. Comparative study with two ANN-based methods has shown the advantages of the system: faster learning and better performance for the tasks tested.

ACKNOWLEDGMENTS

I am indebted to my advisor, Professor John J. Weng, who helped me choose the thesis topic and has been an inexhaustive source of ideas throughout my Ph.D. program at Michigan State University. Without his support, this dissertation would not have been completed. I am grateful to Professors Anil K. Jain, George C. Stockman, R. Lal. Tummala, Bang-Yan Chen, and Jim Zacks, for serving on my committee and providing critical comments on my dissertation. I would like to acknowledge my special thanks to Dr. Jain and Dr. Stockman, for their advice on my research topic.

I enjoyed many helpful discussions with former and current members working on SHOSLIF-related projects, especially Yuntao Cui, Dan Swets, and Wey-Shyan Hwang. I received numerous help from our nice PRIP lab managers: Lisa Lees, Hans Dulimarta, Chitra Dorai, Sally Howden, Karissa Miler, Paul Albee, and Jason Sperber. I would like to thank former Prippies who built our mobile robot Rome which was used throughout my tests. Here, I would like to express my special gratitude to Hans Dulimarta, who provided bug-free programs for robot control.

I benefited from many discussions with former and present PRIP members. Thanks go to: Paul Albee, Hamid Alavi, Vera Bakic, Chaur-Chin Chen, Jinlong

Chen, Sei-Wang Chen, Yao Chen, Scott Connell, Nicolae Duta, Kamen Guentchev, Qian Huang, Lin Hong, Marie-Pierre Jolly, Yonghong Li, Jinhui Liu, Jianchang Mao, Yogesh Pathak, Sharath Pankanti, Nalini Kanta Ratha, Prasun Sinha, Yarle Strand, Daniel Swets, Aditya Vailaya, Oivind Due Trier, Marilyn Wulfekuhler, Shanti Vedula, Gang Wang, Bin Yu, and Yu Zhong.

I owe special thanks to my former advisor Prof. Xueyin Lin at Tsinghua University, P. R. China, for his guidance in vision-based navigation, which is closely related to my dissertation topic.

Finally, I sincerely express my gratitude to my parents and my grandmother for their love and encouragement throughout my life. I would like to thank my wife Xueai Fang, who joined me lately, for her support and understanding.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

The goal of this thesis was to develop vision-based learning techniques that can facilitate the training of mobile robots for autonomous indoor navigation. Practical learning algorithms for this purpose should be able to meet the following three requirements: real-time response, incremental learning, and good performance. These three issues are addressed throughout this dissertation.

Mobile robot systems provide a unique opportunity to develop perception and navigation techniques in real-world environments. In order to perceive the external world, a mobile robot may use various kinds of sensors, including color video, scanning laser rangefinder, stereo video, sonar, radar, and inertial navigation, to collect necessary information. Then higher level processing integrates different sensory inputs, and other *a priori* knowledge, such as coarse maps or road models, to direct vehicle actions.

1.1 A Brief Review of Mobile Robot Vision

Visual input is the most important sensory source for human drivers. Thompson [116] and Moravec [82] are among pioneers in exploiting the role of machine vision in autonomous navigation. The JPL robot [116] used visual input to form polygonal terrain models for path construction. The Stanford Cart [82, 83] employed Moravec's interest operator to pick out distinctive features in the images. Three dimensional information of these features was obtained by a correlation-based stereo algorithm. FIDO [117] used a hierarchical correlation for stereopsis and successfully navigated through a cluttered environment and along a sidewalk. Tsuji *et al.* [122] used another stereo vision system based on matching vertical edges and inferring surfaces.

The early mobile robots had very limited vision systems. The launch of the Autonomous Land Vehicle project in 1984, funded by the Defense Advanced Research Projects Agency, led to numerous research activities [120], including motion tracking by Univ. of Massachusetts and Honeywell, tracking using 3D data by SRI, and qualitative navigation by Advanced Decision Systems. Martin-Marietta built its VITS [123], and CMU built the Navlab [118]. The CMU Navlab group concentrated on difficult vision problems in realistic environments and produced many practical algorithms for mobile robot vision. Their research on the Navlab project up to 1990 is well summarized in a book format [119].

Techniques developed for autonomous navigation can be applied not only for strategic defense uses, but also for civilian purposes that can improve the safety of human drivers and significantly lower the number of traffic accidents.

Beyond the DARPA community, the past decade witnessed several other mobile robot projects. A good survey can be found in Thorpe *et al.* [120]. In the united states, General Motors worked on lane-following. The FMC corporation [72] built a road-following system based on color thresholding. Fijitsu and Nissan in Japan have developed prototype road-following software. In Munich, the group led by Dickmanns has been working on autonomous navigation for a numbers of years [25, 28, 29, 30].

1.2 Road Following Techniques

There are two basic tasks in vision-based navigation: road-following and obstacle avoidance. A mobile robot needs to travel in road areas while avoiding obstacles. In general, these two tasks should not be separated. However, the techniques for these tasks vary in practice since the sensor inputs for these two tasks could be different. Direct range measurements coming from sonar or infrared sensors are useful for obstacle avoidance. Sonar or infrared sensors are not useful in outdoor road following, since they need reflecting surfaces which are generally not available in outdoor environment and the obtained signals are not reliable. On the other hand, rich visual information is necessary for both road following and obstacle avoidance. Since sonar or infrared sensory inputs are of low dimension and can be handled with less computation, many obstacle avoidance systems make use of sonar or infrared sensors.

Vision-based road following algorithms can be broadly classified into six categories: (1) Edge-based technique: edges are detected from the input images and used in road

following. (2) Region analysis: Statistical pattern recognition is applied to road segmentation, e.g. clustering of road and nonroad regions; (3) Stereo-based road following: a stereo algorithm is used to obtain the depth information from either edges or regions for road following; (4) 3D perception using laser scanners; (5) ANN-based approaches: ANN is used either to map detected features into qualitative output or directly map visual input into output steering signals; (6) Dickmann's 4D approach: Feedback from tracking objects modeled with spatio-temporal models are used in optimal control of a vehicle. This approach is a good coupling of optimal control with machine vision.

1.2.1 Edge detection

Edge detection was commonly used in early mobile robot navigation systems [53, 131]. The Terregator [125], a predecessor of the Navlab, also detected road edges and tried to find the central line of the path. The general assumption is that the road lies on the ground plane and two road edges are parallel. A Hough transform or other line fitting process is used to find the path.

IPM (Inverse Perspective Mapping) [125, 62, 15] maps an input image into a view with a looking-straight-down view direction. A view generated from IPM is also called a bird's-eye view. Since the road lines of a path on the ground plane appear parallel after IPM transform, IPM simplifies optical flow computation, obstacle detection, and road following. IPM is commonly used in vision-based navigation systems [125, 62, 2].

The problems with this type of road following technique are: (1) The detected

edges may not be useful for navigation, e.g. in outdoor navigation, the strongest edges are usually caused by shadows, the real road edges are weaker and sometimes not detectable. The same situation happens for indoor navigation. The true path edges may not be detectable due to ambient lighting or other distractions from the background. (2) The real road edges do not necessarily fit the parallel lines model and thus the assumption for line fitting does not apply. (3) The road edges are sometimes not available due to occlusions.

Edge detection is more suited for structured roads where lane markings or stripes are available. One way to overcome the limitation of a single line feature tracking algorithm is to explicitly model different available constraints and features of a structured road. YARF [64, 119] takes advantage of prior knowledge of the environment. According to different feature models, different feature tracking algorithms are designed. One feature tracking method will take over if the current feature tracking fails.

Algorithms taking into account more than general edge detection can achieve better performance. The availability of lane markings can be used to greatly enhance the adaptation performance of a driving system. RALPH [96] is a good example. In RALPH possible road curvatures are predicted, and the one with the highest signature score is picked up as the winner. This technique leads to a successful lateral position handler.

1.2.2 Region analysis

An alternative to edge detection is region analysis. A road scene can be classified into road and nonroad regions. This type of algorithm does not rely so heavily as edge-based approaches on the geometrical road model. Different cues, e.g. texture and color, can be used to classify a pixel into road and nonroad. Because of the rich information available in chromatic space, color is widely used in outdoor navigation. Color video is used in several vision-based navigation systems, including the Navlab [118], VITS [123] and FERMI [72]. Both the VITS and FERMI project image pixels on the color space and do a simple adaptive thresholding.

More robust methods are used in the Navlab. Crisman and Thorpe [119] applied statistical pattern recognition techniques to road scene segmentation. SCARF (Supervised Clustering Applied to Road Following) classifies each pixel into road and nonroad. Each class is represented by a Gaussian distribution, of which the mean vector and the covariance matrix are obtained from a supervised learning process. The classifier is a maximal likelihood classifier. SCARF uses reduced-resolution images with 60×64 pixels and achieves real-time performance. A voting scheme using the Hough transform is used to obtain the road model.

UNSCARF (Unsupervised Clustering Applied to Road Following) makes use of spatial color appearance changes. UNSCARF uses a tuple of five components, $\mathbf{x} = \{RED, GREEN, BLUE, \text{row}, \text{column}\}^t$, where the first three are RGB components of a color scene, and the last two are spatial locations of each pixel. The algorithm clusters those pixels with nearby locations and similar color appearances into a class

with an ISODATA clustering algorithm. The appropriate model is chosen from a set of candidate road models based on a global matching of road edges detected from UNSCARF region analysis. The algorithm is suited for unstructured roads. Both SCARF and UNSCARF have been mapped onto the CMU WARP machine with a systolic architecture. Due to the iterative nature of unsupervised learning used in UNSCARF, it is much slower than SCARF.

1.2.3 Stereo-based road following

Stereo provides depth information that can be used for vision-based navigation. Typically, some measures are taken to reduce the computational complexity, either by hierarchical matching or by matching a few feature points or detected edges. The Stanford CART [82] used Moravec interest operator to pick up a set of features which were tracked and correlated between frames to obtain depth information. The algorithm was very slow, it took five hours to navigate a 20 meter course. The FIDO system [117] used a hierarchical correlation of points chosen by an interest operator. Stereo based on matched edges is commonly used in both indoor and outdoor navigation [122, 1, 71, 74]. Stereo algorithms with real-time constraint can only obtain a sparse depth map, which is the major constraint for its application in vision-based navigation.

1.2.4 3D perception

3D range measurements, if reliably obtained in real-time, can be directly used for navigation. In contrast to stereo-based algorithms, 3D range measurements from a laser scanner can provide a dense depth map. 3D range sensing was used in Navlab project [119]. The Navlab used the ERIM (Environmental Research Institute of Michigan) scanner, which measured the phase shift of an amplitude-modulated laser. A laser scanner has certain advantages over more traditional vision techniques such as a stereo algorithm: it is insensitive to ambient illumination. However, the major disadvantage is that the acquisition rate is too low, which limits its use in high-speed navigation. The typical acquisition rate is one frame per second. 3D perception can be used in more places than road following, e.g. it can also be used in offroad navigation. The laser scanner can be used to build a terrain map for cross-country navigation.

1.2.5 ANN-based approaches

Artificial neural networks have been successfully applied in many engineering applications. ANNs have also found their role in vision-based navigation. One effort of using multilayer perceptron is NEURO-NAV, developed by Meng and Kak [81] at Purdue University, for indoor navigation. NEURO-NAV is a system which couples low-level neural network based modules with high-level semantically based planning. The Hough transform was used to detect edges from intensity images. The result was then fed into neural networks to produce a qualitative output: a "near," "far" or

"at" output indicating the relative position to a landmark. These outputs are then directly fed into a planner for higher level planning.

The most well-known system is ALVINN (Autonomous Land Vehicle In a Neural Network) [95] developed by the NAVLAB group at Carnegie-Mellon University. ALVINN is a feedforward multilayer artificial neural network trained by a back-propagation learning algorithm. The system maps input images to output steering signals for autonomous navigation. ALVINN does not explicitly impose the type of image features upon which the system is based. The neural network approach is computationally simple and real-time navigation performance can be achieved. ALVINN has been successfully tested in a variety of road situations.

Extensions to ALVINN include MANIAC (Multiple Alvinn Networks In Autonomous Control) [56], which combines several ALVINN-type networks and allows the vehicle to move smoothly from one type of road to another. ELVIS [44] uses an eigen-subspace method as an alternative to the artificial neural network in ALVINN.

An alternative to ALVINN have been successfully tested: ROBIN [105, 106], developed at Univ. of Maryland and later extended at Lockheed Martin Corporation, uses a radial basis function neural network (RBFN) to map a low-resolution input image into output steering signal. Because of its built-in regularization mechanism, ROBIN achieved smoother behavior than ALVIN did, as shown in [106].

The self-organizing map (SOM), another major component in neural networks, has also been used in autonomous navigation, but only a simulation study [49] has been reported.

1.2.6 Dickmanns' 4D approach

Vision-based navigation is essentially a control problem with visual sensory feedback. The ideal method should be a perfect coupling of optimal control and machine vision techniques. However, not much work has been done to seamlessly couple optimal control with computer vision for navigation. With great effort along this line, Dickmanns' group [26, 31, 32] used Kalman filtering for feedback control with detected visual features as observations. The internal representation is a set of spatio-temporal models of objects. Since the objects are three-dimensional, and temporal information is used, this approach is termed a 4D approach.

The 4D object databases incorporated generic models for objects as background knowledge and parameterized representation with specified ranges for each subclass. The visual appearance is coded as a parameterized distribution around the object center and the aspect conditions. This approach tracks a set of features which are driven by the model, measures the error between predicted features and measured features, which is used as feedback to control the vehicle. The 4D approach employs an extended Kalman filter to estimate internal states. The nonlinear system model is linearized along nominal trajectories. As the computing engines are getting more and more powerful, this state update cycle is getting shorter and shorter and the linearized model better characterizes the true nonlinear system. Dickmanns' group has been exploring this approach since 1986. Early demonstrations have shown its potential. With an inexpensive inertial system added and more powerful computing engines with Pentium Pros, a remarkable performance was demonstrated in 1997 [32].

The success of the 4D approach lies in its efficient use of spatio-temporal information from visual inputs and its seamless coupling with optimal control. The model-driven feature tracking achieves efficiency in real-time navigation by taking advantage of the structure of well-marked roads. On the other hand, this also limits the system's applicability.

1.3 Some Major Difficulties in Autonomous Navigation

As discussed above, significant advances have been made toward autonomous navigation. However, we are still facing some major difficulties.

The first major difficulty lies in dealing with different road types. Existing approaches to autonomous navigation typically impose restrictions on the environment, such as good pavement conditions, clear lane marks, absence of shadows, continuous road edges, etc. Unfortunately, although such requirements may sometimes be met, it is not necessarily always the case. Thus, it is desirable to seek a method that is more general — applicable to a wide range of scenes that may be encountered in real-world driving conditions.

The second major difficulty lies in illumination. The lighting conditions impose a challenge for current vision-based navigation. This is true for both indoor and outdoor navigation, especially outdoor. The road edges can be invisible due to poor lighting, and some false edges caused by shadows and other objects can also fool

the system. For outdoor navigation, the variation in illumination from night time to day time or from cloudy days to sunny days requires a dynamic range which is not achievable with current visual sensors.

The third major difficulty lies in the lack of efficiency. The above generality requirement means that the system must learn a huge number of scenes, e.g., a few hundred to a few million. Almost all the existing autonomous navigation systems have significant limitations on the type of applicable scenes, with one exception: ALVINN (and its extensions), which virtually does not impose any restriction on the scene type and thus, is capable of learning any scene. However, it still has several problems. A major one is a lack of capability to handle a large number of scene types. Other major ones include the local minima problem caused by the use of artificial neural networks, and the slow learning problem which provides little possibility for on-line learning.

1.4 Adaptive and Learning Mechanisms: Direct Input-to-Output Mapping

The amount of road condition variation is a crucial factor in determining the applicability of a system. To handle a variety of environmental conditions, most experimental navigation systems have employed certain adaptation mechanisms to various degrees. For navigation on paved driveways, when lane marks are well illuminated and easily traceable, a simple technique, such as adaptive color thresholding [72], can be used. In

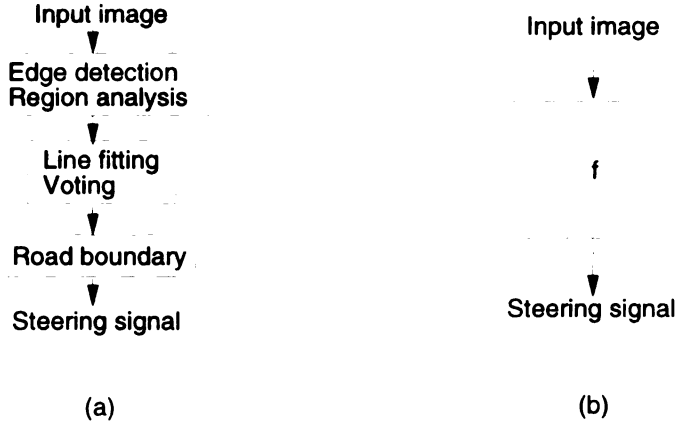


Figure 1.1: Differences between traditional edge-based or region-based approach and direct input-to-output mapping. (a) Traditional approach; (b) Direct input-to-output mapping, where the function f directly maps input image into output steering signal.

situations with unstructured roads with unmarked lanes and sudden changes in illumination (e.g, the sun is occluded by clouds, or shadows cast from nearby buildings), the design of a reliable autonomous navigation system becomes a great challenge. As mentioned in the earlier section, the Navlab group developed various adaptation mechanisms for various outdoor driving environments.

ALVINN [94, 120] and ROBIN [105, 106] directly maps preprocessed input images into output steering signals for autonomous navigation. Direct input-to-output mapping is very different from traditional edge-based or region-based approaches. A schematic difference between direct input-to-output mapping and traditional edge-based or region-based method is shown in Fig. 1.1. In edge-based or region-based approach, edges or regions are first extracted and later used in subsequent image processing steps to obtain information required for vehicle steering. The extracted edges or regions may not be sufficient for making decisions in autonomous navigation. The direct input-to-output mapping uses the whole preprocessed image as input

which has richer information than edge map or region segmentation. It also bypasses intermediate steps of processing edge maps or segmented regions into higher level information for navigation. The whole input-to-output mapping is treated as a global optimization process. Therefore, if properly done, direct input-to-output mapping is expected to achieve more reliability and efficiency.

1.5 Differences between SHOSLIF-N and Two Neural Network Approaches

The work presented here is a navigational part of the Self-organizing Hierarchical Optimal Subspace Learning and Inference Framework (SHOSLIF) [133].

The navigation system SHOSLIF-N described here was motivated by the following considerations.

- Automatic feature derivation which minimizes the imposition of restrictions on the type of the scene the system will handle.

  The framework automatically derives the most expressive features (MEF) and the most discriminating features (MDF), which respectively maximize representation and classification powers from information available in the input data. Therefore, the method is potentially applicable to a wide variety of conditions.

- Nonparametric recursive partitioning regression.

  Nonparametric recursive partitioning regression[37], which is more flexible than global parametric regression used in either ALVINN or ROBIN, is used in direct

input-to-output mapping. Nonparametric recursive partitioning regression is realized using a recursive partition tree (RPT).

- Self-organizing mechanism.

  The approach uses a *self-organizing* mechanism for systemwise learning, which is responsible for mapping input images to vehicle control signals.

- Low computational complexity.

  The recursive partition tree has a logarithmic retrieval complexity and can be used to address the complexity issue in learning a large number of scenes. Due to this low complexity, the experimental system Rome is capable of real-time navigation while computation is performed by an on-board SUN SPARC-1 without using any special image processing hardware other than a frame grabber.

A schematic difference between our SHOSLIF-N and current neural network approaches is shown in Fig. 1.2. Current neural network approaches use an explicit function f to directly map a visual input I into output signal O. Here f is represented by a linear or nonlinear function of linear or nonlinear combination of first and second layer neurons. The adjustable parameters are weights of the first and the second layers in a feedforward network, and widths and centers of first layer and weights of second layer for a RBF network. In SHOSLIF-N, a novel input I is used to retrieve its k nearest neighbors in the learned samples. Then the associated outputs of k samples are interpolated to obtain the actual output. Notice that there is a task decomposition here: the visual input is used in content-based retrieval, while associated

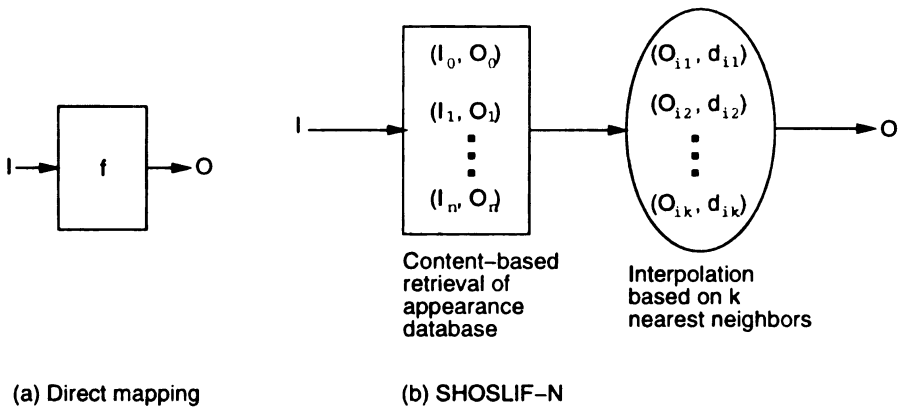


Figure 1.2: Schematic difference between current neural network approaches for vision-based navigation and SHOSLIF-N. (a) Direct input-to-output mapping using neural networks. (b) SHOSLIF-N.

outputs are weighted according to their distances to current input in the interpolation process to get the desired output. Direct input-to-output mapping in Fig. 1.2(a) can be regarded as a global approximation trying to minimize the global sum of square error between target outputs and simulated network outputs. SHOSLIF-N can be regarded as a piecewise approximation of input-to-output mapping based on learning samples. Better and better approximations can be achieved by adding more and more learning samples. On the other hand, more space is required for storing these samples.

SHOSLIF-N uses local adaptive nonparametric regression, which is more flexible than global parametric regression. Recursive partitioning provides a good tradeoff between speed and performance. Projection pursuit regression [52] is another well known adaptive algorithm for function approximation, but with very high computational complexity. Friedman [37] provides an excellent discussion on this subject.

1.6 Our Mobile Robot ROME

Our mobile robot, RoME (Robotic Mobile Experiment), was built by former students Steve Walsh, Jon Courtney and Hans Dulimarta. The major hardware components consist of:

1. A Sun SPARCstation-1 enhanced with 10 serial ports.

2. A 386SX laptop as terminal.

3. TRC LABMATE wheeled locomotion base, which can be controlled by sending commands via a RS-232 serial port.

4. 24 ultrasonic sensors and 8 infrared sensors are placed in a circular pattern and can be controlled via a RS-232 serial port through the TRC Proximity Subsystem. The maximum detection range of the ultrasonic sensors is 10 m. Each infrared sensor returns a binary number which indicates whether it detects an object within a range of 76 cm.

5. Two Panasonic GP-KR202 CCD color cameras with 6mm and 3.6mm lenses.

6. Two Sun VideoPix image grabbers.

7. A pan/tilt carousel for mounting the cameras.

Walsh [126] developed a vision-based navigation scheme using symbolic landmark maps. Courtney [17] applied classification techniques for localization of a mobile robot. Dulimarta [33] used a client-server control architecture for robot navigation.

Figure 1.3: A picture of Rome outside our PRIP lab. A single CCD camera is mounted on top of the robot with a looking down tilt angle.

The above setup of Rome served as the testbed for all experiments presented in this thesis. A picture of Rome outside our PRIP lab is shown in Fig. 1.3. Currently Rome is under upgrade to its more powerful version. The major upgrade includes:

1. The host computer will be replaced by PentiumPro 200 MHz computer.
2. Matrox framegrabber.
3. More flexible pan/tilt units from Directed Perception Inc.
4. 3.6mm autoiris lenses.

1.7 Thesis Overview

In this thesis research, the author focuses on developing machine learning techniques and applying them to vision-based navigation by mobile robots. In Chapter 2 the application of SHOSLIF [133] to vision-based navigation is described. The vision-based navigation problem is treated as a content-based retrieval problem where each newly grabbed image is used to retrieve a recursive partition tree (RPT) which is trained by sample drives of the robot using a joystick control. The retrieved output control signal associated with the best-matched image is used as the next steering signal.

In Chapter 3, the above learning scheme is done incrementally for ease of training Rome to obtain satisfactory performance. The complexity of incremental learning is low and can be done on-line in real-time. Several experiments were conducted to verify the performance of the proposed algorithms.

System	Developer	Brief description	Real-time performance	On-line incremental learning	Use more than 1 frame of information
ALVINN	Pomerleau (CMU)	Feedforward multilayer perceptron (MLP) with back propagation learning.	X	X	
ELVIS	Hancock & Thorpe (CMU)	Eigensubspace method. The concatenation of an input image and its associated output signal is treated as a single long 1D vector.	X		
ROBIN	Rosenblum & Davis (Maryland)	Radial basis function neural network (RBFNN).	X		
SHOSLIF-N	Weng & Chen (MSU)	Content-based retrieval with a recursive partition tree (RPT). An input image is represented as a long 1D vector.	X	X	
State-based SHOSLIF-N	Weng & Chen (MSU)	SHOSLIF-N enhanced with state information and simple visual attention.	X	X	X

Table 1.1: A comparison of the thesis work with start-of-the-art vision-based navigation algorithms.

To deal with more scene types and improve the system's performance, state information and a simple visual attention mechanism are incorporated into SHOSLIF as state-based SHOSLIF. The state-based SHOSLIF is described in Chapter 4.

A comparison of the proposed learning mechanism for vision-based navigation with some representative state-of-the-art mobile robot vision systems which directly map an input image into output control signal is shown in Table 1.1. Although so far the algorithms described in this dissertation have been widely tested only in indoor navigation, they could possibly be applied to outdoor navigation.

Finally, Chapter 5 gives a summary of work done in this dissertation and future research directions are discussed.

Chapter 2

SHOSLIF for Autonomous Navigation

In this chapter, the earliest version of SHOSLIF-N—batch-mode SHOSLIF-N—for vision-based navigation is described.

2.1 Introduction

A central part of an autonomous vision-guided navigation system is the navigator. It accepts an input image $\mathbf{X}$ as well as an intention signal $\mathbf{P}$ which indicates which path to take. The navigator outputs the control signal vector $\mathbf{C}$ which controls the vehicle. Fig. 2.1 gives a schematic illustration of such a view. Thus, the navigator can be denoted by a function f that maps every element in the space of $(\mathbf{X}, \mathbf{P})$ to an element in the space of $\mathbf{C}$. A challenging task of autonomous navigation is to construct an automatic system which approximates (i.e., learns) the complicated

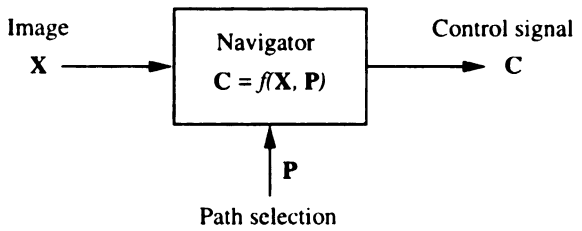


Figure 2.1: Navigator as a mapping from the image and path selection space to the control signal space.

function $f(\mathbf{X}, \mathbf{P})$ and computes quickly in real time given $\mathbf{X}$ and $\mathbf{P}$.

2.1.1 Existing systems

The function f is typically extremely complex, because of the high-dimension of the image space of $\mathbf{X}$. One common way to address this problem is to design some rules based on a particular type of image collections and then a system is developed by implementing these rules. For example, some outdoor road followers use human designed rules to follow road edges (e.g., Wallace *et al* [125], Dickmanns and Zapp [25], Thorpe *et al* [118], Hebert [48]). Indoor corridor followers mainly rely on floor or ceiling edges (e.g., Meng and Kak [81], Lebesgue and Argarwal [74]). In the past, autonomous navigation systems that used only intensity images rely very much on a type of predefined feature, such as road edge, lane mark, floor edge, etc.

The amount of road condition variation is a crucial factor in determining the applicability of a system. To handle a variety of environment conditions, several experimental navigation systems have employed certain adaptation mechanisms to various degrees. For navigation on paved driveways, when lane marks are well illuminated and easily traceable, a simple technique, such as adaptive color thresholding [72], can be used. In situations with unstructured roads with unmarked lanes and sud-

den changes in illumination (e.g, the sun is occluded by clouds, or shadows cast from nearby buildings), the design of a reliable autonomous navigation system becomes a great challenge. Investigation around the Navlab developed different adaptation mechanisms for different outdoor driving environments. SCARF [20] [120] was designed to handle various roads with adaptive color classification. YARF [64] [120] deals with different structured roads by explicitly modeling different available constraints and features, since a single technique for road detection may fail on different structured roads.

Many existing systems predefine the type of features that the system will use. For greater adaptability, some systems do not specify which specific features to use. With ALVINN [94] [120], a feedforward artificial neural network (ANN), trained by a back-propagation learning algorithm, maps input images to output steering signals for autonomous navigation. ALVINN does not explicitly impose the type of image geometric features upon which the system is based. ALVINN has been successfully tested in a variety of road situations. An alternative neural network approach is ROBIN [105], [106], developed at Univ. of Maryland and later extended at Lockheed Martin Corporation. ROBIN used a radial basis function (RBF) neural network to map a low-resolution input image into output steering signal. Because of its build-in regularization mechanism, ROBIN achieved smoother behavior than ALVINN did, as reported in [106].

2.1.2 Motivations

The work presented here is a navigational part of the Self-organizing Hierarchical Optimal Subspace Learning and Inference Framework (SHOSLIF) [133]. The navigation system SHOSLIF-N described here was motivated by the following considerations.

- Automatic feature derivation which minimizes imposing restrictions on the type of the scene the system will handle.

  The framework automatically derives the most expressive features (MEF) and the most discriminating features (MDF), which maximize representation and classification powers, respectively, from information available in the input data. Therefore, the method is potentially applicable to a wide variety of conditions.

- Self-organizing mechanism.

  The approach uses a *self-organizing* mechanism for systemwise learning, which is responsible for mapping input images to vehicle control signals.

- Low computational complexity.

  To address the complexity issue in learning a large number of scenes, a recursive space-partition tree is introduced, which has a logarithmic retrieval complexity. Due to this low complexity, the experimental system Rome is capable of real-time path following while computation is performed by an on-board SUN SPARC-1 without using any special image processing hardware other than a frame grabber.

Although feedforward networks and radial basis function networks can also deal with high dimensional mapping, our experimental result indicated that SHOSLIF-N performs better for the tasks tested. SHOSLIF-N does not need to start with a random initial guess which is difficult to lead to the correct solution [95]. It does not require human interactive selection of population centers as with radial basis networks [106].

This chapter is organized as follows. Section 2.2 introduces basic concepts and related results in learning with a nearest neighbor (NN) approximator. Section 2.3 discusses statistical tools for automatically deriving linear features from learning samples. Section 2.4 explains the self-organizing method which automatically constructs a tree. Complexity of SHOSLIF-N is analyzed in Section 2.5. Some experimental results are reported in Section 2.6, and a brief comparison with current neural network approaches is presented in Section 2.7. Finally Section 2.8 provides some concluding remarks.

2.2 Learning as Efficient Function Approximation

2.2.1 Image space

A digital image with r pixel rows and c pixel columns can be denoted by a vector in (rc)-dimensional space. For example, the set of image pixels $\{I(i,j) \mid 0 \leq i < r, 0 \leq j < c\}$ can be written as a vector $\mathbf{X} = (x_1, x_2, \cdots, x_d)^t$ where $x_{ri+j+1} = I(i,j)$ and $d = rc$. The actual mapping from the 2-D position of every pixel to a component in

the d-dimensional vector $\mathbf{X}$ is not essential but is fixed once it is selected. Since the pixels of all the practical images can only take values in a finite range, we define $\mathbf{X}$ as a point in a *domain* S of a finite size, where

$$S = \{\mathbf{X} \mid \mathbf{X} = (x_1, x_2, \cdots, x_d)^t, |x_i| \leq M/2, i = 1, 2, \cdots d\} \tag{2.1}$$

which can be called a d-dimensional hypercube. The second order statistics between pixels is represented by the corresponding covariance matrix Σ_x of the random vector $\mathbf{X}$, as we will see later. For example, if two pixels of the assemble at locations (i, j) and (i', j') are highly correlated in the original image, their corresponding cross-covariance stored in the corresponding element at $(ir + j, i'r + j')$ of the matrix Σ_x typically has a large absolute value. Therefore, treating a two-dimensional image as an one-dimensional vector $\mathbf{X}$ is for notational convenience and the representation itself does not necessarily lose any two-dimensional information.

2.2.2 Functions in the image space

The navigator shown in Fig 2.1 is a function f defined in a huge space of $(\mathbf{X}, \mathbf{P})$. For notational simplicity, we drop the input $\mathbf{P}$ in notation and simply denote $f(\mathbf{X}, \mathbf{P})$ as $f(\mathbf{X})$, regarding $\mathbf{P}$ as a part of $\mathbf{X}$.

In fact, many sensor-based understanding and control problems can be modeled as a function $\mathbf{C} = f(\mathbf{X})$, where $\mathbf{X}$ is the input vector and the output vector $\mathbf{C}$ can be real-valued. In a control problem, $\mathbf{C}$ is a real-valued vector. In an autonomous navigation problem, the vector can be, e.g., $\mathbf{C} = (c_1, c_2, c_3)$ where c_1 is the heading

direction, c_2 the speed, and c_3 the next step size. In classification problems, $\mathbf{C}$ can be a scalar $y \in \{1, 2, 3, \cdots, k\}$ indicating a class label of the input $\mathbf{X}$. However, in practice, such a classification output is often not sufficient without a confidence measure. More generally, the nominal output for n classes can be n dimensional, with ith component indicating the confidence (or some probability measure) for the input $\mathbf{X}$ to arise from the i-th class. Therefore, the categorical output can be considered as a special case of real-valued output.

From the discussion, we can see that many problems in sensor-based classification and control can be represented by a partial function f that maps elements in a domain S to an element in a codomain C:

$$f : S \mapsto C \tag{2.2}$$

where S is in defined in (2.1) and C is an n-dimensional space $C = R^n$. We regard f as a partial function because it is not necessarily defined on the entire set of S. For example, S may denote all the possible images but the images we see in our life is only a subset of S.

2.2.3 Learning as an approximation process

Our objective of constructing a navigator is equivalent to approximating function $f : S \mapsto C$ by another function $\hat{f} : S \mapsto C$. The error of approximation can be indicated by certain measure of the error $\hat{f} - f$. One such measure is the mean

square error

$$E\|\hat{f} - f\|^2 = \int_{\mathbf{X}\in S} \|\hat{f}(\mathbf{X}) - f(\mathbf{X})\|^2 dF(\mathbf{X})$$

where $F(\mathbf{X})$ is the probability distribution function of $\mathbf{X}$ in S. In other words, $\hat{f}$ can defer a lot from f in parts where $\mathbf{X}$ never occurs, without affecting the error measure. Another measure is the pointwise error $\|\hat{f}(\mathbf{X}) - f(\mathbf{X})\|$ for any point $\mathbf{X}$ in S', where $S' \subset S$ is a subset of S that is of interest to a certain problem. In navigation, S' may consist of all the inputs that may arise.

Of course, the function f is typically high-dimensional and highly complex. For navigation, it maps from a high-dimensional input vector $\mathbf{X}$, which represents a meaningful image and the current intention, to the correct control parameter vector.

A powerful method of constructing $\hat{f}$ is using learning. Specifically, a series of cases is acquired as the learning data set:

$$L = \{(\mathbf{X}_i, f(\mathbf{X}_i)) \mid i = 1, 2, \cdots, n\}. \tag{2.3}$$

Then, construct $\hat{f}$ based on L. For notational convenience, the sample points in L is denoted by $X(L)$:

$$X(L) = \{\mathbf{X}_i \mid i = 1, 2, \cdots, n\}.$$

$X(L)$ should be drawn from the real situation so that the underlying distribution of $X(L)$ is as close to the real distribution as possible.

2.2.4 Case-based learning for function approximation

Superficially, the function approximation problem we are dealing with here looks very much like a typical regression problem in statistics or pattern recognition, i.e., predicting y from a given feature vector $\mathbf{X}$. They are in fact very different due to the underlying assumption. In statistics, features are preselected so that a simple relationship between y and $\mathbf{X}$ exists. Therefore, the linear regression tree and k-d tree (also called tree classifier) are popular in statistics [5], [45] and pattern recognition [16], [38]. However, those methods do not work well here because of the very complex nature of the function f: it is highly nonlinear, high-dimensional in domain, and the number of samples can be even smaller than the dimension of the domain. For example, such a function cannot be reasonably approximated by any parameter set, let alone the linear regression parameter vector used in linear regression. The k-d tree does not work either because no threshold on any component of $\mathbf{X}$ can give a meaningful intermediate classification. The challenging problem of learning a highly nonlinear and high-dimensional function from real-world cases is fundamental to intelligence but has not received much attention in the past. In the following, we introduce some concepts for this task and establish related results.

A learning set L as in (2.3) consists of a sequence of cases. We need to investigate its relation with the space S.

Definition 1 *A partition $P = \{P_1, P_2, \cdots, P_n\}$ of S (i.e., $P_i \cap P_j = \emptyset$ when $i \neq j$ and $\bigcup_{i=1}^{n} P_i = S$) is consistent with a learning set L if $\mathbf{X}_i \in P_i$, $i = 1, 2, \cdots, n$.*

A partition P consistent with L can be considered as a result of partitioning S based

on the learning set L. Given an L, infinitely many consistent partitions are possible. The Dirichlet tessellation, also well known as the Voronoi diagram [54], is a special partition consistent with L in which each P_i consists of those points of S that are closer to $\mathbf{X}_i$ than to any other $\mathbf{X}_j$ in $X(L)$ with $i \neq j$.

Definition 2 *Given a learning set L, a nearest-neighbor (NN) approximator $\hat{f}$ of f associated with L is defined as follows. For any $\mathbf{X}$ in partition S,*

$$\hat{f}(\mathbf{X}) = f(\mathbf{X}_i)$$

where $\mathbf{X}_i$ is the nearest neighbor of $\mathbf{X}$ in $X(L)$. That is, $\mathbf{X}_i \in X(L)$ and $\|\mathbf{X} - \mathbf{X}_i\| < \|\mathbf{X} - \mathbf{X}_j\|$ for any $\mathbf{X}_j \in X(L)$ with $i \neq j$.

An NN approximator $\hat{f}$ of f is a piecewise constant function, constant in every P_i of the Dirichlet tessellation $P = \{P_1, P_2, \cdots, P_n\}$ which is consistent with L. The NN approximator is not unique, given an L, because the value at the boundary of the Dirichlet tessellation is arbitrary. In practice, its value along the boundary can take the value of either side of the boundary, whichever is convenient.

Definition 3 *Given a learning set L, a k-nearest-neighbor distance-based (KNDB) approximator $\hat{f}$ of f associated with L is defined as follows. For any $\mathbf{X} \in P$,*

$$\hat{f}(\mathbf{X}) = \frac{1}{\sum_{i=1}^{k} w_i} \sum_{i=1}^{k} w_i f(\mathbf{X}_{n_i})$$

where $\mathbf{X}_{n_i} \in X(L)$ is the i-th nearest neighbor of $\mathbf{X}$ and $w_i = w(\mathbf{X}, \mathbf{X}_{n_1}, \cdots, \mathbf{X}_{n_k})$ is the value of a scalar weighting function.

The KNDB approximator uses more neighbors than the NN version and thus is more immune to noise in the learning set L. When the top $s \leq k$ nearest neighbors have the same distance to $\mathbf{X}$, the value of $\hat{f}(\mathbf{X})$ takes roughly the average of these top s neighbors. As an example, the weighting function $w(\mathbf{X})$ can take the form

$$w_i(\mathbf{X}, \mathbf{X}_{n_1}, \cdots, \mathbf{X}_{n_k}) = \alpha^{-\|\mathbf{X}-\mathbf{X}_{n_i}\|/(\epsilon+\|\mathbf{X}-\mathbf{X}_{n_1}\|)}$$

where ϵ is a small positive number to avoid the denominator to become zero. The value of α determines how fast the weight will decrease for other runners up. A point $\mathbf{X}_{n_i}$ at twice the distance compared to that of the nearest neighbor X_{n_1} will have its weight decreased by a factor of $1/\alpha$ from that of the nearest neighbor. For example, $\alpha = 10$ is a reasonable choice for α. When α approaches infinity, the KNDB degenerates into the NN approximator.

Other interpolation schemes that may be used here include generalized multi-quadratics [113], [133] or radial basis functions [97], [98], [47].

2.2.5 Hierarchical partition and the RPT

An immediate issue that arises here is that a large number of samples is required in the learning phase to approximate a high-dimensional function f. In the performing phase, given an input $\mathbf{X}$, the NN approximator needs to find the nearest neighbor from a large number n of recorded items. A linear search in this database will require $O(n)$ time, which is not acceptable for real-time navigation with a large n, e.g., $n = 1,000,000,000$. The following hierarchical space partition scheme leads to a

$O(\log(n))$ retrieval complexity.

Definition 4 *A hierarchical partition P of S is a set of partitions $P = \{P_0, P_1, \cdots, P_m\}$, where every P_i, $i = 0, 1, \cdots, m$ is a partition of S. P_{i+1} is a finer partition of P_i. Suppose $P_i = \{P_{i,j+1}, P_{i,j+2}, \cdots, P_{i,j+l}\}$. For each $P_{i,j} \in P_i$, P_{i+1} contains either $P_{i,j}$ itself or $P_{i+1,k+1}, P_{i+1,k+2}, \cdots, P_{i+1,k+b}$ so that*

$$P_{i,j} = \bigcup_{i=1}^{b} P_{i+1,k+i}$$

A recursive partition tree (RPT) is a tree which represents a hierarchical partition P. $P_0 = S$ at level 0 is the root, representing the entire space S. A node $P_{i,j}$ at level i has b children if $P_{i,j}$ is further partitioned at level $i+1$ by b cells $P_{i+1,k+1}, P_{i+1,k+2}, \cdots, P_{i+1,k+b}$. A node $P_{i,j}$ is a leaf node if it is not further partitioned by more cells in P_{i+1}. If $P_{i,j}$ appears at several levels, $P_{i,j}$ is represented by a leaf at the lowest level at which it appears.

A hierarchical Dirichlet tessellation (or hierarchical Voronoi diagram) is a hierarchical partition that satisfies an additional condition: Every cell is further partitioned at a deeper level, if it does, only by a Dirichlet tessellation. As shown in Fig. 2.2, although further partition of each cell is a Dirichlet partition, the entire partion P_m which corresponds to all the leaf nodes in the RPT tree, is not necessarily a Dirichlet partition.

The entire space S is typically huge. The hierarchical partition covers the space using cells of different sizes. In areas where samples are very sparse, only large cells

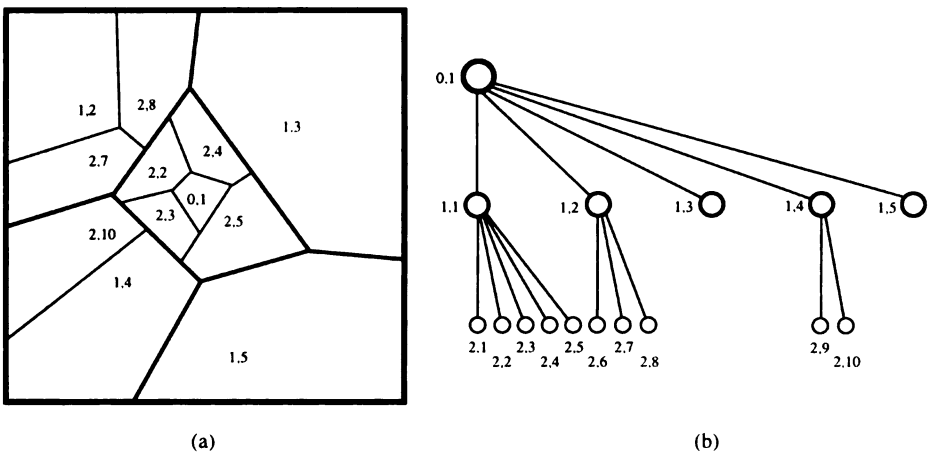


Figure 2.2: A 2-D illustration of a hierarchical Dirichlet partition and the corresponding recursive partition tree (RPT). (a) The partition, where the label indicates the center of a cell. The label of the child to which its parent's center belongs is not shown due to the lack of space. (b) The recursive partition tree.

are created to cover them. In areas where samples are very dense, many small cells are created to cover them. As we will see, the size of the cells is directly related to the accuracy of approximation. In other words, regions in which f is not defined or of low probability are approximated very roughly by a few coarse cells, but regions of high density are approximated in detail by many small cells.

2.2.6 Approximation using the RPT

We defer our discussion about the construction of the RPT to section 2.4. In this section, we study how to use the RPT for function approximation.

The use of hierarchical partition structure drastically reduces the complexity to find the cell in P_m to which a given $\mathbf{X}$ belong.

```
RPT Search Algorithm: Given X, find a leaf node c so that X ∈ c.
node = root;
for each child c of node {
    if (X ∈ c) {
        if (c is not a leaf)
            node = c;
        else return(c);
    }
}
```

Suppose L is a complete tree in which every internal node has b children. A tree of level $l+1$ has $n = b^l$ leaf nodes. At each level, the algorithm needs to visit b children. The number of nodes visited by the above search algorithm is $bl = b\log_b(n)$. With a complete tree, the search complexity is logarithmic $O(log_b(n))$ in the number of leaf nodes. We will analyze the complexity in detail in Section 2.5.

Although a hierarchical partition allows a very fast retrieval algorithm, the final partition P_m in a hierarchical partition P is not a Dirichlet partition in general, as indicated in the example shown in Fig. 2.2. This is because the boundary of a coarse cell at level l is defined according to the cells at level l. Such a coarse-level boundary is present at the finest partition P_m and is typically inconsistent with that of the Dirichlet tessellation defined on all the given samples. Therefore, the NN approximator cannot be directly defined on a hierarchical partition.

Our way of dealing with this problem is to explore k competitive paths down the RPT and use the technique of the KNDB approximator at the k leaf nodes that are reached.

Definition 5 *Given a RPT, corresponding to a hierarchical partition* $P = \{P_1, P_2, \cdots, P_m\}$ *and an integer constant* $k > 0$, *the* k *competitive leaves of an input*

X *are defined recursively as follows:*

1. *At level 1, the k children of the root whose centers are the k nearest neighbors of* **X**, *among all nodes at level 1, are competitors at level 1.*

2. *At level l, let C contains all the children of the k competitors at level* $l-1$ *plus those competitors at level* $l-1$ *which are already leaves at a level* $\leq l-1$. *The k elements in C whose centers are the k nearest neighbors of* **X**, *among all other elements in C, are the k competitors at level l.*

The k competitive leaves *of* **X** *are the k competitors at the last level m.*

Note that the k competitive leaves of a RPT is not necessarily the k nearest neighbors of **X**, although they could be. In particular, the nearest neighbor of **X** is included in the k competitive leaves only if every node along the path from the root to the nearest-neighbor leaf is included in the k competitors at the corresponding level. The larger k is, the more likely that occurs.

Definition 6 *A k-competitor distance-based (KCDB) approximator* $\hat{f}$ *of* f *associated with the given learning set L and the associated RPT is the same at the KNDB, except that the k nearest-neighbors is replaced by the k competitive leaves obtained from the RPT.*

As long as the nearest-neighbor is included in the k competitors, the value of interpolation from k competitive leaves are typically very close to that of the KNDB approximator, because it is the nearest neighbor of **X** that plays a major role in the interpolation. If the nearest neighbor is missing from the k competitive leaves, the

result of interpolation should still be reasonably good if the k competitive leaves are not all too far from $\mathbf{X}$. The resilence of the KCDB approximator is due to the use of interpolation instead of using a single nearest neighbor from the k competitive leaves.

The next important issue is how to derive features which determine the shape and size of the cells in partition so that each cell will cover a large region in which the function f is nearly constant or does not change much. This is very important since it does not only reduce the number of leaves in the RPT for efficiency, but also allows the RPT to generalize effectively from a relatively small number of learning samples.

2.3 Automatic Feature Derivation

If we cannot restrict the type of scenes that a vehicle can come across, it is then improper to predefine features that the system can use. For example, if we define long straight edges to use for indoor navigation, the system will fail in curved hall ways or places where no long straight edges are available. If we define smooth curved edges to use for outdoor navigation, the system will fail when the road condition is so poor that no clearly definable edges are present. Even when road edges are clear, an edge tracker may fail in the presence of passers-by who occlude road edges. Therefore, we must let system to automatically derive features by itself, depending on the type of environment in which the system learns and operates.

2.3.1 Meaningful images in the image space

We know that the dimension of an image space is typically very large. For a moderate 128×128-pixel image, the dimension is $d = rc = 128 \times 128 = 16,384$! Suppose that the absolute intensity value of all image pixel is bounded by a number M, and the space of all the *possible* images is given by S in (2.1). But all the images a robot can see in its operational environment distribute within a very small subset S' of this huge space S, although the region shape of S' can be very complex. In other words, the function $\hat{f}$ needs only to approximate f within a much smaller space S'. An objective of learning is to find a representation to represent the subspace S' well enough for approximation of f in it. The actual value of f in $S - S'$ is not of major concern.

The Karhunen-Loeve projection [24] is a very efficient way to represent a small subspace in a high-dimensional space. It reduces the dimension of representation from d in S to a much lower dimension for S' while still keeps most information in the data. However, the Karhunen-Loeve projection is not effective for discriminating different scenes, for which we will use the discriminant analysis to derive the best features. For a general discussion of dimension reduction using projection pursuit, the reader is referred to an excellent survey by Huber [52].

2.3.2 The most expressive features

As we discussed above, images of objects in a category can be regarded as samples represented by an d-dimensional random vector $\mathbf{X}$ in S, which can be represented by

a linear combination of d orthonormal basis vectors, $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_d$, so that

$$\mathbf{X} = \sum_{i=1}^{d} y_i \mathbf{v}_i = V\mathbf{Y}$$

where V is an orthogonal $d \times d$ matrix consisting of orthonormal column vectors $\mathbf{v}_i, i = 1, 2, \cdots, d$. Without loss of generality, we can assume that the mean the random vector $\mathbf{X}$ is a zero vector, since we can always redefine $\mathbf{X} - E\mathbf{X}$ as the feature vector.

To represent vectors in S', we can predict that it might be enough to use a relatively small number of expansion terms to characterize the object scene to some degree of precision. Suppose we use m basis vectors, each of which is a column vector of V, and the covariance matrix of $\mathbf{X}$ is $\Sigma_x = E[(\mathbf{X} - E\mathbf{X})(\mathbf{X} - E\mathbf{X})^t]$. The approximate representation of $\mathbf{X}$ is $\hat{\mathbf{X}}(m) = \sum_{i=1}^{m} y_i \mathbf{v}_i$. It has been proved [75] that the optimal $\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_m$ that minimize mean square error

$$\epsilon^2(m) = E\|\mathbf{X} - \hat{\mathbf{X}}(m)\|^2 = \|\mathbf{X} - \sum_{i=1}^{m} y_i \mathbf{v}_i\|^2$$

are the m unit eigenvectors of the covariance matrix Σ_x associated with the m largest eigenvalues. After the computation of $\mathbf{v}_i$'s, the m components, y_i, of the projected vector of $\mathbf{X}$ can be computed by

$$y_i = \mathbf{v}_i^t(\mathbf{X} - E\mathbf{X}) \quad i = 1, 2, \cdots, m \tag{2.4}$$

or $\mathbf{Y} = V^t\mathbf{X}$, where $V = [\mathbf{v}_1\ \mathbf{v}_2\ \ldots\ \mathbf{v}_m]$. This is known as the Karhunen-Loeve

projection [75]. Since the m features, $\mathbf{v}_i$'s in (2.4), give the minimum mean-square error, we can call them the *most expressive features* (MEF) in that they best describe the sample population in the sense of a linear transformation.

We can choose m so that the ratio of the mean square error over the total variance satisfies

$$\frac{\sum_{i=m+1}^{d} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \leq \tau \tag{2.5}$$

(e.g. $\tau = 5\%$).

In practice, we are given a set of n learning images, $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_n$, Σ_X is approximated by scatter matrix

$$S = \sum_{i=1}^{n} \frac{1}{n} (\mathbf{X}_i - \mathbf{m})(\mathbf{X}_i - \mathbf{m})^t = UU^t \tag{2.6}$$

where $U = [\mathbf{U}_1\ \mathbf{U}_2\ ...\ \mathbf{U}_n]$ with $\mathbf{U}_i = \mathbf{X}_i - \mathbf{m}$, $\mathbf{m} = (1/n) \sum_{i=1}^{n} \mathbf{X}_i$. If the number of learning images, n, is smaller than dimension d, instead of computing the eigenvalues and eigenvectors of the a very large $d \times d$ matrix UU^t directly, we can find the eigenvector and eigenvalues of a smaller $n \times n$ matrix U^tU, which in fact has the same non-zero eigenvalues as $S = UU^t$. If $\mathbf{w}_i$ is an eigenvector of U^tU associated with the eigenvalue λ_i, then $\mathbf{v}_i = U\mathbf{w}_i$ is the eigenvector of $S = UU^t$ associated with the same eigenvalue.

The Karhunen-Loeve projection reduces the image dimension from d to m, which is typically a lot smaller than d. Fig. 2.3 illustrates the meaning of the MEFs. The regions within which the sample points distribute has a much lower dimension than the

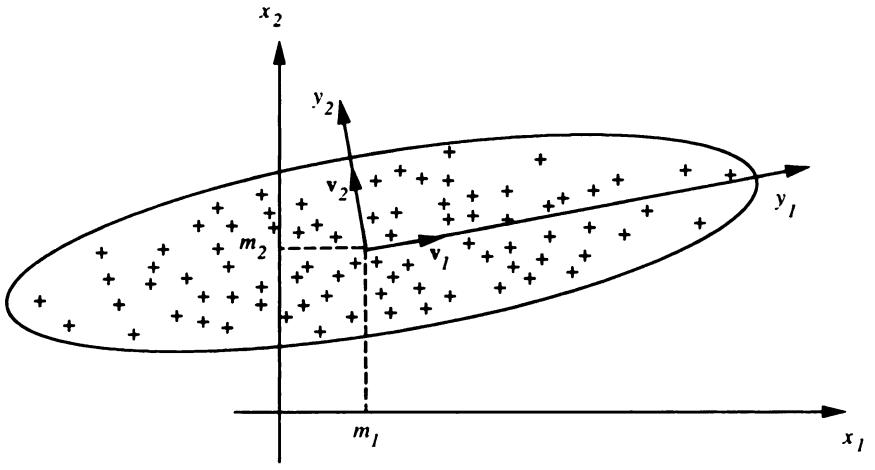


Figure 2.3: A 2-D illustration of the most expressive features (MEF). The MEFs are $\mathbf{v}_1$ and $\mathbf{v}_2$ and the MEF values are the projected values onto y_1 and y_2 axes.

original space. The first two eigenvectors $\mathbf{v}_1$ and $\mathbf{v}_2$ tell the most significant directions of sample variation. The MEFs are useful to reduce the dimension of representation. They are the best linear features to maximally preserve the Euclidean norm among the samples. For some works on using MEF for recognition-related problems, the reader is referred to [124] [86].

2.3.3 The most discriminating features

During navigation, the system needs to distinguish a variety of road types. Is it a cross intersection, a Y-shaped branching, a T-junction or a straight way? The road types will be used, coupled with the intention $\mathbf{P}$, to decide next navigation action. Furthermore, given the same type of scene, different current heading directions (assuming that the camera is fixed on the vehicle), result in different scene appearances. For example, the next heading direction should depend on the way in which the road extends as seen from the current heading direction, left, straight, or right. All these

require that the navigator to classify scenes according what it sees.

In the MEF space, a nearest neighbor type of criterion is not well suited for scene classification, since the Euclidean distance in MEF space does not necessarily indicate a perceptual distance. Due to the effects of many unrelated factors, such as lighting, the Euclidean distance between two views of the same scene at different times may be much larger than those between two different scenes. In other words, the pixel-to-pixel distance, whether in the original image space or MEF space, cannot well characterize why two scenes are considered different.

When the category labels of the sample data are known, the discriminant analysis explained below can be used to automatically derive features that best characterize different categories.

Suppose samples, $\mathbf{Y}$'s, are m-dimensional random vectors from c classes. The ith class has a probability p_i, a mean vector $\mathbf{m}_i$ and a scatter matrix Σ_i. The *within-class scatter matrix* is defined by

$$S_w = \sum_{i=1}^{c} p_i E\{(\mathbf{Y} - \mathbf{m}_i)(\mathbf{Y} - \mathbf{m}_i)^t \mid \mathbf{Y} \in \text{class i}\} = \sum_{i=1}^{c} p_i \Sigma_i \tag{2.7}$$

The *between-class scatter matrix* is $S_b = \sum_{i=1}^{c} p_i(\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^t$ where $\mathbf{m}$ is defined as $\mathbf{m} = E\mathbf{Y} = \sum_{i=1}^{c} p_i \mathbf{m}_i$. The *mixture scatter matrix* is defined by $S_m = E\{(\mathbf{Y}-\mathbf{m})(\mathbf{Y}-\mathbf{m})^t\} = S_w + S_b$. We want to maximize the between-class scatter with respect to the within-class scatter. In the discriminant analysis [142], an objective

function we wish to maximize is

$$\text{trace}\{S_w^{-1} S_b\} \tag{2.8}$$

In other words, we want to find features in which the scatter between classes is large but the scatter within every class is small. Suppose we use k-dimensional linear features $\mathbf{Z} = W^t\mathbf{Y}$ where W is an $m \times k$ rectangular matrix whose column vectors are linearly independent. The above mapping represents a linear projection from m-dimensional space to k dimensional space. The objective is to determine an $m \times k$ matrix W so that in the new space, the objective function in (2.8) $f_Z(m) = \text{trace}\{S_{Zw}^{-1} S_{Zb}\}$ is maximized. Alternatively, we can also consider another objective function. $\det\{S_{Zw}\}$ and $\det\{S_{Zb}\}$ measure the hyperellipsoidal scattering volume of S_{Zw} and S_{Zb}, respectively. We may maximize their ratio

$$\frac{\det\{S_{Zb}\}}{\det\{S_{Zw}\}} = \frac{\det\{W^t S_{Yb} W\}}{\det\{W^t S_{Yw} W\}} \tag{2.9}$$

It has been proved that both objective functions (2.8) and (2.9) lead to the same projection matrix W whose column vectors are the eigenvectors of $S_{Yw}^{-1} S_{Yb}$ associated with the first k largest eigenvalues. The column vectors of W are called *the most discriminating features* (MDF). They are the best linear features that maximize the ratio in (2.9). Since the rank of S_{Yb} is at most $c-1$, we know that only at most $c-1$ features are needed and others do not contribute to the maximum of the objective functions.

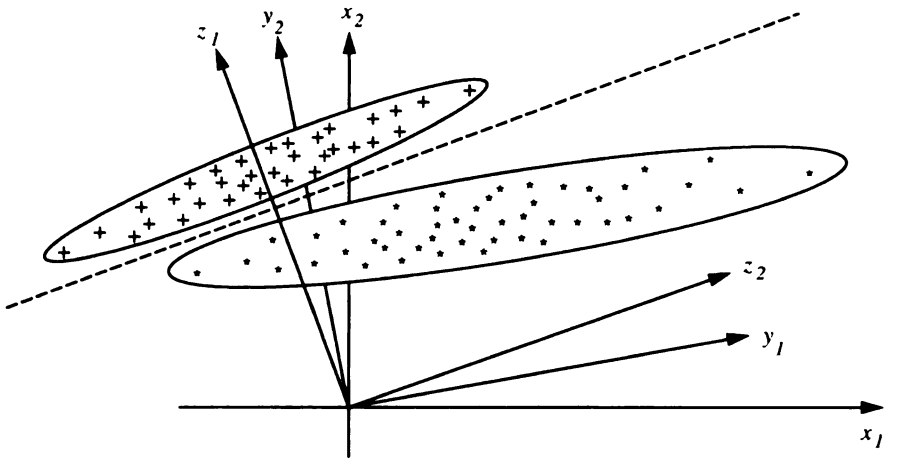


Figure 2.4: A 2-D illustration of the most discriminating features (MDF). The MDF is the projection along z_1. The MEF along y_1 cannot separate the two subclasses.

Fig. 2.4 illustrates the meaning of MDF. Without class labels, all the samples are used to compute MEFs. In the figure, the first MEF along y_1 cannot separate the two subclasses. The direction along the difference of the two means cannot either. Although the variation along the first MDF z_1 is not large, it catches the major feature that is crucial for classifying two subclasses.

In practice, we are given images from c categories. Let the number of samples belonging to ith class be n_i, the jth sample in ith class be $\mathbf{Y}_j^{(i)}$ and the mean vector of ith class be $\mathbf{m}^{(i)}$. The grand mean vector for all samples is replaced by the grand sample mean

$$\mathbf{m} = \frac{1}{n}\sum_{i=1}^{c} n_i \mathbf{m}^{(i)} \quad \text{where } n = \sum_{i=1}^{c} n_i$$

The mixture scatter matrix is replaced by the sample scatter matrix:

$$S_m = \sum_{i=1}^{c}\sum_{j=1}^{n_i} (\mathbf{Y}_j^{(i)} - \mathbf{m})(\mathbf{Y}_j^{(i)} - \mathbf{m})^t$$

Define the sample scatter matrix for the ith class as

$$S^{(i)} = \sum_{j=1}^{n_i} (\mathbf{Y}_j^{(i)} - \mathbf{m}^{(i)})(\mathbf{Y}_j^{(i)} - \mathbf{m}^{(i)})^t$$

The within-class scatter matrix is replaced by

$$S_w = \sum_{i=1}^{c} S^{(i)}$$

and the between-class scatter matrix is replaced by

$$S_b = \sum_{i=1}^{c} n_i (\mathbf{m}^{(i)} - \mathbf{m})(\mathbf{m}^{(i)} - \mathbf{m})^t$$

The relation $S_m = S_w + S_b$ still holds for the sample scatter matrices.

The stable solution of eigenvectors and associated eigenvalues of $S_{Y_w}^{-1} S_{Y_b}$ can be obtained via simultaneous diagonalization [38].

2.3.4 The DKL projection

The discriminant analysis procedure breaks down when the within-class scatter matrix S_w becomes degenerate, which is true when the dimension of the input image is larger than the number of learning samples.

In fact, the discriminant analysis can be performed in the space of the Karhunen-Loeve projection (i.e., MEF space), where the degeneracy typically does not occur. Thus, the new overall discriminant projection is decomposed into two projec-

tions, the Karhunen-Loeve projection followed by the discriminant projection. To do this, we first project the d-dimensional $\mathbf{X}$-space onto m-dimensional MEF space ($\mathbf{Y}$-space) using the Karhunen-Loeve projection. Then, we project $\mathbf{Y}$-space on to the k-dimensional MDF space ($\mathbf{Z}$-space). Mathematically, we define the new *DKL projection* (short for Discriminant Karhunen-Loeve projection) from the d-dimensional space of $\mathbf{X}$ to the k-dimensional space of $\mathbf{Z}$ as $\mathbf{Z} = W^t V^t \mathbf{X}$.

We must determine the dimension m of $\mathbf{Y}$ so that S_w is not degenerate. Given s samples from c classes, the maximum rank of S_w is $s - c$. Therefore, in order to make S_w nondegenerate, the input space of MDF projection m (i.e., the size of S_w) cannot be larger than $s - c$. That is, $m \leq s - c$. This means that, in the MEF projection, we need to discard $(s - c + 1)$-th up to $(s - 1)$th largest eigenvalues of Σ_X. As we can expect, these eigenvalues are typically extremely small, because the $(s - 1)$th eigenvalue is the smallest among all the nonzero eigenvalues. In practice, we would like to discard more because it is very unlikely that they are important for classification either. On the other hand, m cannot be smaller than the number of classes c. Therefore we have $k < c \leq m \leq s - c$. Typically, s is much larger than c and thus, m can be chosen as an integer value in $[c, s - c]$ which also satisfies the variation criterion in (2.5).

2.3.5 Separability

All the features we discussed here are linear features since only linear projections are considered for deriving features. These linear features possess the optimal properties

as we discussed above and they allow a fast computation. However, an important question is the separability. That is, whether linear features can separate regions of arbitrary shapes in the space S.

Geometrically, we have seen in Section 2.2.6 that the final partition in a hierarchical partition P is the finest partition P_m. Other intermediate partitions are used mainly to speed up the search. The use of linear features here implies that cells in the partition P_m are bounded by hyperplanes. Geometrically, we can see from Fig. 2.2 that piecewise linear hyperplance can approximate virtually any region to a desired accuracy.

Mathematically, Cover [18] showed that NN estimator is a general estimator: under mild continuity conditions, the unconditional risk of the NN estimate is bounded by twice of the Bayesian risk when the sample size approaches infinity. Therefore, using our coarse to fine hierarchical partition scheme where the tessellation is based on the finest cells, linear features are sufficient to solve virtually any smooth nonlinear problem.

Unlike feedforward neural networks trained with a type of iterative minimization method such as back-propagation, a successful class partition is guaranteed if a sufficient number of samples are used. The use of MDF makes the partition to be accomplished by fewer hyperplanes for better generalization.

2.4 Self-Organizing RPT Generation through Learning

2.4.1 Data organization

First, consider how to automatically organize visual data. We might first find clusters in the sample data and search for the best clusters to be assigned to each child. Then, cells are defined, each corresponds to a child. Recursively, a hierarchical clustering structure in the data may be explored so that a hierarchical tree can be constructed. However, algorithms for clustering analysis are very time consuming due to their iterative nature, and their complexity is not suited for learning a huge number of samples.

Our objective is to approach a learning complexity that is close to $O(n)$ for a learning set of size n. We cannot hope for any lower complexity since each item needs to be examined for learning.

With this goal, we can recursively partition samples into smaller and smaller sets, without worrying about whether each cell corresponds to a cluster or not. Because we are interested in finding the best matching sample for an input, cluster is not a major concern.

Our partition tree is different from the existing ones in that features are recursively derived based on the actual samples each node receives in the learning phase. In other words, nodes at the same level or different levels all use different sets of features, each is best for the task of each node.

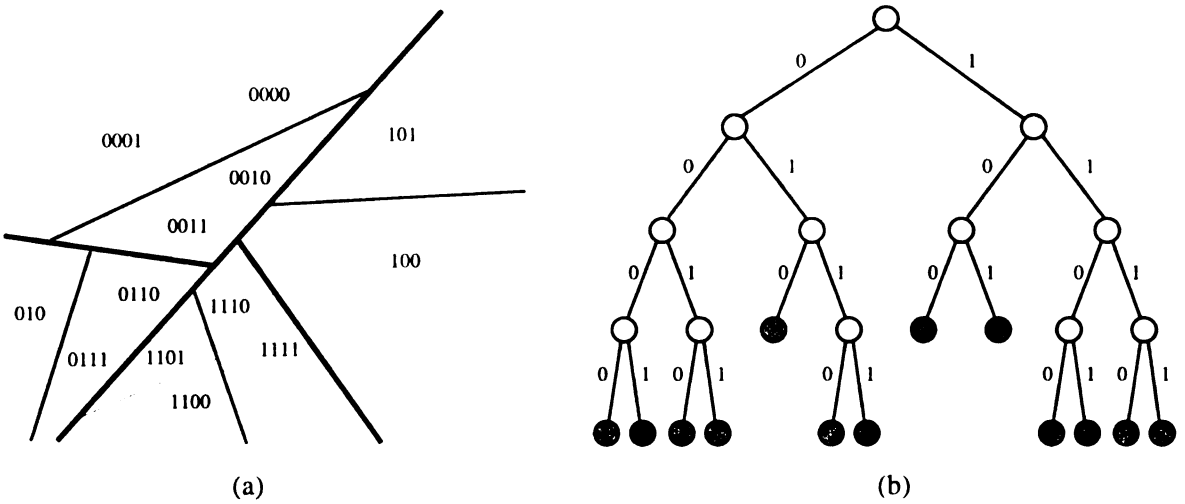


Figure 2.5: Geometrical illustration of the binary RPT. (a) The geometric cells. Thicker lines indicate boundary of coarser-level cells. The thinner lines indicate boundary of finer-level cells. (b) The corresponding RPT.

2.4.2 Binary recursive partition tree

For real-time navigation, the speed of tree search is a central issue. To project a d-dimensional input into a k-dimensional space, k inner products need to be computed, each for a d dimensional vector. Since d is large, ranging from 300 to 4800 with a reduced-resolution image in our experiment, we chose to adopt $k = 1$ for the advantage of minimum computation and simplicity in implementation. In other words, each node uses only a single feature.

Since only one feature is used at each internal node, we use the sample feature mean as the threshold to break the samples into two groups, one for the left child and one for the right. Thus, the result is a binary tree, as shown in Fig. 2.5.

In the context of organizing samples into a decision tree, various ways have been used to select the feature at each node to assign samples to child nodes. Mui and Fu [84] used an iterative method to classify nucleated blood cells. Sethi and Savarayudu

[107] maximized the amount of average mutual information gain at each partition step. Chou [16] employed the criterion of information divergence. In our case, we use Karhunen-Loeve projection or DKL projection, respectively, to automatically derive linear features that are best at each node. In applications where no sample label is available, the MEF feature is used. When class label is available, the MDF is used.

2.4.3 Constructing MEF RPT

Given a set of learning samples

$$L = \{(\mathbf{X}_i, f(\mathbf{X}_i)) \mid i = 1, 2, \cdots, n\}. \tag{2.10}$$

the objective is to automatically construct an MEF RPT. The root of the tree accepts the entire set L for learning. At each node, the samples that go to this node are used to compute the first MEF feature vector. All the samples that have a negative MEF value go to the left child and the remaining ones go to the right. Such a process is performed recursively from the root down to each node's children. The recursive subdivision is terminated at a node when it has only one sample.

The power method [39] can be used to compute the dominant eigenvector as the first MEF. Let n and d be the number of samples and the dimension of input sample vector, respectively, and $n \leq d$. The time complexity for computing U^tU is $O(n^2d)$, for the dominant eigenvector of U^tU is $O(n^2)$, that for projecting n input samples to the first MEF feature is $O(nd)$. The total complexity is then $O(n^2d)$. When the number of learning samples is very large, the computation for MEF features becomes

time consuming. Coarse level nodes correspond to large cells and thus they have more samples than finer level nodes. However, at a coarse level we do not need to use all the samples to compute the exact MEF, because the partition at a coarse level is in a rough sense anyway. Thus, we adopt a constant q as the maximum number of samples used to compute MEF at every level. Whenever the number of samples available at a node exceeds q, a subset of q samples are randomly selected to be used for MEF computation. Of course, we will not waste the other remaining samples. All the samples of a node will be assigned to the corresponding children where they will be used. At fine levels, the number of samples assigned to each node becomes small. As soon as the number of samples of a node is not larger than q, all the samples are used for its MEF computation. The following is a pseudo code of the algorithm.

```
MEF RPT Construction Algorithm: Given L, construct the MEF RPT.
call construct(root, L);
construct(node, L) {
    If (||L|| = 1) { node = L; return; }
    else if (||L|| > q}) L' = pick-q-samples(L);
    else L' = L;
    compute MEF1 from L';
    mean = average of MEF1-projections of all s in L';
    left-set = right-set = ∅;
    for each s in L {
        if (MEF1-projection(s) - mean < 0)
        left-set = left-set∪{s};
        else right-set = right-set∪{s};
    }
    construct(node→leftchild, left-set);
    construct(node→rightchild, right-set);
}
```

2.4.4 Constructing MDF RPT

The MDF RPT uses MDFs and thus class labels are required for each sample in the learning set. Two types of labels can be used, single-level label or hierarchical label. A hierarchical label is of the form $X_1.X_2.\cdots.X_t$, where each X_i is a class label for level i. The hierarchical label is intended to direct the RPT to group the samples in the way desired by the hierarchical label. For example, a coarse level label X_1 can have two classes, corridor and corner, so that the tree will try to group samples into corridor images and corner images before further subdividing samples into finer classes. A single-level label is a special case of the hierarchical label in that $t = 1$.

The construction algorithm of MDF RPT is similar to that of the MEF RPT. The major difference is the computation of MDF instead of MEF, and the use of label. A set of labeled learning samples is of the form

$$L = \{(\mathbf{X}_i, f(\mathbf{X}_i), b_i) \mid i = 1, 2, \cdots, n\}. \tag{2.11}$$

where b_i is the label of sample $\mathbf{X}_i$. If the labels are hierarchical, $b = b_1.b_2.\cdots.b_t$, where the label b_i is intended to be used at level i. For any node at a level number larger than t, the deepest level label, b_t, is used. The recursive subdivision of the samples is terminated at a node if the node has only samples of a single label b_t. A form of pseudo code of the algorithm is given bellow.

```
MDF RPT Construction Algorithm: Given a labeled L, construct the MDF RPT.
call construct(root, L, 0);
construct(node, L, level) {
    If (samples in ||L|| have the same deepest label b_t) { node = L; return; }
    else if (||L|| > q}) L' = pick-q-samples(L);
    else L' = L;
    compute MDF1 from L' using labels in L';
    mean = average of MDF1-projections of all s in L';
    left-set = right-set = ∅;
    for each s in L {
        if (MDF1-projection(s) - mean < 0)
        left-set = left-set∪{s};
        else right-set = right-set∪{s};
    }
    construct(node→leftchild, left-set, level+1);
    construct(node→rightchild, right-set, level+1);
}
```

Of course, instead of using mean only for assigning the left and right sets, the mean and variance of the samples in each class can be used to select a better threshold so that the boundary will not cut through a single-class cluster. Again, as we know, the boundary does not have to be a perfect classification due to the coarse-to-fine nature of the RPT partition. A fast algorithm is more important when the number of sample images is very large.

2.4.5 Learning phase and performing phase

The result of the learning phase is an RPT, as illustrated in Fig. 2.6. At each leaf node, each learned image $\mathbf{X}_i$ has its corresponding control signal vector $f(\mathbf{X}_i)$.

In the performing phase, the KCDB approximator will use more than one leaf node if $k > 1$. However, which k leaves will be used depends on query input $\mathbf{X}$.

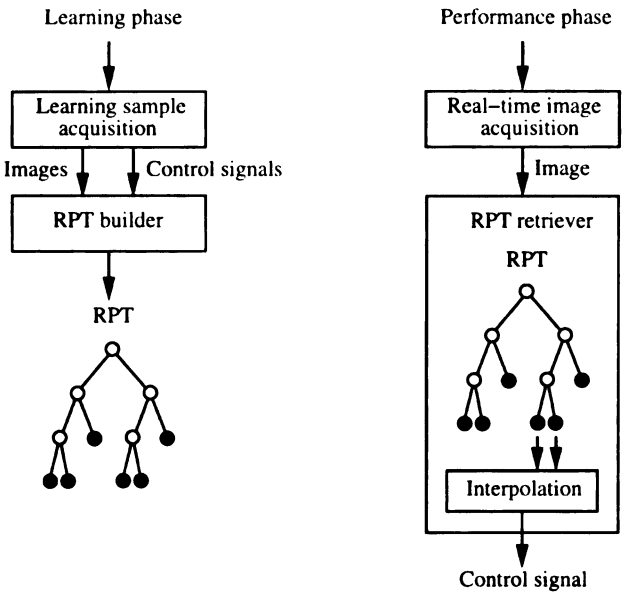


Figure 2.6: The learning and performing phases.

Therefore, the connection from k leaves to the interpolation part is neither a part of tree nor stored. It is generated in run-time. The RPT can be regarded as a visual information database. Since the entire input $\mathbf{X}$ is used for query, the retrieval from this database in content-based.

2.4.6 Differences between SHOSLIF-N and two neural network approaches

A schematic difference between our SHOSLIF-N and current neural network approaches is shown in Fig. 2.7. Current neural network approaches use an explicit function f to directly map a visual input I into output signal O. Here f is represented by a linear or nonlinear function of linear or nonlinear combination of first and second layer neurons. The adjustable parameters are weights of the first and

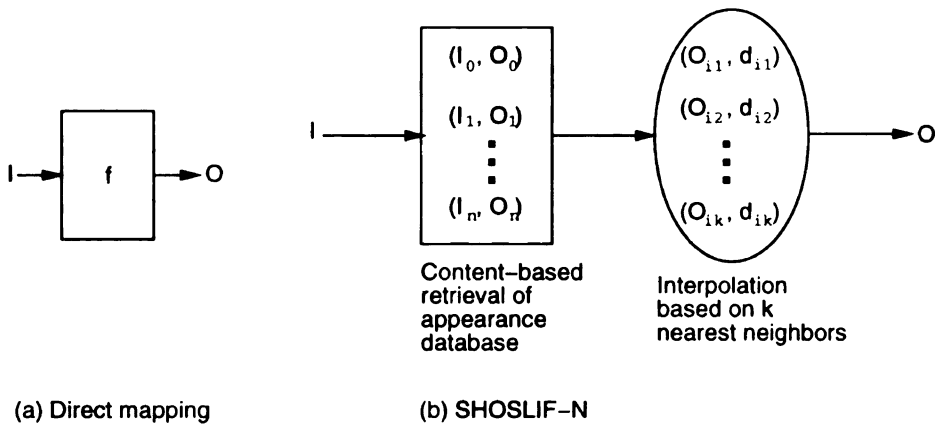


Figure 2.7: Schematic difference between current neural network approaches for vision-based navigation and SHOSLIF-N. (a) Direct input-to-output mapping using neural networks. (b) SHOSLIF-N.

the second layers in a feedforward network, and widths and centers of first layer and weights of second layer for a RBF network. In SHOSLIF-N, a novel input I is used to retrieve its k nearest neighbors in the learned samples. Then the associated outputs of k samples are interpolated to obtain the actual output. Notice here there is a task decomposition: the visual input is used in content-based retrieval, while associated outputs are weighted according to their distances to current input in the interpolation process to get the desired output. Direct input-to-output mapping in Fig. 2.7(a) can be regarded as a global approximation trying to minimize the global sum of square error between target outputs and simulated network outputs. SHOSLIF-N can be regarded as a piecewise approximation of input-to-output mapping based on learning samples, as shown in Fig 2.8. Better and better approximation can be achieved by adding more and more learning samples. On the other hand, more space is required for storing these samples.

We notice that simultaneous to our work [11], Hancock and Thorpe [44] applied eigen-subspace method to outdoor navigation in their ELVIS system. ELVIS uses a

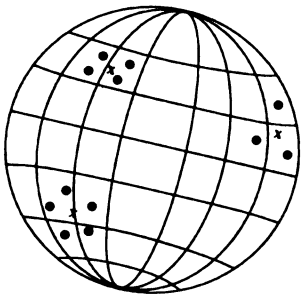


Figure 2.8: Piecewise approximation of SHOSLIF-N: mapping of visual input to output control signal is based on a local neighborhood of the input image. The sphere is used to represent a high-dimensional input space. 'x' denotes a sample visual input. The output associated with input 'x' is a value based on interpolation of outputs associated with its nearest neighbors which are marked as solid dot patterns.

global parametric regression method. Each training sample pair with input image I and output O, a Gaussian distribution peaked at desired heading, is treated as a monolithic vector. The coordinates of I in the eigenspace of (I, O) are used as weights to generate output from the output part of eigenvectors. Our method is a nonparametric method which uses local interpolation and thus can better approximate a function than the global parametric method.

2.5 Complexity

A detailed complexity analysis for the MEF RPT is given. The analysis for MDF RPT is similar, except that at each nonleaf node MEF and MDF projections (i.e., DKL) are both employed.

2.5.1 Complexity in the learning phase

The time complexity for computing the first MEF feature from n samples of dimension d is $O(nd^2)$, when $n > d$. Given a set of N learning images with a dimension d, typically a low resolution (30×40) for images is sufficient and we use a large number of learning samples. Therefore, $N > d$. The MEF-RPT will have an average leaf-node depth $O(\log N)$ [65] and thus the time complexity for the learning algorithm is

$$\sum_{l=1}^{\log N} \sum_{i=1}^{2^l} O\left(N_{l,i-1} d^2\right)$$

where $N_{l,i}$ is the number of learning images coming to the i-th node at level l. We know that $\sum_{i=1}^{2^l} N_{l,i-1} \leq N$, so the complexity is $O(Nd^2 \log N)$.

The time complexity is $O(N \log N)$, where N is the number of learning samples.

This low learning complexity is due to the fact that the method avoids time-consuming iterative analysis of input samples, which makes it possible to extend the method to future incremental learning, where the time required for learning a single sample at a time is $O(\log N)$.

Next, we consider the space complexity. The number of nodes in the binary RPT tree is no more than $2N - 1$. The number of leaves in the RPT is no more than N. Each child node need only to store the dominant eigenvector.

The space complexity of learning phase is $O(dN) = O(N)$.

2.5.2 Complexity in the performing phase

First consider the time complexity. The average leaf-node depth of a N-leaf binary tree is $O(\log N)$. A query of a new input image needs one projection at each inner node along the searched path, so the complexity of retrieving the best match for a new image is $O(d \log N) = O(\log N)$. This low time complexity allows the system to response fast even when N is large.

For each internal node of the learning tree, we need to keep only the dominant eigenvector. The space complexity of the performing phase is then $O(Nd) = O(N)$.

2.6 Experimental Results

The experiments include performance evaluation and real runs with a mobile robot at MSU, Rome, which is built on a Labmate platform from TRC.

2.6.1 Image acquisition

The test site for our navigation experiments is inside of the Engineering Building at MSU. The experimental area consists of several corridors with various turns. Rome was controlled manually to take pictures at different positions for learning. At each position, a set of five images with different heading directions were obtained: two left headings (5 and 10 degrees), two right headings (5 and 10 degrees) and one straight-ahead direction. The corresponding corrected heading directions were also recorded. For straight corridors, each set of pictures were taken at roughly every 2 to 3 meters. At turns, learning images were taken during sample drives controlled manually us-

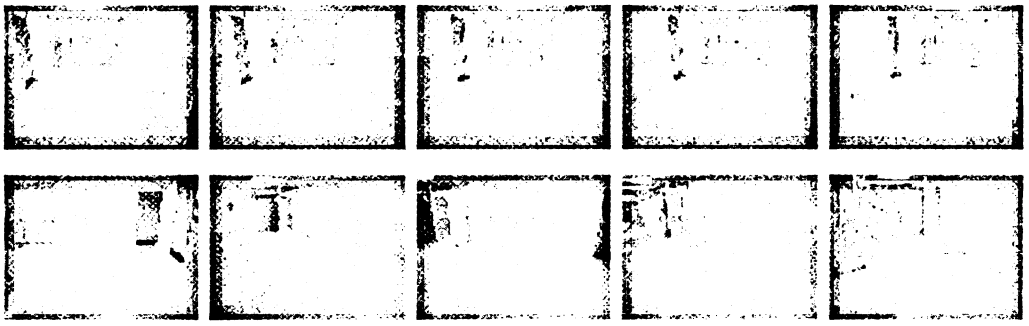


Figure 2.9: Some sample learning images in the straight corridors and the corners. The upper row is a straight corridor. The lower one is a left turn.

ing a joystick. The corresponding corrected heading directions were obtained using consecutive heading increments. The pictures at turns were grabbed in shorter steps, less than 0.5 meter per step, since more frequent heading updates are needed during a turn. Some of the sample learning images are shown in Fig 2.9.

2.6.2 MEFs, MDFs and their clustering effects

We show some experimental results to indicate quantitatively how the MEF and the MDF may perform very differently in classifying scenes. We computed MEFs and MDFs, respectively, from 210 images along a straight corridor and 108 images at a corner, all grouped into 6 classes. The first five classes are straight corridors classified corresponding to the next heading direction needed to recover the correct heading. That is, class 0 for 10°; class 1 for 5°; class 2 for 0°; class 3 for $-5°$; and class 4 for $-10°$. Class 5 consists of 108 images at the corner. Fig. 2.10 shows the first five MEFs and MDFs, respectively. As can be seen, first MEFs mainly record large area of contrast while MDFs record locations of edges with increasing spatial resolutions. Fig. 2.10(c) shows the learning samples in the subspace spanned by the first 2 MEFs

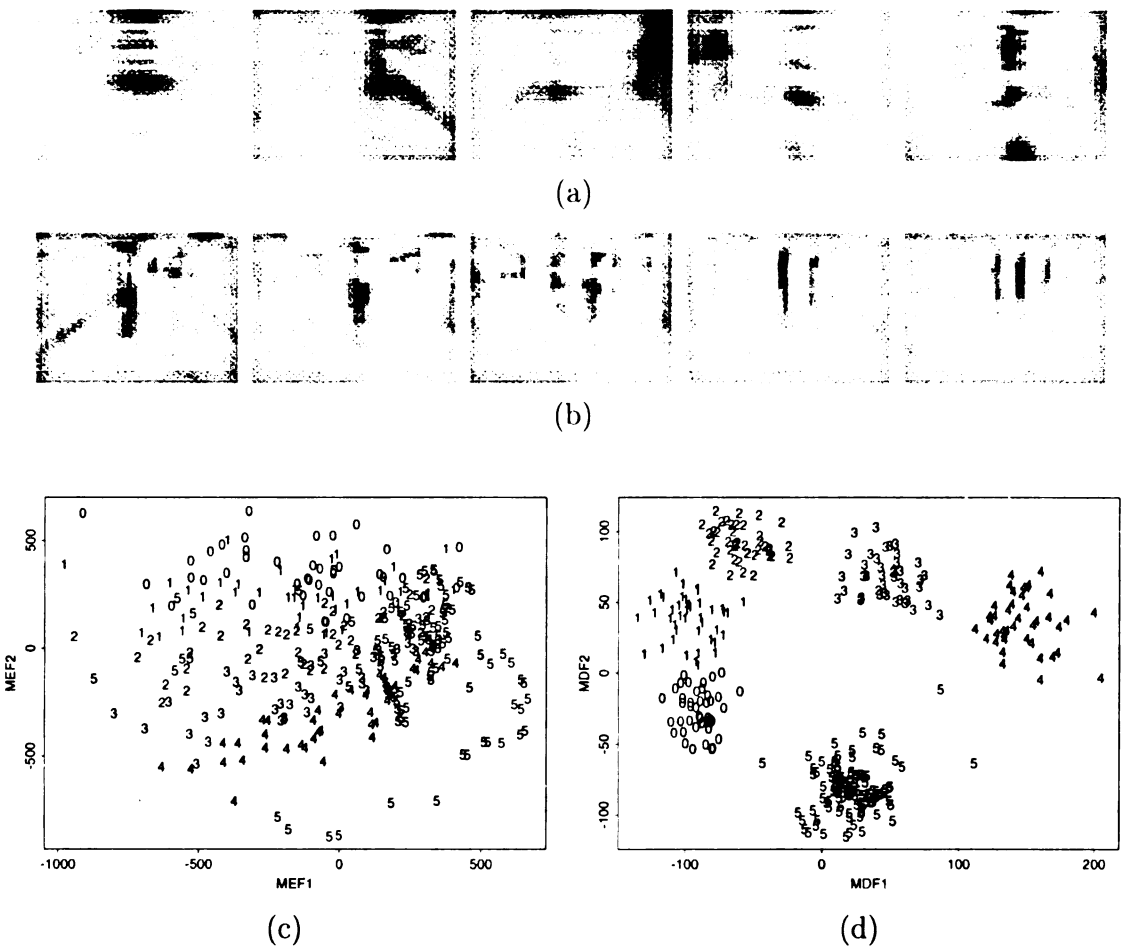
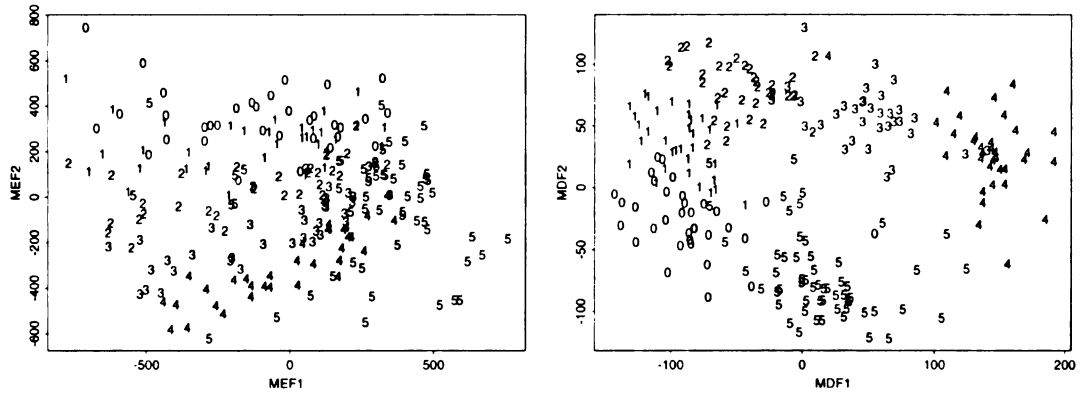


Figure 2.10: The difference between MEF and MDF in representing learning samples from a straight corridor and a corner. (a) The first five MEFs of the learning set. (d) The first five MDFs of the learning set ($\tau = 80\%$). (c) Learning samples represented in the subspace spanned by the first two MEFs. (d) Learning samples represented in the subspace spanned by the first two MDFs. The numbers in the plot space are the class labels of the learning samples.

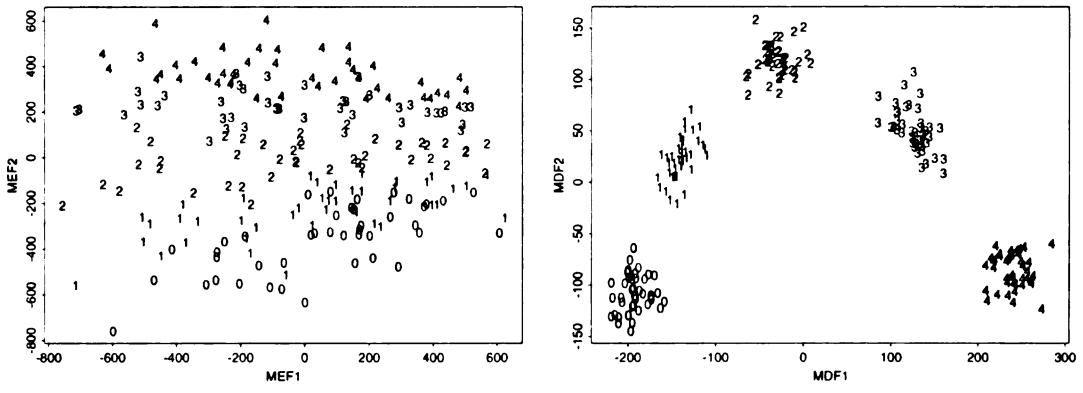


(a) (b)

Figure 2.11: Test images in the subspace spanned by the first two MEFs and MDFs, respectively, all of which are not included in the learning set. (a) The MEF subspace. (b) The MDF subspace.

and Fig. 2.10(d) shows them in that by the first 2 MDFs. As clearly shown, in the MEF subspace, the samples from a single class spread out widely and the samples of different classes tend to mix together. However, in the MDF subspace, the samples of each class are clustered more tightly and the samples from different classes are farther apart. From Fig. 2.10, we can see that to classify an unknown image from the same environment using the nearest neighbor rule, the MEF space is not as good as the MDF space.

To observe how the first two MEFs and MDFs, respectively, perform when they are used to represent images that have not been included in the learning sample set, we plot a set of 227 new images, of which 170 are from the straight corridor, 34 from the trained corner and 23 from a untrained corner, in Fig. 2.11. In Fig. 2.11(b) the structure of each class cluster is still visible in the MDF subspace, though not as tight as the case in Fig. 2.10(d).



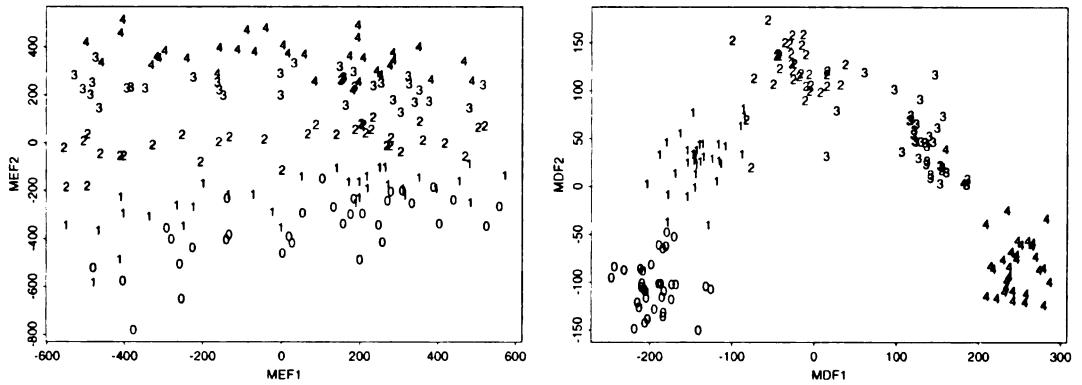
(a) (b)

Figure 2.12: Fewer classes in the learning set: using only straight corridor images for training. (a) Learning samples represented in the subspace spanned by the first two MEFs. (b) Learning samples represented in the subspace spanned by the first two MDFs. The numbers in the plot space are the class labels of the learning samples.

2.6.3 Better clustering in children

In the RPT, the children of a node divide the sample images handled by the parent node into smaller groups and each group has a smaller number of classes, in general. To see how fewer classes will allow the MDF to more effectively cluster classes, we computed MDFs separately, one set from straight corridor images and another set from corner images.

Using the first two MDF computed from the corresponding images, the straight corridor learning images are shown in Fig. 2.12 for the corresponding MEF and MDF subspaces, respectively. Comparing Fig. 2.12(b) with Fig. 2.10(d), we see that each class in Fig. 2.12(b) groups tighter and between class distance is larger. In Fig. 2.13 straight corridor images not in the learning set are plotted in the subspaces of MEF and MDF, respectively, that used only corridor images as the learning set. As we can predict, the test images now group more tightly than that in Fig. 2.11.



(a) (b)

Figure 2.13: Test images represented in the subspaces learned from a learning set with 170 corridor images only (without corner images). (a) The MEF subspace. (b) The MDF subspace.

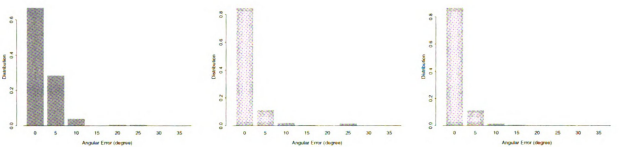
2.6.4 Effects on the tree size

For comparison purpose, two types of tree were experimented with, MEF RPT and MDF RPT. The former uses MEF features and the latter uses MDF features. However, since a pure MEF RPT does not perform well, as shown in the previous examples, we used MDF for the root level of the MEF RPT to classify coarse scene types, corridor and corner, before using MEFs for further classification.

Table 2.1 shows the distribution of nodes over the levels in the MDF tree and the MEF tree, respectively. Both trees used the same 318 learning images, 210 from the straight corridor and 108 from the corner. The MDF tree has only a total of 69 nodes, with only 35 leaf nodes; while MEF tree has a total of 635 nodes, with 318 leaf nodes. As expected, an MDF tree is typically smaller than the corresponding MEF tree, since the hyperplanes represented by MDF are more effective to cut along the boundary of classes.

MDF Tree		MEF Tree	
Level	Number of nodes	Level	Number of nodes
0	1	0	1
1	2	1	2
2	4	2	4
3	8	3	8
4	10	4	16
5	14	5	32
6	22	6	64
7	8	7	128
		8	208
		9	138
		10	34
Total	69	Total	635

Table 2.1: The sizes of MDF tree and MEF tree, trained with the set of 318 images.



(a) (b) (c)

Figure 2.14: Distribution of absolute angular error (degree) of retrieved heading directions.These two plots are based on queries using a test set of 204 images, of which 170 are from straight hallway and 34 from the learned corner. (a) Using a MEF RPT ($k = 1$). (b) Using a MDF RPT ($k = 1$). (c) Using a KNDB MDF RPT (k=4).

2.6.5 RPT Retrieval

Since we have taken a large number of test images that are not in the learning set, we conducted some "virtual runs." Fig. 2.14 shows the error distribution of retrieved heading directions for a set of 204 test images, when the trees are trained using the a disjoint set of 318 images, with the straight hallway and the corner. Comparing Fig. 2.14(a) and Fig. 2.14(b), we can see that the errors in retrieving MDF tree

Figure 2.15: Examples of retrieval. The first row: query images; the second row: retrieved using MDF RPT; the third row: retrieved from MEF RPT.

are relatively smaller, especially for the bin of 5-degree error, which implies better generalization power.

To give some intuition about what kind of images were retrieved using the RPT, some retrieval results are shown in Fig. 2.15.

2.6.6 Real runs

For the real runs, the chosen control signal has three components: corrected heading direction, speed and step size. When a new image is grabbed, the retrieval program searches down the RPT for the matching leaf node, and uses the corresponding stored control signal to control the robot. For the learning, τ in (2.5) is set to 5%. We trained Rome through three learning drives in which 363 learning images were taken, 280 of which from two straight corridors and 83 from a corner.

The image size for the MEF and MDF projections is 30×40, rescaled from its original 480×640 size with smoothing to reduce the effect of input noise. The total computer time for learning phase ranged from 4.5 to 7.5 minutes, as shown in

Workstation	Time (minutes)
SPARC-20	4.5
SPARC-10	7.5

Table 2.2: The total learning time for a set of 363 images.

Workstation	Average time per retrieval
SPARC-10	19.0ms
SPARC-2	37.0ms
SPARC-1	74.7ms

Table 2.3: Time for retrieving from the trained RPT, given a new input.

Table 2.2, depending on the SUN workstation used.

We let the Rome, learned using the MEF-RPT, navigate autonomously, at a walking speed, over 30 times along three straight corridors and two corners (including one corridor that has not been learned). All these over 30 drives were successful [11]. By Spring, 1995, the MDF RPT version had been implemented and it has been used for Rome testing and demonstrations since then. The MDF version exhibited more consistent direction history than the MEF version. During various tests and demonstrations, there were people walking along the corridors, but Rome was not bothered because it recognized the entire scene instead of a particular type of landmarks (e.g., road edges).

Table 2.3 shows the time used for retrieval from the trained MEF RPT. Clearly, a SPARC-1 is fast enough (13Hz) for real-time navigation in this experiment. The MDF RPT is even faster.

Fig 2.16 shows a trajectory of a sample navigation and Fig 2.17 shows some images to illustrate the environment.

To test the scalability performance of the algorithm, we collected a large set of

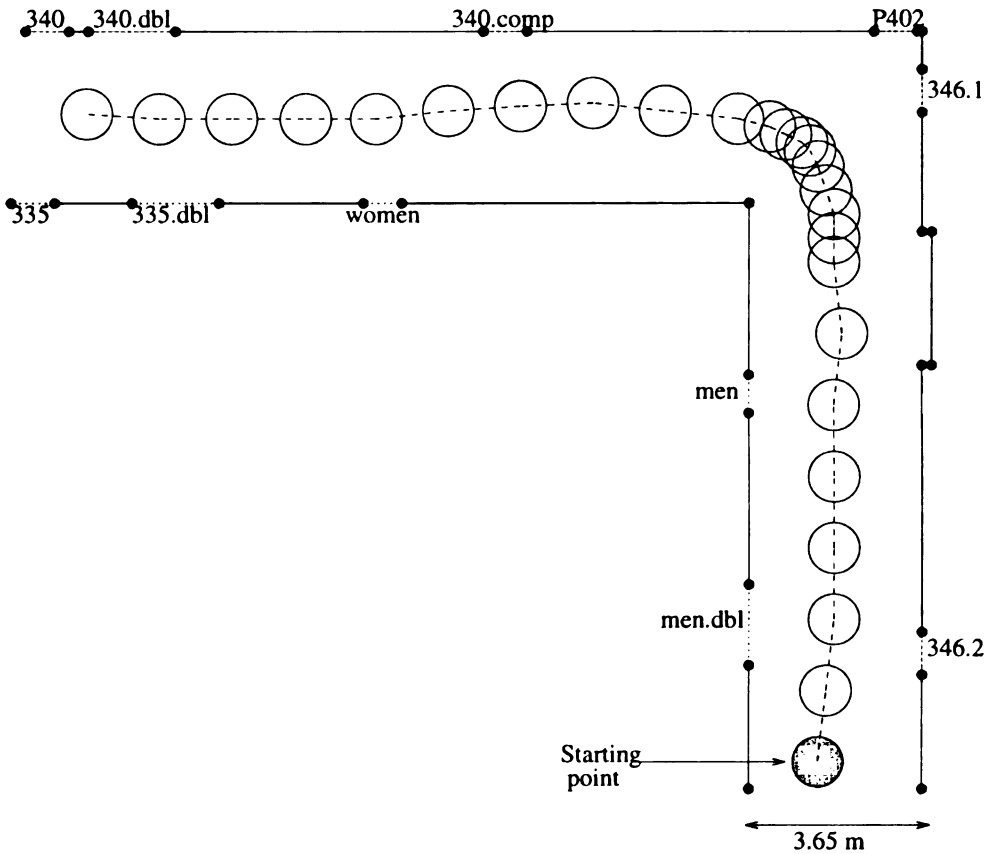


Figure 2.16: A sample of autonomous navigation in the trained corridor: two straight corridors and one corner. The circles indicate the locations where images are taken.

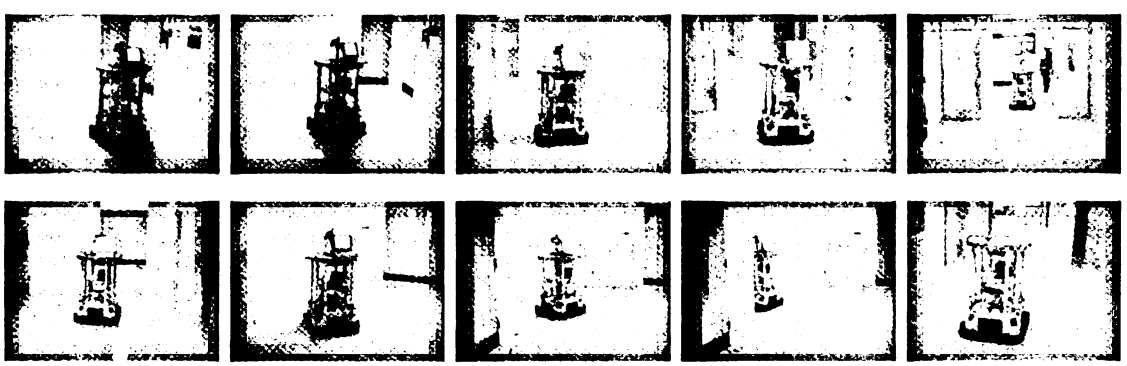


Figure 2.17: Rome navigates automatically at two corners: the first corner in the 1st row and the second corner in the 2nd row.

Per retrieval	MEF tree	flat version	
		in MEF space	in image space
Time (in milliseconds)	27.7	738.3	2853.7
Time ratio w.r.t MEF tree version	1	26.7	103.0

Table 2.4: The comparison in retrieval timing of SPARC-10 between MEF tree version and flat versions, where tree structures are not used. A set of 2850 learning images were used in training.

2850 images from various corridor sections for training and got the timing data for comparison between the tree version and the flat version where the nearest neighbor was found in the corresponding space. The computer timing data are shown in Table 2.4. From this table, it is clear that the use of MEF tree can greatly speed up the retrieval and a real-time speed can be achieved only with the tree version.

2.7 Comparison with Two ANN-based Approaches

Artificial neural networks have been successful in various engineering applications, including vision-based navigation. ALVINN [94] and ROBIN [105], [106], developed by CMU and Univ. of Maryland respectively, are two well known systems that map visual inputs directly into output control signals. ALVINN uses a two-layer feedforward network while ROBIN uses a RBF network. To compare our method and these two approaches, we use MATLAB to do the simulation.

We used the same output scheme as ALVINN and ROBIN: the output pattern of each training sample is a Gaussian distribution peaked at the desired corrected heading. In our simulation, the output layer has 21 nodes with a resolution of two

Figure 2.18: Sample training input–output pair: the top part is a 30×40 image; the last row is the associated output signal, which is a Gaussian distribution peaked at the corrected heading direction.

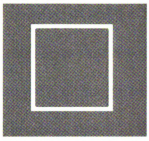
degrees apart. The i-th ($i = 1, 2, \cdots, 21$) output node corresponds to $(i - 11) \times 2$ degrees in corrected heading. A sample training input–output pair is shown in Fig. 2.18. After the neural network is trained, the simulated output heading is taken as the peak of a Gaussian fit to the outputs of neurons at output layer.

Three sets of data are used in this study:

Set 1 Real images from straight hallway sections: a set of 100 (30×40)-pixel images with five different corrected heading directions: $-10°$, $-5°$, $0°$, $5°$ and $10°$. Each of these five classes has 20 samples.

Set 2 A full set of 318 (30×40)-pixel real images are used. This set of images are used in our training of SHOSLIF-N for vision-based navigation. It includes 210 images from straight hallway sections and 108 images from corner sections.

Set 3 Synthesized data set: Given a trajectory map, synthesized sample images are generated with different orientations and translations. For a square trajectory, a typical set consists of 100 (30×40)-pixel synthetic images with five different headings: $-10°$, $-5°$, $0°$, $5°$ and $10°$. Some sample images are shown in Fig. 2.19.



(a) Road map. The dark area is nonroad and the white area is the road.

(b) Synthesized sample images.

Figure 2.19: Some synthesized sample images generated At different locations and orientations along the simulated navigation path. Images were generated under a perspective projection camera model.

These three sets of images are used to test different learning approaches under various situations.

2.7.1 Simulation of a feedforward neural network

For simulation of a two-layer feedforward neural network, we used "trainbpx" with adaptive learning rate available in the MATLAB neural network toolbox. Similar to ALVINN, we use four to nine hidden nodes in our simulation.

When the size of problem is small, i.e. the number of images is small and the size of image is also small, the training of feedforward network went successfully: it found a reasonable solution within a few trials starting from initial random guesses. But we found that when the input dimension is high, e.g. 30×40, the neural network would not converge to reasonable solutions with random initial weights. We run our MATLAB script files for 100 times, each time with a different random guess of initial weights. The training epoch is set to 10,000, large enough to converge to a local

minimum. Fig. 2.20 shows the training record for Set 1. Here we use four hidden nodes as indicated in [95]. Fig. 2.20 shows that the weights does sometimes converge to a good solution for the training set, although only a few trials provide solutions with small sum-of-squares (SSE) errors as shown in Fig. 2.20(a). Our simulation with synthetic image set got consistent results: the learning process can easily get stuck to poor local minimums.

Fig. 2.21 shows 100 trial training record for Set 2, the error histogram corresponding to the best SSE solution marked with a circle in Fig. 2.21 is shown in left column of Fig. 2.22(b). Again, the convergence to a reasonable solution is difficult. Training of Set 2 with 10,000 epoches for each random initial guess took about 75 minutes on a SUN SPARC-Ultra-I. To get a reasonable solution, typically about 2,000 epoches were needed. It took about 15 minutes for batch training of Set 2 with 2,000 epoches. All 100 random trials shown in Fig. 2.21 took 5.2 days of CPU time.

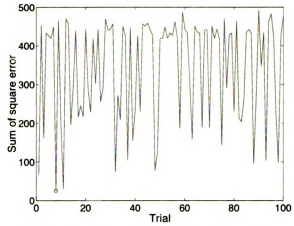
2.7.2 Simulation of a radial basis function network

The implementation of ROBIN is different from a typical RBF network in the following aspects:

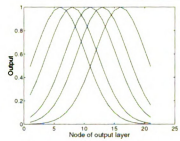
1. Normalization. The responses of the receptive fields are normalized to have unit sum response for all neurons in the first layer. This normalization improves the interpolation capability, as shown by references sited in [106].

2. Center selection.

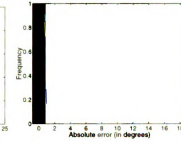
   Rosenblum and Davis [105], [106] used K-means clustering algorithm for center



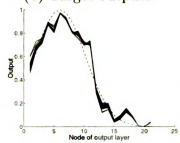
(a) Sum of square error for 100 trials.



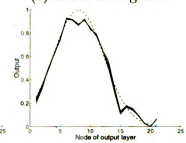
(b) Target outputs.



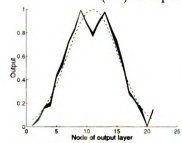
(c) Error histogram.



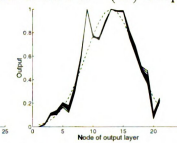
(d1) Outputs of class 0.



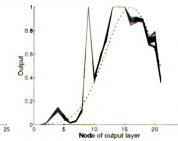
(d2) Outputs of class 1.



(d3) Outputs of class 2.



(d4) Outputs of class 3.



(d5) Outputs of class 4.

Figure 2.20: Training with Set 1. 100 trials with different random guesses of initial weights. (a) shows the record of sum of square error (SSE). The trial with minimal SSE is marked with a circle. (b) shows the target output for five classes of training samples with different corrected headings: $-10°$, $-5°$, $0°$, $5°$ and $10°$. (c) is the error histogram of retrieved heading dierctions. (d1) to (d5) show the network output overlaid on target output for each class: simulated outputs and target outputs are plotted in solid lines and dotted lines respectively.

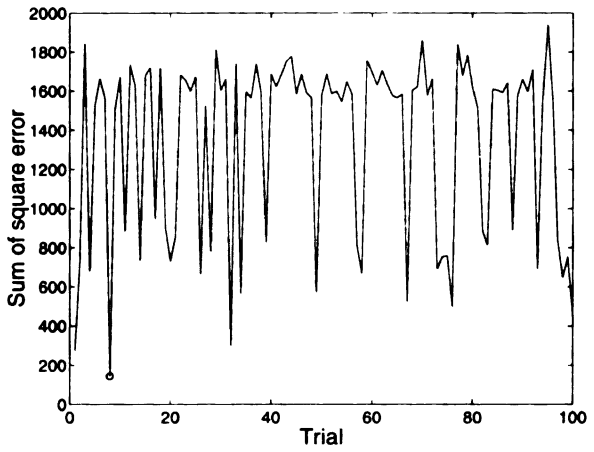


Figure 2.21: Training record for Set 2: Sum of square error for 100 trials. The best SSE solution is marked with a circle.

selection but the results were not satisfactory. So they used "forced-clustering" by manually assigning subset of input patterns as centers. This center selection process needs tedious human-machine interaction.

For our simulation of a RBF network, we manually select a subset of typical views from the training set. For Set 1 and Set 3, we got zero error in heading direction for the training set when about 20 centers were chosen. For Set 3, we used similar number of centers as in [106]: 60 centers were manually selected. ROBIN [106] used 53 centers. We trained the RBF network with images in Set 3 and tested with the set of 204 images used in Fig. 2.14. After several trials with manual center selection, the error histograms of the training and test sets for the most reasonable selected centers are shown in Fig. 2.22(c). After manual center selection, the training of Set 2 took about 20 minutes on a SPARC-Ultra-I.

2.7.3 Comparison of SHOSLIF-N with two ANN-based approaches

We used a realistic image set — Set 3 to do the training. Another disjoint set of 204 images, the same data set used in Fig. 2.14, was used for testing. The error histograms are plotted in Fig. 2.22, for all three different approaches. SHOSLIF-N always gets perfect retrieval for the training set, so the error histogram concentrates on a single bin with zero error. Comparing Fig. 2.22(b) with Fig. 2.22(c), we can see that the feedforward neural network and the RBF network obtain compatible error performance in heading for the test set, but the RBF network is a little bit worse for the training set. For both the training and test sets, SHOSLIF-N gives better heading accuracy.

According to our experience in simulation, SHOSLIF-N is advantageous in at least two aspects: (1) ease of training; (2) good performance after it is trained. Although the feedforward network could give reasonable solution, many random trials are needed for initial weights. The RBF network does provide comparable performance. However, the manual center selection is tedious and we don't even know how many centers will be sufficient. When most of the training samples are used as centers, the RBF network will act like a nearest neighbor estimator. But then the computation complexity will be prohibitive when the number of training samples is large. On the other hand, our SHOSLIF-N has to pay extra cost in storing training samples. The memory size of the SHOSLIF-N tree is 3.1M bytes for Set 2. It seems that the payoff from extra storage enables SHOSLIF-N to reach a good performance

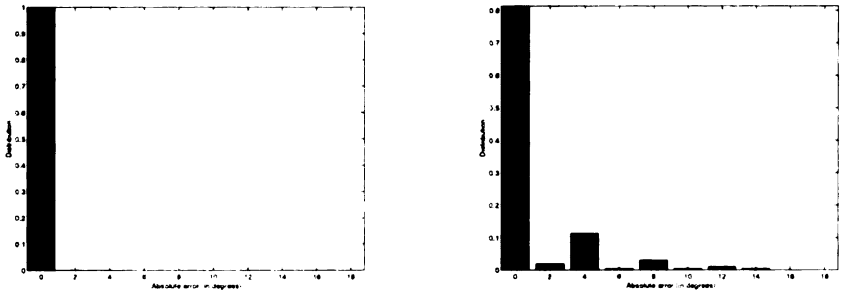
accuracy with a real-time speed. The training is also relatively fast.

2.7.4 Mapping SHOSLIF-N into ANN architecture

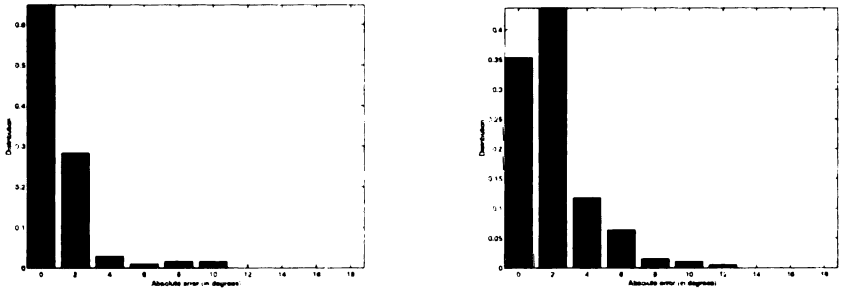
Learning algorithm used in SHOSLIF can be easily mapped into neural network architecture. The most dominant eigenvector for binary partition of each inner node can be computed by various neural learning algorithms (e.g. Oja, 1992 and references therein). The tree structure of SHOSLIF can also be mapped into neural network architecture [108]. Suppose that we have a binary tree with t splits and $t+1$ leaves. A network in which the first hidden layer of t nodes computes the split functions, and the second layer has a node for each of the $t+1$ paths to a leaf which ANDs the outputs along the nodes on the same path. The output layer then ORs the leaves with similar desired outputs.

2.8 Conclusions and Future Work

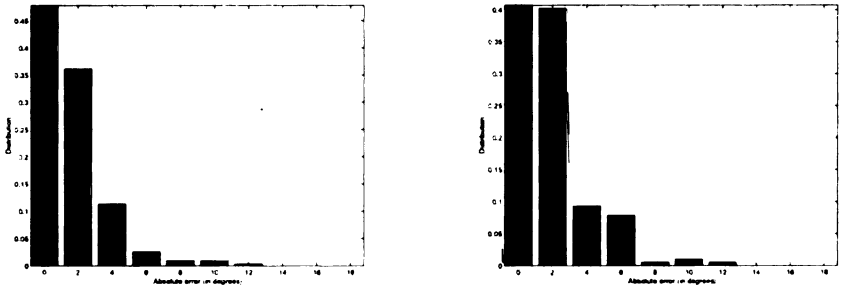
The framework presented here does not restrict the scene type and thus is potentially applicable to various scenes. Instead of relying on a particular scene feature, it uses automatically derived spatially global features for decision making, and thus is relatively insensitive to local and small scene changes, such as road edges being occluded by objects or the presence of passers-by. The MDF version of the RPT seems to generalize better than the MEF version with its relatively tighter class clusters and a smaller tree size. Due to availability of inexpensive, large hard disks and the fast logarithmic retrieval that is made possible by the RPT, it seems that storing a relatively



(a) SHOSLIF-N.



(b) A feedforward network.



(c) A RBF network.

Figure 2.22: Comparison of three approaches: SHOSLIF-N, the best feedforward network and the best RBF network we obtained. A set of 318 images are used for training and a disjoint set of 204 images are used in testing. The error histogram for the training set is shown in left column while the error histogram for the test set is shown in right column.

large number of learning images is not unreasonable. The method presented seems particularly attractive for situations where consistent floor edges are not available or passers-by are not avoidable.

The future work includes experiments with a wider variety of driving environments. Since this approach requires a significant number of images for learning, real time incremental learning is a future direction for this line of research.

Chapter 3

On-Line Incremental Learning for Vision-Guided Navigation

In last chapter a batch method of training RPT for vision-based navigation is discussed. In this chapter the incremental learning issue is addressed. Since binary RPT requires only the most dominant eigenvector at each inner node for finer partition, it turns out that the learning mechanism discussed in last chapter can be done incrementally with low computational complexity.

3.1 Introduction

Autonomous vision-guided navigation has been a challenging task. In the past, most of the autonomous navigation systems relied on tracking specific features, such as lane markings of outdoor roads, floor or ceiling edges of hallways, while others detected road regions based on features such as color or texture. All of these systems had a

strong reliance on an *a priori* model about the road's appearance, and hand-crafted rules were implemented as the model fitting algorithm. Unfortunately, these models and rules are not always suited due to changes of environmental conditions. Some roads may not have clear lane markings, and some none at all. Variations in illumination and road condition can often invalidate the underlying assumptions used in the vision algorithms.

In order to deal with changes in environmental conditions, it is necessary for the system to employ adaptive mechanisms [20, 81, 94, 96]. According to different design goals and working environments, adaptive mechanisms may vary from simple adaptive color thresholding [72] to complex machine learning using, e.g., artificial neural networks [94, 57].

For outdoor navigation, ALVINN [94], an artificial neural network trained by a back-propagation training algorithm, learns characteristic features of particular roads under current conditions. It has been successfully tested in a variety of road conditions. ELVIS [44] uses an eigen-subspace method as an alternative of the artificial neural network in ALVINN. The availability of lane markings can be used to greatly enhance the adaptation performance of a driving system, as shown in the RALPH [96]: Possible road curvatures are predicted, and the correct one with the highest signature score is picked up as the winner. This technique leads to a successful lateral position handler.

In this chapter, we explore the incremental learning issue in the framework of SHOSLIF-N [138]. Incremental learning is a natural way in which humans gain their driving experience. With incremental learning, the training can be done on-line.

This is very important in machine learning for autonomous navigation, since the total sensed information in training is huge and highly redundant, and may exceed the limit of physical storage device. A batch training method is less practical and maybe less efficient.

The training of artificial neural networks [47] has built-in incremental learning capability, e.g. pattern-by-pattern updating. However, since these computations with neural networks are iterative using the same set of images many times, it is hard to implement learning in real time, which is essential in a domain like autonomous driving. We proposed in [12, 139] an incremental learning algorithm by using eigen-subspace learning with a tree structure. Here we present new computation methods that reduce maximal response time, which is critical in on-line learning.

The remainder of this chapter is organized as follows: Section 3.2 describes efficient dominant eigenvector computation for incremental learning. Section 3.3 presents how to realize the incremental update. Section 3.4 gives the incremental learning algorithm. The experimental results and conclusions are reported in Section 3.5 and Section 3.6, respectively.

3.2 Incremental Learning and Dominant Eigenvector Computation

First, we briefly describe our autonomous navigation scheme within the framework of SHOLSIF-N [138].

3.2.1 Navigation as a content-based retrieval problem

A navigator can be regarded as a complicated function which maps high-dimensional input images into low-dimensional output control signal. In the learning phase, a set of training images is used to build a recursive partition tree (RPT), in which each learned image has its corresponding control signals. At each inner node of the RPT, the eigen-subspace method is used to extract characteristic features for further partition of input image space. This process is done recursively down the tree until the current node contains only images with a similar control signal. In the performance phase, each newly grabbed image is used to retrieve the best-matched image from the trained RPT, and the control signal associated with the retrieved image is used as the next steering control signals.

Notice that ELVIS [44] also uses an eigen-subspace method for outdoor navigation. In ELVIS input image and its output vector are treated as a long monolithic vector. Here we treat input image and output steering vector separately, and we deal with on-line incremental learning while ELVIS does not.

We chose binary partition for its simplicity. At each inner node, further space partition is based on images' projection onto a single eigenvector associated with the largest eigenvalue, which captures the largest variance in the sample images assigned to the node. Therefore, at each inner node, we need only obtain the eigenvector associated with the largest eigenvalue. This leads to an efficient on-line incremental learning algorithm.

The incremental eigenspace update algorithm has been discussed in [43, 78, 85].

The incremental update of eigenspace using QR decomposition is discussed in classic textbook by Golub and Van Loan [39]. All of these discussions focus on incremental update of an eigen-subspace. However, since we only need to solve the eigenvector associated with the largest eigenvalue, much simpler and more efficient methods can be used. The power method, Lanczos algorithm, and the Rayleigh quotient conjugate gradient method [34, 39, 73, 104, 146] are among the most popular ways in computing dominant eigenvalue/eigenvector of a real symmetric matrix.

3.2.2 The power method

The power method is one of the oldest methods in solving an eigensystem [39, 73, 141]. Suppose that a matrix A is diagonalizable, so that $E^{-1}AE = \text{diag}(\lambda_1, \cdots, \lambda_n)$ with $E = [e_1, \cdots, e_n]$, and $\lambda_i (i = 1, \cdots, n)$ are in decreasing order. Let u_0 be an arbitrary vector and let the sequences v_s and u_s be defined by the equations

$$v_{s+1} = Au_s, \quad u_{s+1} = v_{s+1}/\|v_{s+1}\|, \tag{3.1}$$

where $\|x\|$ is used to denote the l_2 norm. Clearly, we have

$$u_s = A^s u_0/\|A^s u_0\| \tag{3.2}$$

and if we write

$$u_0 = \sum_{i=1}^{n} \alpha_i e_i, \tag{3.3}$$

then apart from the normalizing factor, u_s can be written as

$$\sum_{i=1}^{n} \alpha_i \lambda_i^s e_i = \lambda_1^s \left[\alpha_1 e_1 + \sum_{i=2}^{n} \left[\frac{\lambda_i}{\lambda_1} \right]^s e_i \right]. \tag{3.4}$$

If $\|\lambda_1\| > \|\lambda_2\| \geq \|\lambda_3\| \cdots \|\lambda_n\|$, provided that $\alpha_1 \neq 0$, we have $u_s \rightarrow e_1/\|e_1\|$.

If there are a number of eigenvectors corresponding to the dominant eigenvalue, $\lambda_1 = \lambda_2 = \cdots = \lambda_r$ and $\lambda_r > \lambda_{r+1} \geq \cdots \geq \lambda_n$, the procedure still converges, since we have

$$A^s u_0 = \lambda_1^s \left[\sum_{i=1}^{r} \alpha_i e_i + \sum_{i=r+1}^{n} \alpha_i \left[\frac{\lambda_i}{\lambda_1} \right]^s e_i \right] \sim \lambda_1^s \sum_{i=1}^{r} \alpha_i e_i.$$

The iterates tend to converge to some vector lying in the subspace spanned by the eigenvectors $e_1, \cdots, e_r$. When the ratio λ_{r+1}/λ_r is close to unity, the convergence is slow. But the iterates still tend to a meaningful vector. The term in square brackets of equation (3.2) can be regarded as a weighted sum of the eigenvectors $e_1, \cdots, e_n$, with coefficients $\alpha_1, \alpha_2(\lambda_2/\lambda_1)^s, \cdots, \alpha_n(\lambda_n/\lambda_1)^s$. The contribution from eigenvector e_i tends to shrink by a factor of λ_i/λ_1, as the power method iterates. Only those dominant eigenvectors have significant contributions.

A random vector can be chosen as u_0, since α_1 is always non-zero due to the truncation error. In practice, a fixed number of iterations usually give satisfactory results. Letting the number of iterations be K_p, the time complexity for the power method is $O(n^2 K_p)$.

3.2.3 The Rayleigh quotient conjugate gradient (RQCG) method

Let A be a real symmetric matrix of order n, $\lambda_1, \lambda_2, \cdots, \lambda_n$ be nonincreasing eigenvalues and their associated eigenvectors be $e_1, e_2, \cdots, e_n$ respectively. Then,

$$\lambda_1 = \max_x \frac{(Ax, x)}{(x, x)} = \max_x \frac{(Ax, x)}{\|x\|} \tag{3.5}$$

where $(\cdot, \cdot)$ denotes inner product of two vectors, and $\|\cdot\|$ defines the norm of a vector.

Let $R(x) = \frac{(Ax,x)}{\|x\|}$, R(x) is called the Rayleigh quotient. From eq. (3.5), it is clear that the dominant eigenvector is the vector that maximizes the Rayleigh quotient. A nice property of the Rayleigh quotient $R(x)$ is that its critical points are either unstable saddle points or the global maximum. A practical way to solve the maximization problem of (3.5) is the conjugate gradient method.

Let $g(x)$ be the gradient of the Rayleigh quotient $R(x)$. Then

$$g(x) = 2Ax - 2R(x)x.$$

From any approximation x_k of e_1, an improved approximation x_{k+1} is computed by

$$x_{k+1} = x_k + \delta_k p_k.$$

Here, p_k is the search direction found by the conjugate gradient technique,

$$p_k = -g_k + \beta_k p_{k-1}.$$

The value of δ_k is chosen by solving

$$\max_{\delta_k \in R} R(x_k + \delta_k p_k). \tag{3.6}$$

The closed-form solution of (3.6) is

$$\delta_k = (-b_k - \sqrt{b_k^2 - 4a_k c_k})/(2a_k),$$

where

$$\begin{aligned} a_k &= (x_k, p_k)(p_k, Ap_k) - (x_k, Ap_k)(p_k, p_k), \\ b_k &= (p_k, Ap_k) - (x_k, Ax_k)(p_k, p_k), \\ c_k &= (x_k, Ap_k) - (x_k, Ax_k)(x_k, p_k). \end{aligned} \tag{3.7}$$

The value of β_k can be computed by various ways [146, 77]. A practical choice is the *Polak-Ribiere method*, where

$$\beta_k = \frac{(g_{k+1} - g_k, g_{k+1})}{(g_k, g_k)}$$

is used.

The initial dominant eigenvector is randomly chosen on the unit n-dimensional sphere. The above RQCG method is an iterative method, each iteration taking $O(n^2)$ time. The iteration is looped up to a certain number of times or until the change in R_k is very small. In practice, the RQCG method typically converges rapidly, much faster than the power method. Letting the number of the RQCG iterations be K_r, the time complexity for the RQCG method is $O(n^2 K_r)$.

Both the power method and RQCG method are easy to implement, and have a complexity of $O(n^2 K)$ where K is the fixed maximal iterations. We use K_p and K_r to denote the iteration number of the power and RQCG methods respectively.

Another two candidates in computing dominant eigenpair of a real symmetric matrix are the accelerated power method [104] and Lanczos algorithm. Writing a numerical stable Lanczos algorithm for this purpose is highly nontrivial.

3.3 Update for Incremental Learning

Let $\{u_i \mid i = 1, 2, \cdots, k\}$ be the set of images at an inner node, and its mean vector and covariance matrix be m_k and V_k, respectively. If we let $\bar{u}_i = u_i - m_k$ and $\bar{U}_k = [\bar{u}_1, \bar{u}_2, \cdots, \bar{u}_k]$, then $V_k = \frac{1}{k}\bar{U}_k\bar{U}_k^t$. Assuming that u_i is represented as a 1D vector with length d, $\bar{U}_k\bar{U}_k^t$ is of dimensionality $d \times d$. In our vision-based navigation, the number of images at an inner node, k, typically is smaller than the number of pixels d. It can be easily shown that if Φ is an eigenvector of the covariance matrix V_k with $V_k\Phi = \lambda\Phi$. Then $\bar{U}_k^t\Phi$ is an eigenvector of $\frac{1}{k}\bar{U}_k^t\bar{U}_k$ with the same eigenvalue λ. Therefore instead of computing the eigen-system of a $d \times d$ matrix V_k, we can

calculate the eigen-system of a smaller $k \times k$ matrix $W_k = \frac{1}{k} U_k^t U_k$.

3.3.1 Insert a sample

The mean vectors m_k can be updated incrementally:

$$m_{k+1} = \frac{1}{k+1}[k m_k + u_{k+1}]. \tag{3.8}$$

Letting $\delta m_k = m_{k+1} - m_k$, W_{k+1} can be updated incrementally:

$$\begin{aligned} W_{k+1} &= \frac{1}{k+1} \left\{ \begin{bmatrix} kW_k & \bar{U}_k^t \bar{u}_{k+1} \\ \bar{u}_{k+1}^t \bar{U}_k & \bar{u}_{k+1}^t \bar{u}_{k+1} \end{bmatrix} - \left[\bar{U}_{k+1}^t \delta m_{k+1}, \cdots, \bar{U}_{k+1}^t \delta m_{k+1} \right] \right. \\ &\left. - \begin{bmatrix} \delta m_{k+1}^t \bar{U}_{k+1} \\ \vdots \\ \delta m_{k+1}^t \bar{U}_{k+1} \end{bmatrix} + \begin{bmatrix} \delta m_{k+1}^t \delta m_{k+1} & \cdots & \delta m_{k+1}^t \delta m_{k+1} \\ \vdots & \vdots & \vdots \\ \delta m_{k+1}^t \delta m_{k+1} & \cdots & \delta m_{k+1}^t \delta m_{k+1} \end{bmatrix} \right\}. \end{aligned} \tag{3.9}$$

The above update rule for inserting a sample can be done with time complexity $O(k^2 + kd)$. Since in our application typically we have $d \gg k$, the time complexity for inserting or deleting a sample is $O(kd)$.

3.3.2 Reconstruction of W matrix from parent node's W matrix

Let p be an inner node of RPT with a child node q, W be $p \rightarrow W$ and $\widehat{W}$ be $q \rightarrow W$, respectively. Suppose that p has n samples $\{u_i \mid i = 1, 2, \cdots, n\}$ with mean vector m,

and $\hat{n}$ of p's samples go to its child node q with mean vector $\widehat{m}$: u_{l_i} $(i = 1, 2, \cdots, \hat{n})$. Let $A_{ij} = (u_i - m)^t(u_j - m)$ and $\hat{A}_{ij} = (u_i - \widehat{m})^t(u_j - \widehat{m})$. Then

$$W = \frac{1}{n}[A_{ij}], \quad i, j = 1, 2, \cdots, n,$$

and

$$\widehat{W} = \frac{1}{\hat{n}}[\hat{A}_{ij}], \quad i, j = l_1, l_2, \cdots, l_{\hat{n}},$$

where $[A_{ij}]$ and $[\hat{A}_{ij}]$ denote matrices with elements A_{ij} and $\hat{A}_{ij}$, respectively.

We know that

$$A_{ij} = (u_i - m)^t(u_j - m) = u_i^t u_j - m^t u_j - u_i^t m + m^t m, \quad i, j = 1, 2, \cdots, n, \tag{3.10}$$

and

$$\hat{A}_{ij} = (u_i - \widehat{m})^t(u_j - \widehat{m}) = u_i^t u_j - \widehat{m}^t u_j - u_i^t \widehat{m} + \widehat{m}^t \widehat{m}, \quad i, j = l_1, l_2, \cdots, l_{\hat{n}}. \tag{3.11}$$

Combining (3.10) and (3.11), $\hat{A}_{ij}$ can be expressed as

$$\hat{A}_{ij} = A_{ij} + (m - \widehat{m})^t u_j + u_i^t(m - \widehat{m}) + (\widehat{m}^t \widehat{m} - m^t m), \quad i, j = l_1, l_2, \cdots, l_{\hat{n}}. \tag{3.12}$$

The computation of $\widehat{W}$ using (3.11) can be done with $O(\hat{n}^2 d)$ complexity, while the reconstruction of $\widehat{W}$ with (3.12) using available W takes only $O(\hat{n}d)$ time.

3.3.3 The incremental RQCG method

Instead of starting from a random initial guess of dominant eigenvector, the RQCG method can be done in an incremental way: estimated dominant eigenvector of W_k can be used to generate a good initial dominant eigenvector of W_{k+1}, and the iteration time can be reduced significantly. We use K_{ri} to denote the iteration times of the incremental RQCG method.

From (3.9) we know that when $k \gg 1$, we can assume that δm_{k+1} is very small and thus the second, third and fourth terms in RHS of (3.9) can be ignored. In this case we have

$$W_{k+1} \approx \frac{1}{k+1} \begin{bmatrix} kW_k & \bar{U}_k^t \bar{u}_{k+1} \\ \bar{u}_{k+1}^t \bar{U}_k & \bar{u}_{k+1}^t \bar{u}_{k+1} \end{bmatrix}.$$

Ignoring the scale factor, the only difference between W_{k+1} and W_k is the newly appended $(k+1)$-th row and $(k+1)$-th column. Because of *the momentum effect*, the true dominant eigenvector of W_{k+1} is close to the true dominant eigenvector of W_k appended with a zero in the $(k+1)$-th component. So the estimated dominant eigenvector of W_k appended with a zero can be used as the initial dominant eigenvector of W_{k+1}.

3.4 Incremental Learning Algorithm (ILA)

We modify our RPT tree [138] so that it can be used for incremental learning. The RPT has to be built dynamically as more input images come, and whether each coming image should be ignored or not should be decided on-line. Here we briefly

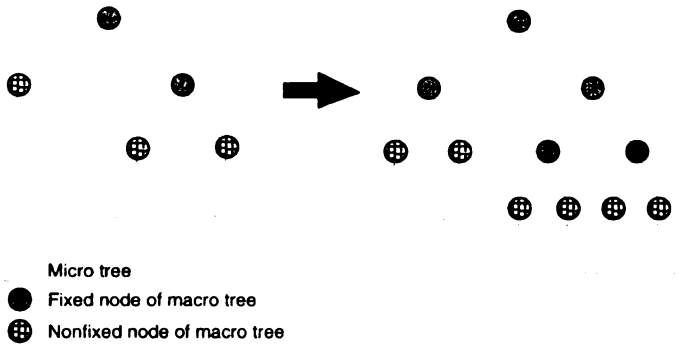


Figure 3.1: The evolution of a incremental learning tree. In this example, three nonfixed nodes of macro tree are evolved into fixed nodes.

describe our incremental learning algorithm, further details can be found in [12, 139].

Two parameters P_1 and P_2 are chosen. When the number of images going to a node reaches P_1, the dominant eigenpair is computed using the power method and the node is fixed in subsequent learning. Its child nodes are created. But the number of images flowing into a node may still be large. In order to speed up the retrieving, a second type of RPT, *a micro tree*, is introduced. The former RPT is therefore called *a macro tree* for easy discrimination. Both types of RPT's are built recursively by projecting along the dominant eigenvector. P_2 is used to decide when to stop growing child nodes. When the number of images going to a node is less than P_2, the node stops growing and becomes a leaf node.

The structure of an incremental learning RPT is shown in Fig. 3.1. There are three types of nodes in the RPT: fixed nodes of macro tree, nonfixed nodes of macro tree and nodes of micro tree. The number of images going to a nonfixed macro node lies between P_2 and P_1. When a nonfixed node gets fixed, the dominant eigenvector is computed and its child RPT's are rebuilt. Each new image with associated control signals is checked with a cross validation procedure to decide whether it should be

ignored or not. A near neighbor of the input image is obtained by using content-based retrieving of the learned RPT. If the associated retrieved control signals are within tolerance of the desired control signals, the input image is ignored. Otherwise, the incremental learning is conducted.

The following items are maintained for each node:

```
fixed                    /* 1: fixed; 0: nonfixed */
k                        /* number of images flowing into the node*/
*list                    /* index numbers of k images*/
m_k                      /* incrementally updated mean vector*/
M_0                      /* mean vector for retrieving*/
U_k = [u_1 u_2 ... u_k]^t /* accepted learn images flowing into the node */
W_k = U_k^t U_k          /* (1/k) U_k U_k^t is the covariance matrix*/
H                        /* dominant eigenvector
```

The use of hierarchical partition structure of RPT can achieve efficient retrieving. The retrieving procedure is shown below.

```
RPT Search Algorithm: Given an input image u, find its corresponding
control signal c_r.
call retrieve(root, u);
retrieve(node, u) {
    if node is a leaf {
    do a linear search in images stored in node → U,
    find the nearest image u_r = arg min_i ||u − u_i||;
    return (c_r=the associated control signal of u_r);
    }
    else {
    if ((node → H)^t(u − node → M_0) ≤ 0) retrieve(node → leftchild );
    else                                  retrieve(node → rightchild);
    }
}
```

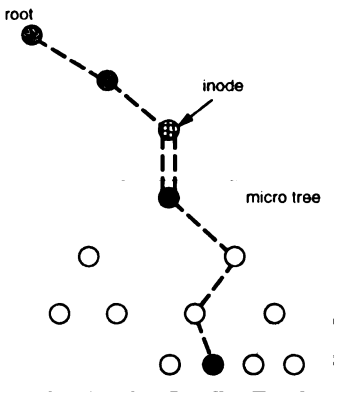


Figure 3.2: Update RPT when a new image is accepted. The search path is represented by dashed lines. The updates are done only at two nodes: the first nonfixed macro node on the search path and the leaf micro node. They are marked as "black nodes".

For each newly accepted learn image, the update of RPT can be done in the way shown in Fig. 3.2. First we locate *inode*, the first nonfixed macro node in the search path, insert the image into the inode, and then do the incremental update. If *inode* is not a leaf micro node, find the leaf micro node in the search path and do the update. Given two fixed numbers P_1 and P_2, the following procedure build the recursive partition tree (RPT) incrementally.

1. Grab a new image u and get its associated desired control signal c.

2. Retrieve the learned RPT by calling c_r=retrieve($root$, u). If the retrieved control signal c_r satisfies the condition: $\|c-c_r\| \leq T_c$ (where T_c is a tolerance threshold), u is ignored, go back to step 1.

3. Incremental learning:

   Locate the first nonfixed macro node along the search path in retrieve($root$, u), denote it as $inode$;
   For each $node$ between $inode$ and the leaf node in the search path of retrieve($root$, u)
   {
   if ($node == inode$ OR $node$ is a leaf node) {
   $node \rightarrow k = node \rightarrow k + 1$;
   Incrementally update node$\rightarrow m_k$ and node$\rightarrow W_k$
   according to eq. (3.8) and (3.9);
   }
   if ($k == P_1$ and $node \rightarrow fixed == 0$) {
   Compute the eigenvector associated with the dominant eigenvalue
   using power method;
   rebuild the child RPTs;
   $node \rightarrow fixed = 1$;
   $node \rightarrow M_0 = node \rightarrow m_k$;
   }
   else if ($k >= P_2$) {
   compute the eigenvector associated with largest eigenvalue;
   build its two child nodes;
   $node \rightarrow M_0 = node \rightarrow m_k$;
   }
   }

3.4.1 Analysis

In unusual cases when the child RPT's need rebuilding, the time complexity is at most $O(P_1 d)+O(\log(n))+O(P_1^2 K)$, where d, n and K are the image dimension, the number of accepted learned images and iteration times for computing dominant eigenvector respectively. The first term is the time complexity for updating W_k matrix, the second is the complexity for retrieving the learned RPT, and the third the complexity for

computing eigenvector using the power or RQCG method.

The RPT built using ILA has an important property that the accepted learned images are perfectly retrieved.

Property 1 *For each accepted learned image u, the content-based retrieving using retrieve(root, u) always locate the same image.*

Proof: In ILA, each newly accepted learned image is inserted to the first nonfixed macro node *inode* and the leaf micronode if necessary. This guarantees the exact retrieval before any new evolution of the learned RPT. As the learned RPT further evolves, at each fixed time, each accepted learned image has a fixed retrieved path determined by the tree rebuilding procedure. This always results in the same search path for each accepted image.

3.5 Experimental Results

Our mobile robot, Rome, was built on a Labmate platform from TRC. It uses a SUN SPARC-1 as the host computer. Rome was controlled to take pictures at different locations for our incremental learning. The corresponding corrected heading directions were also recorded. Some sample images for incremental learning are shown in Fig. 3.3. The tested sites are indoor hallways with various turns. The lighting condition is relatively fixed in the tested hallway sections. To take into account lighting changes between daytime and night time, all images are preprocessed to have a zero mean and a unit variance.

We use the following notation in describing different versions of ILA, which are different in the computations for the dominant eigenvector and the W matrix computation:

- *ILA0:* our earlist version of ILA [12, 139] where the power method is used and the W matrix of a child node is computed by (3.11).
- *ILA1:* improved version of ILA. The RQCG method is used and the W matrix of a child node is computed by (3.12).
- *ILA2:* further improved version of ILA1. The incremental RQCG method is used and the W matrix of a child node is computed by (3.12). An initial dominant eigenvector of W_k is computed with the nonincremental RQCG method when k first reaches $P_1/2$. The later dominant eigenvector of the same node is computed incrementally.

In our experiments, P_1 and P_2 were set to 50 and 10 respectively. These two numbers were chosen experimentally. The first P_1 accepted images were cross-validated for acceptance using linear search. The maximal iterations K_r of the RQCG method is set to 30. Fig. 3.4 shows the timing records on a SparcStation-2 for learning the first 182 accepted images. From Fig. 3.4(b) we can see that the time for incrementally learning each accepted image is typically within 0.3 second. There are certain times when the learning takes about 1 second. It happens when the number of images flowing into a nonfixed macro node reaches P_1 and the reconstructions of child RPT's are recursively conducted. The W_k matrices and their associated dominant eigenvectors for all the nodes of child RPT's require much computation; they have to be computed

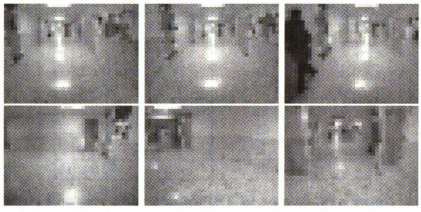
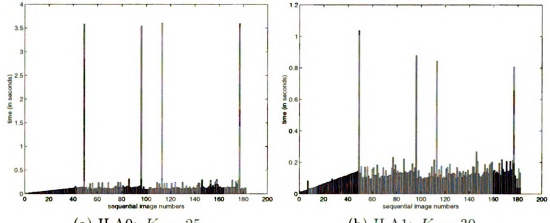


Figure 3.3: Sample training hallway images. The upper row is a straight hallway, while the lower is a left turn.



(a) ILA0: $K_p = 25$. (b) ILA1: $K_r = 30$.

Figure 3.4: The timing record on a SparcStation-2 for learning a set of 182 images (those ignored images in incremental learning are not recorded. $P_1 = 50$ and $P_2 = 10$.)

in a batch mode, which takes most of the time. The improvement of ILA1 over ILA0 is obvious.

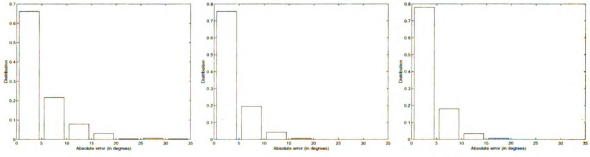
The structure of the learned tree with 180 accepted images is shown in table 3.1. The learned tree has 7 levels, including macro nodes and micro nodes.

In order to evaluate the system performance, we tested the learned tree with a set of 328 images. The error histogram in the retrieved heading directions is shown in Fig. 3.5. Since there is no guarantee to retrieve the nearest image in the accepted

Level	Macro nodes	Micro nodes	Nodes
0	1	0	1
1	2	0	2
2	4	0	4
3	2	6	8
4	0	16	16
5	0	20	20
6	0	4	4
Total	9	46	55

Table 3.1: The sizes of the learned RPT for a set of 180 accepted images.

learned images, it is likely to obtain better performance by exploring P competitive paths and picking up the best one. For the comparison, we also show the error histogram of linearly searching the best match among the set of all 180 accepted learn images. From Fig. 3.5, we can see that the use of P competitive paths does provide better results.



(a) $P = 1$ (b) $P = 4$ (c) Linear search.

Figure 3.5: Distribution of absolute angular error (in degrees) of retrieved heading directions. These three plots are based on queries using a test set of 328 images, of which 195 are from straight hallway and 133 from the two learned corners.

The tested sites are shown in Fig. 3.6, where learned hallway sections and corners are marked with thicker lines. Our robot Rome was trained to go in both clockwise and counter-clockwise directions. In real navigation, Rome rejects sudden change in heading direction by assuming a smooth steering behavior. The driving speed is slowed down when the retrieved image comes from places around a corner. With

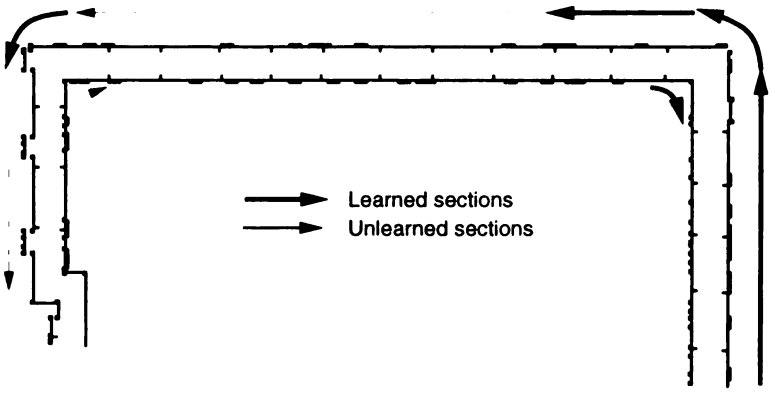


Figure 3.6: The map of the tested sites, trained sections and untrained sections are marked with thicker and thinner lines respectively.

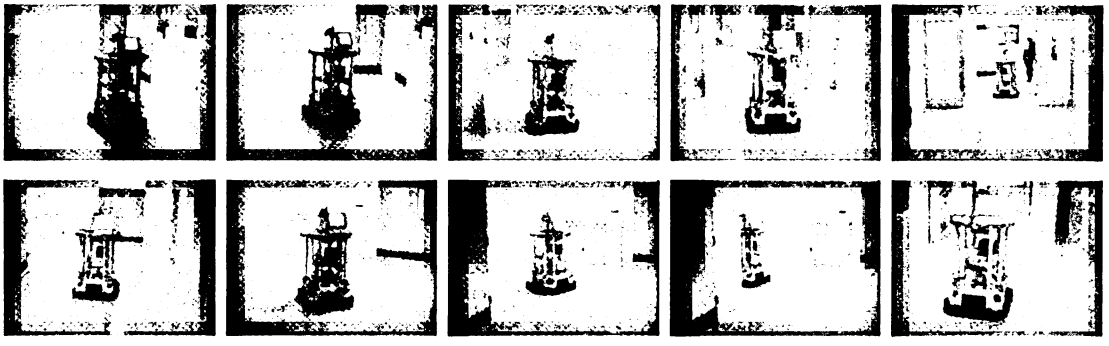
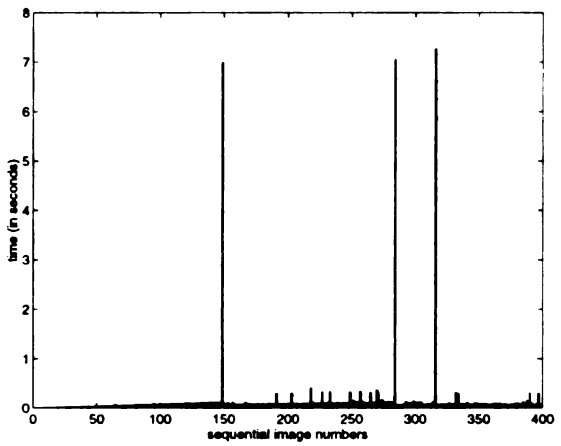


Figure 3.7: Rome navigates automatically at two turns: the first turn in the 1st row and the second turn in the 2nd row.

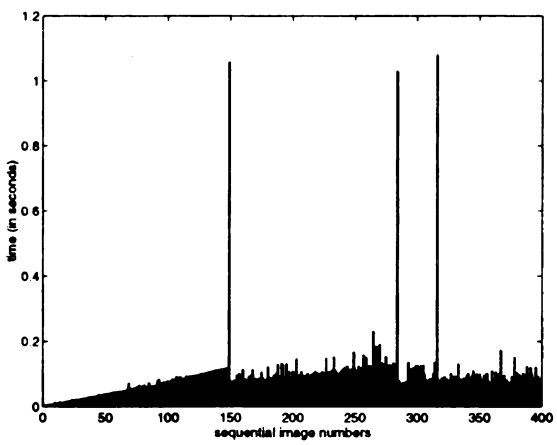
the above trained tree, Rome has consistently performed well in our numerous experiments. Fig. 3.7 shows two sample sequences of images when Rome navigates autonomously in the hallway.

To compare the response time of three versions of ILA, a set of 400 samples was used. By artificially assigning output vectors, the algorithm will always accept each new sample. The parameters were set as $P_1 = 150$, $P_2 = 30$. Fig. 3.8 shows the timing data of ILA0, ILA1 and ILA2. The maximal response time of three ILA algorithms are 7.3s, 1.1s and 0.9s, respectively. The improvement of ILA1 over ILA0 is obvious. The further improvement of ILA2 by using the incremental RQCG method is less,

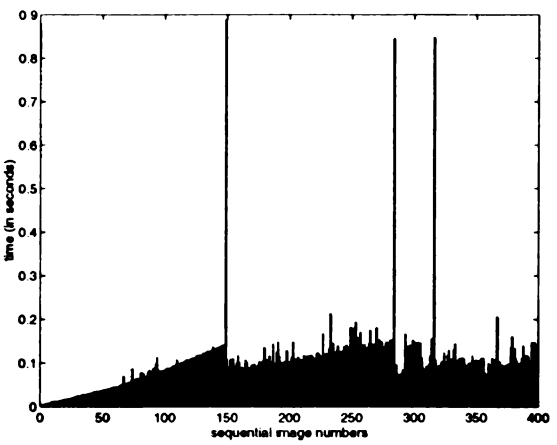
but still visible. As P_1 increases, the improvement of ILA2 over ILA1 will increase since the time complexity for computing dominant eigenvector increases quadratically. The heights at x-coordinates 150, 284 and 316 in Fig. 3.8(a) are different, since the constructed trees have different degrees of balance. Three peaks due to reconstruction when the number of samples reach P_1 are all visible in these three figures.



(a) ILA0: $K_p = 25$.



(b) ILA1: $K_r = 30$.



(c) ILA2: $K_r = 30$, $K_{ri} = 5$.

Figure 3.8: The timing records on SPARC-20/model61 for learning a set of 400 images (P_1=150, P_2=30).

3.6 Conclusions

The author has proposed improved versions, ILA1 and ILA2, over previous incremental learning algorithm ILA0, to reduce the maximal response latency. The ILA algorithms have been successfully applied to indoor mobile robot navigation. The incremental learning can be done on-line in real time. In various experiments the RQCG method converged much faster. It results in consistent, stable behavior and outperforms the power method.

The eigen-subspace method for feature derivation in this chapter is applicable to linear discriminant analysis (LDA) which has been shown to work well for indoor navigation [138]. The RQCG algorithm can be used to solve generalized eigenvalue problem $Ax = \lambda Bx$ without the computation of B^{-1} [147].

Several issues need to be pursued further. For example, the scalability of improved ILA versions for increased P_1 can be studied more carefully. The ILA algorithms make extensive use of vector processing, which can be greatly improved by the use of a vector processor. And finally, an interpolation of the outputs can be used to achieve smoother navigation behavior.

Chapter 4

State-Based SHOSLIF for Indoor Navigation

Our earlier efforts of using SHOSLIF tree for indoor navigation have experienced some difficulties when more complicated situations are involved. When the number of corners or intersections was increased, the robot might fail to make a turn. The system could get confused around a corner: when to start a turn, and when did the robot really realize that it did enter a corner. State information is useful in solving the ambiguity problem under these situations. We also notice that visual attention mechanism is useful for solving the ambiguity problem. In this chapter, states and a simple visual attention are incorporated into our earlier SHOSLIF.

4.1 Introduction

One difficulty we experienced in applying SHOSLIF-N to more complicated environment is that learning more training samples from sample drives could cause conflicts which are hard to solve using a single-framed image. This problem could be solved by using states which keep the information about the relative position between the robot and the environment. The Markov chain and its variant the hidden Markov model (HMM) or partially observable Markov decision process (POMDP) have been successfully applied to indoor navigation by several researchers [66]. Koenig and Simmons [66, 111] applied POMDP to integrate topological and metric information for navigation of Xavier. Xavier maintained a probability distribution over its current pose and successfully handled the uncertainty intrinsic in navigation, including actuator and sensor uncertainty. Two kinds of inputs were used in [111, 66]: motor reports derived from odometer on-board the robot, and sensor reports generated from circularly deployed sonar sensors.

A great gain in using state information is the capability of using visual attention to disambiguate globally similar but locally very different scenes. With the state information the robot is able to determine when to acquire local views from a global scene. We use a finite state machine to model the states in navigation. But the state transition probability is not directly estimated due to a high cost in estimating the probability distributions when high-dimensional visual inputs, rather than sonar and odometer sensory inputs, are used. The probability is retrieved by SHOSLIF and approximated by the distances to training samples. Since vision provides more

global information compared to sonar, fewer states are needed. Howvever, it should be noticed that states have little help in dealing with unanticipated events and hence some measures should be taken to deal with unexpected events. For example, in autonomous navigation, the mobile robot can be programmed to stop when it detects a novelty beyond a certain range. In the following sections, we show how state information can be naturally embedded into our SHOSLIF-N with the use of a recursive partition tree (RPT). To efficiently disambiguate states, attention windows, similar to virtual cameras in [58] are employed.

In this chapter, the author presents a systematic framework through which system states can be defined and learned online to deal with situations where a stateless system can not handle, such as where visual attention is required. The learning-based approach allows the teacher to define states online during learning, instead of preprogramming control rules into a static control scheme. Thus, the same learning scheme can potentially handle more navigation scenes, without the need for reprogramming for each different scene. The remainder of this chapter is organized as follows: In Section 4.2, we show that the problem can be formulated as an observation-driven Markov model realized by the nearest neighbor regression. The nearest neighbor regression is computed efficiently with a stochastic recursive partition tree (SRPT), as shown in Section 4.3. Vision-based indoor navigation is discussed in Section 4.4. Section 4.5 presents some experimental results. Section 4.6 compares SHOSLIF-N with two ANN-based approaches. Finally, some conclusions and future work are discussed in Section 4.7.

4.2 State and Attention: Observation-Driven Markov Model

One difficulty we experienced in applying an input-to-output mapping scheme, such as SHOSLIF [11, 139], or available neural networks, to more complicated autonomous navigation, is that learning more training samples from sample drives could cause ambiguities which are hard to be solved using a monolithic image. In Fig. 4.1, (a1) and (b1) show two images taken around a corner. These two images are similar, which implies that they have a small overall Euclidean distance. However, they require very different actions: one going straight ahead while the other turning left. But if we look at the upper-left subregions marked with white boxes, we can see that the difference is more obvious. Moreover, from the correlation images in (c) and (d), we can see that attention images are more sensitive to the translation along column direction along which depth increases. Here the heights of two blobs in (c) and (d) indicate the sensitivity of robot's translation along column direction. Therefore, we can see the use of attention window, a window that extracts a part of view, if properly chosen, can make it easier in judging different stages of an action. This is well known as landmark selection issue. However, it is not trivial to incorporate this mechanism using a systematic learning method, without writing a separate program for every different scene. In our work reported here, we use the framework of observation-driven Markov model coupled with the high-dimensionality, real-time capability of SHOSLIF.

Along straight sections of a hallway, normally global input images are sufficient

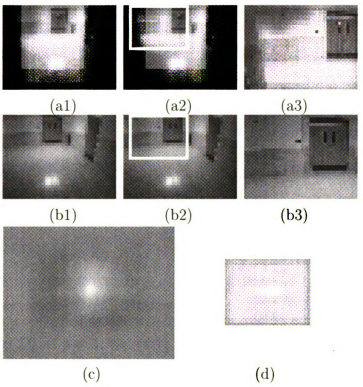


Figure 4.1: Why states and attention? (a1) and (b1) show two images around a corner. White boxes in (a2) and (b2) are attention windows of (a1) and (b1) respectively. The attention images of (a1) and (b1) are further shown in (a3) and (b3). (c) and (d) show the correlation image of (a1) with (b1) and (a3) with (b3) respectively, after zeros are padded to the periphery. (c) and (d) are resized to have the same spatial resolution.

for visual navigation. Attention windows are needed only for critical sections, e.g. corners and intersections. How does the system know that it is time to use local views and how does it act differently for global and local views? This problem could be solved by using system states which keep the history information.

Let $\mathbf{Y}_t$ be outcome random covariates at time t. Let D_t be the present and past input images and past outcomes, i.e. $\mathbf{D}_t = \{\mathbf{X}_t, \mathbf{X}_{t-1}, \cdots, \mathbf{X}_{t-p}, \mathbf{Y}_{t-1}, \mathbf{Y}_{t-2}, \cdots, \mathbf{Y}_{t-p-1}\}$.

Define the regression function as

$$m(D_t) = E(\mathbf{Y}_t | \mathbf{D}_t = D_t), \tag{4.1}$$

which is an "observation-driven" Markov model (ODMM) [19, 149]. Our goal is to estimate $\mathbf{Y}_t$ given D_t. If $p = 1$, Eq. (4.1) is an observation-driven first-order Markov model and can be rewritten as

$$m(X, Y) = E(\mathbf{Y}_t | (\mathbf{X}_t, \mathbf{Y}_{t-1}) = (X, Y)). \tag{4.2}$$

The learning set consists of triples of form $(X_t, Y_{t-1}, \tilde{Y}_t)$, $t = 1, 2, \cdots, n$, where n is the total number of learning samples. X_t, Y_{t-1} and $\tilde{Y}_t$ are the current observation, previous outcome and current outcome respectively. For efficiency under high dimensional X_t, we use the nearest neighbor (NN) estimator to approximate function m.

Let the NN approximator be denoted as $\hat{m}$, then

$$\hat{m}(X, Y) = \tilde{Y}_i$$

if (X_i, Y_{i-1}) is the nearest neighbor of (X, Y) in the learning set. Here X_i and Y_{i-1} are the input part, while $\tilde{Y}_i$ is the output part, of a training sample $(X_i, Y_{i-1}, \tilde{Y}_i)$.

The nearest neighbor estimator [18] has been widely used in function approximation: $\mathbf{y} = \hat{f}(\mathbf{x})$, i.e. the nearest neighbor estimation of $E(\mathbf{y}|\mathbf{x})$. It has also been used in approximation of Markov time series. Nonparametric estimators, including the kernel estimator [145] and the nearest neighbor estimator, have been used for prediction of Markov time series, $E[\mathbf{X}_{t+1}|\mathbf{X}_t]$. Yakowitz [144] successfully applied the nearest neighbor estimator to the prediction of rainfall/runoff time series. Under certain assumptions, the convergence properties of the nearest neighbor estimation of $E[\mathbf{X}_{t+1}|\mathbf{X}_t]$ for Markov time series have been proved (cf. Yakowitz [144] and references therein). Here we apply the nearest neighbor estimator to approximate observation-driven Markov model.

The Markov chain and its variant the hidden Markov model (HMM) or partially observable Markov decision process (POMDP) have been successfully applied to indoor navigation by several researchers. Koenig and Simmons [66] applied POMDP to integrate topological and metric information for navigation of Xavier. We notice the difference between ODMMs and popularly used hidden Markov models. Normal hidden Markov models [100] are stationary, i.e. the transition probability doesn't depend on input. Here the transition probability of an observation-driven Markov

model explicitly depends on observation at each time step. Therefore observation-driven Markov model is nonstationary.

In our vision-based navigation, three outcomes are needed: state (s), control signal (c) for mobile robot, and visual attention signal (a) used to choose the attention window. The observation is a preprocessed input image (I) at each time step. Thus, in our model, $\mathbf{X}_t = I_t$ and $\mathbf{Y}_t = s_{t-1}$. Three nearest neighbor estimators are needed for the following regressions:

$$\begin{cases} m_s(I,s) = E(\mathbf{s_t}|(\mathbf{I}_t, \mathbf{s}_{t-1}) = (I,s)) \\ m_a(I,s) = E(\mathbf{a_{t+1}}|(\mathbf{I}_t, \mathbf{s}_{t-1}) = (I,s)) \\ m_c(I,s) = E(\mathbf{c_{t+1}}|(\mathbf{I}_t, \mathbf{s}_{t-1}) = (I,s)) \end{cases} \tag{4.3}$$

Among them only the first equation is an observation-driven Markov model, since the outcome is the next state. The rest two are normal mapping functions without recurrence of variables.

The overall architecture for vision-based navigation is shown in Fig. 4.2. A single SRPT tree is used for simultaneous nearest neighbor estimation of s (state), c (control signal) and a (attention signal). Current input image I_t and previous state s_{t-1} are used to derive the next control signal c_{t+1} and next attention signal a_{t+1}, and the current state s_t. During real navigation, c_{t+1} and a_{t+1} are used to control the next motion of mobile robot and to extract the following attention windows from the video camera respectively.

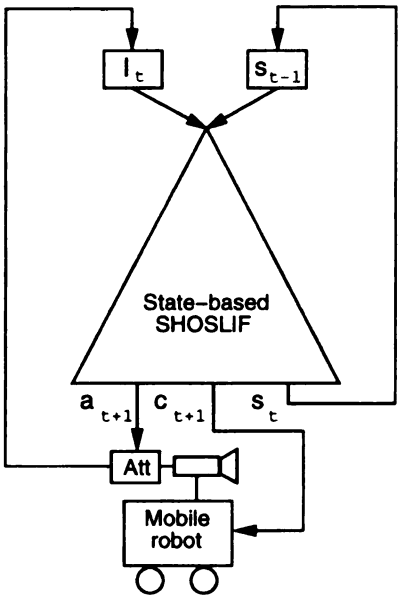


Figure 4.2: The architecture of state-based SHOSLIF for vision guided navigation. "Att" stands for "Attention control".

4.3 Stochastic Recursive Partition Tree for Nearest Neighbor Regression

A challenge here is that I_t has a very high dimensionality, typically at least a few thousands. We need to briefly describe how SHOSLIF [139] addresses this challenging problem.

4.3.1 Navigation as a content-based retrieval problem

A navigator can be regarded as a complicated function which maps a high-dimensional input (image and state) into the corresponding low-dimensional output (control signal, action, the new state, etc). In the learning phase, a set of training images is used to build a recursive partition tree (RPT), in which each learned sample records a desired input-output pair. At each inner node of the RPT, an eigen-subspace method is

used to extract characteristic features from high dimensional input space for further partition of input space. This process is done recursively down the tree until the current node contains only samples with a similar output signal. In the performance phase, each newly grabbed input is used to retrieve the best-matched input from the trained RPT, and the output signal associated with the retrieved input is used as desired output.

We chose binary partition for its simplicity and speed. At each inner node, the further space partition is based on the projection of inputs onto the principle component vector of PCA of the sample inputs assigned to the node, which captures the largest variance in these inputs. Therefore, at each inner node, we need only to obtain the eigenvector associated with the largest eigenvalue. This leads to our efficient on-line incremental learning algorithm [139].

The above learning process was done incrementally. First, the data covariance matrix C was estimated from n input samples $x_1, x_2, \cdots, x_n$ as $\hat{C} = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^t$. Then the dominant eigenvector of the estimated $\hat{C}$ was computed by either the Power method or the Raylaigh quotient conjugate gradient (RQCG) method.

However, the dominant eigenvectors can be estimated without estimating the covariance matrix C as a prerequisite. Several authors addressed this problem using stochastic approximation. Stochastic approximation has been shown especially useful when high accuracy is not required while only a few eigenvectors need to be estimated. Here we apply Oja and Karhunen's method [92] for stochastic approximation of the dominant eigenvector. For stochastic matrices $A_1, A_2, \cdots$, with finite constant mean $A = E\{A_k\}$, Oja and Karhunen's iteration can be represented as follows:

$$\tilde{u_k} = u_{k-1} + \mu_k A_k u_{k-1}, \tag{4.4}$$

$$u_k = \tilde{u_k}/\|\tilde{u_k}\|, \tag{4.5}$$

where μ_k is the learning rate and $\|\cdot\|$ denotes the l_2-norm of a vector. In our computation for the dominant eigenvector of covariance matrix A, A_k is set as the outer product of sample x_k: $A_k = x_k x_k^t$ according to Oja and Karhunen [92].

The convergence of the above stochastic approximation has been proved [92].

Theorem 1 *(Oja and Karhunen, 1985) Under the following assumptions:*

1. *Each A_k is almost surely bounded and symmetric and the A_k matrices are mutually statistically independent with $E\{A_k\} = A$ for all k.*

2. *The largest eigenvalue of A has unit multiplicity.*

3. $\mu_k \geq 0$, $\sum \mu_k^2 < \infty$, $\sum \mu_k = \infty$.

4. *Each A_k has a probability density bounded away from zero uniformly in k in some neighborhood of A in $R^{n \times n}$.*

Then, u_k tends to the eigenvector associated with the largest eigenvalue almost surely as $k \to \infty$.

For each inner node, the incremental learning of a new sample can be conducted recursively. When learning a new sample, the mean and dominant eigenvector are updated incrementally. For the corresponding child nodes, only the changes caused

by this update need to be processed. Let old_list and new_list be the old and new sample lists going into a child node respectively, m_k and H_k be mean vector and dominant eigenvector of the child node. Then the processing for each child node can be done recursively as shown below:

1. Accept new_list from parent node.
2. Form the d_list and i_list:

   d_list=old_list−new_list;

   i_list=new_list−old_list;
3. Delete samples in d_list, update m_k and H_k.
4. Insert samples in i_list, update m_k and H_k.
5. Obtain new_list for child nodes.

4.4 A Case of Indoor Navigation

Compared with outdoor road following, vision-based indoor navigation faces similar challenges because there is no stable features (e.g. floor edges) and no stable contrast pattern along typical navigation paths.

For our indoor navigation, the corridor types are shown in Fig. 4.3. In this figure, we use the following brief notation to indicate corridor types: "L" for left turn; "R" for right turn; "LZ" for left Z junction; "RZ" for right Z junction; "X" for four-way intersection; and "S" for straight corridor.

For each corner or intersection, the situations are further grouped into states: approaching, entering, and exiting corner or intersection. To reduce the number of

Figure 4.3: Different types of corridor structure in our indoor navigation.

states, some states can be merged. For example, an indoor corridor section with types "L", "LZ", "R" and "RZ" can be represented by a six-state model, as shown in Fig. 4.4. Here approaching corners or intersections is represented as a single state, ambiguity state ("A"). After transition to "A" state, the next transition can loop to itself or jump to "L", "LZ", "R" or "RZ", depending on current visual image I_t and previous state s_{t-1}. The transition from each state depends on the current observation and the previous state estimated by the NN estimator. Visual attention for local view is needed only at state "A" to disambiguate the possible next states. In our case the raw input image is of size (60 × 80) pixels: it is averaged with 2 × 2 window and reduced to half size (30 × 40) to get the global view; the left and right local attention windows are subregions of the raw input image centered at (15, 22) and (15, 55) respectively but with the same size (30 × 40). For this arrangement, the attention signal needs only a binary flag to specify whether global view or local view will be used.

$$a_{t+1} = \begin{cases} 1 & \text{if } s_t = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

Koenig and Simmons [66] use sonar and odometer sensory inputs as observation in their HMM model. They define states on the spatial grid of the hallway paths. Usually a large number of states are needed. If similar HMM is directly applied to our

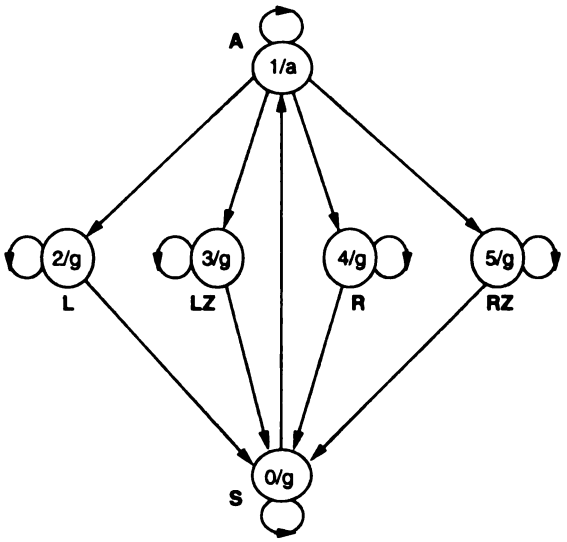


Figure 4.4: Observation-driven Markov model for indoor vision-based navigation, where x/y is used to represent a state x with associated action y. 'a' or 'g' are used to indicate either two local attention views or a single global view is used. Associated with each arc is the current observation, either a local view or a global view.

vision-based navigation, the complexity in training HMM will be very high, since here the input dimensionality with image is much higher. Our vision-based navigation task is much more challenging than sonar-based navigation, since the visual images does not relate to control as directly as range data. On the other hand, with the use of visual images the number of states can be greatly reduced, since visual input provides richer information and the system does not need to remember odometer information.

For the ODMM model shown in Fig. 4.4, its corresponding representation in our case is shown in Fig. 4.5. Each image I_t is preprocessed to have zero mean and a unit l_2-norm. S_0 to S_5 are binary fields representing state 0 to 5 respectively; s_t is the predicted current state; a_{t+1} is the predicted attention selection, which is a binary flag indicating whether global view or attention window will be used; o_{t+1} is the associated output control signal for current (I_t, s_{t-1}) pair. The fields marked with solid boxes are concatenated to form a single long 1D vector as input to SRPT. Those fields in

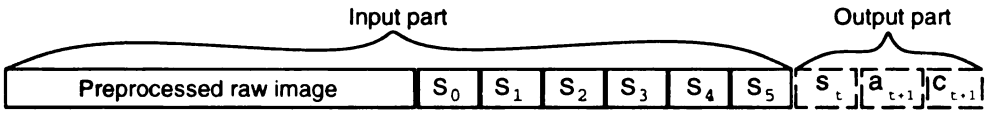


Figure 4.5: The representation of input and output parts.

dashed boxes are either predicted or associative quantities.

We need to collect quadruples of (s_{t-1}, I_t, s_t, c_t). The control signal of mobile robot is the corrected heading direction. We need to train all the state transitions in our observation-driven Markov model.

4.5 Experimental Results

Our mobile robot Rome (RObotic Mobile Experiment), built on a TRC labmate, was used to test our algorithms. In our experiments, we trained the robot to navigate along the loop shown in Fig. 4.6, which is on the third floor of MSU Engineering Building. The floor plane covers an area of approximately 136 × 116 square meters. We have five types of corridor structure shown in Fig. 4.3: "L", "R", "LZ", "RZ", 'T' and "S". "X" type in Fig. 4.3 was ignored without affecting the performance. We tested our algorithm on the third floor of our Engineering Building. The loop includes a "L", a "LZ", a "RZ" and three 'T' junctions. Some sample training images around these corners and junctions are shown in Fig. 4.7. These six corners or junctions are very typical for indoor navigation. Their appearances and structures are quite different, and the widths of straight corridor segments are different too. Navigation using only vision in this environment is challenging for any method that uses predefined features such as floor edges.

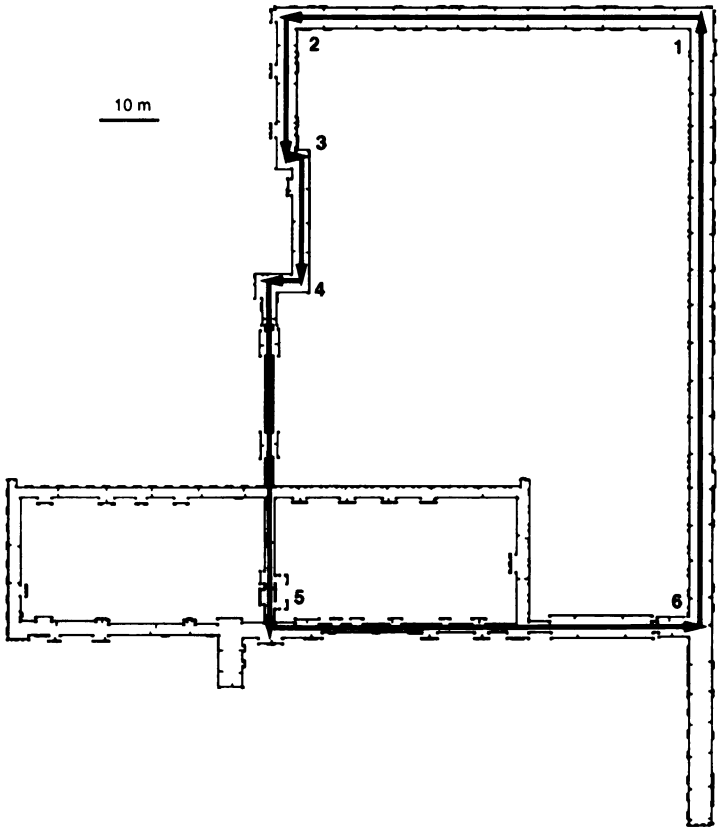


Figure 4.6: A map of the test site. The test loop is indicated by thick solid lines. Six different corners or intersections are trained in our tests. They are marked with bold-faced arabic numbers.

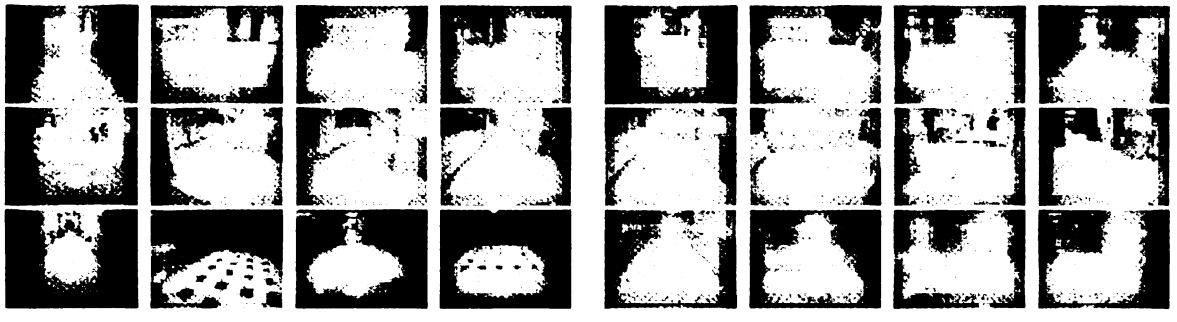


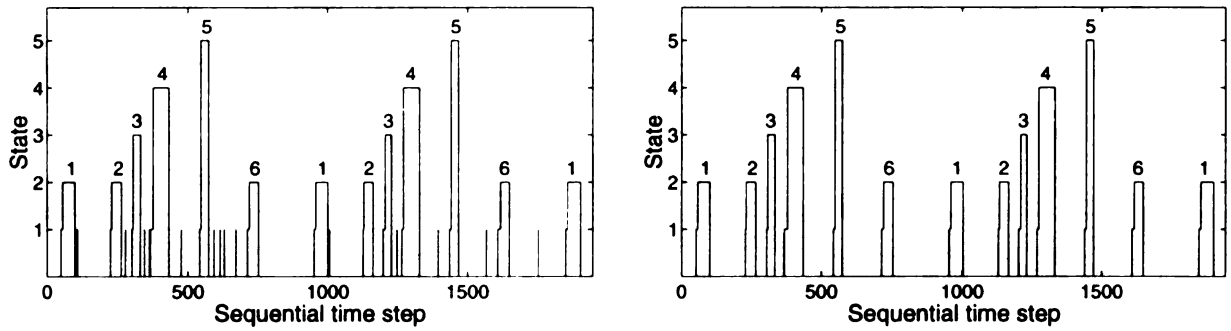
Figure 4.7: Some sample training images around corners or intersections. Six sets of four consecutive images are from six different corners or intersections in the test loop.

The training samples were collected under human control from sample drives. At each site, an input image, previous state, current state and its associated output control signal were obtained. The input image was normalized to have a zero mean and a unit l_2-norm to suppress the effect of absolute lighting brightness and contrast variation to some degree. Then this preprocessed raw input image and the previous state information were used to query the SRPT. If the retrieved sample had the same current state and its associated heading was within tolerance, the new sample was discarded without being learned; Otherwise, incremental learning was conducted for this training sample.

First we collected a reasonable set of training samples around each section of the loop. Then we tested our algorithm, letting the system continue to learn at each location. When all the presented samples are rejected, Rome can be set free.

Rome normally roams at a speed of 40 cm/s. The robot slows down when the correction in heading direction is larger than 10° or the robot enters a non-straight state.

In order to remove the spurious state transitions without resorting to higher order Markov models, a voting scheme was used. We kept the state history up to five steps. The histogram of history states was computed. The state transition was confirmed only when the number of votes was more than two; otherwise the current state was unchanged. Fig. 4.8 shows sample state transitions of more than two passes along the tested loop. Around each corner or intersection, a transition to state "A" always precedes the confirmed corner or intersection state. From Fig. 4.8 we can see that the smoothing scheme did help in achieving a stable navigation behavior.

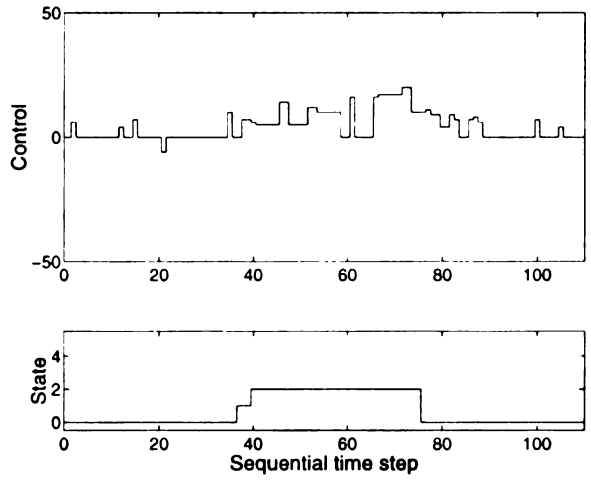


(a) Raw output states. (b) Output states after smoothing.

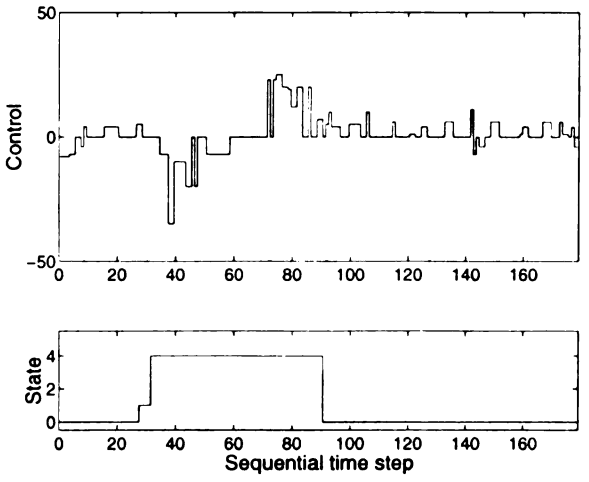
Figure 4.8: Sample state transitions covers more than two passes of continuous running along the tested loop with six corners or intersections, which are labeled with digits in this plot.

Some sample control signals with associated states are showed in Fig. 4.9. The robot turned left in Figs. 4.9(a) and (c). In Fig. 4.9(b), the robot turned right then left around corner 4, which is a "RZ". From Fig. 4.9(d), it can be observed that along a straight section, the robot turned with much smaller magnitude which resulted smooth navigation behavior.

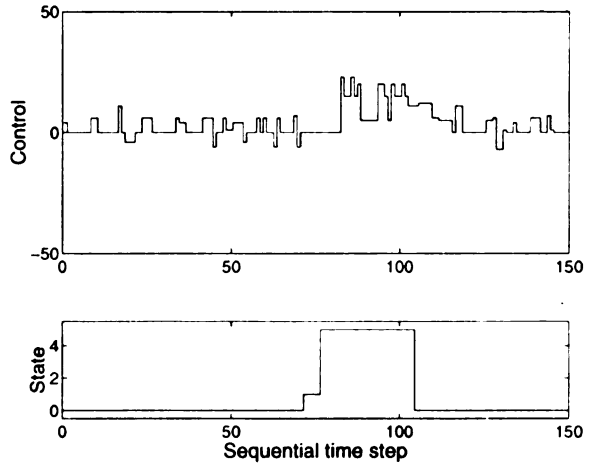
Without the use of states, stateless SHOSLIF experienced difficulties in guiding the robot to navigate through the tested loop. It usually failed to make the 4th corner, which is a "RZ". Fig. 4.10 shows the scenario with plots collected from sample navigation trajectories. Fig. 4.10(a) and (b) show two failure cases when states were not used. Fig. 4.10(a) shows a case when the robot retrieved images with left turn around the critical turning points, since the visual appearances around these positions are similar to some other images around a left turn, e.g. the 6th corner. The robot turned left and then failed to make the turn. Fig. 4.10(b) shows another case where the robot underturned right and failed to make the 4th corner. Fig. 4.10(c) shows how the robot performed after the state information was incorporated: the robot



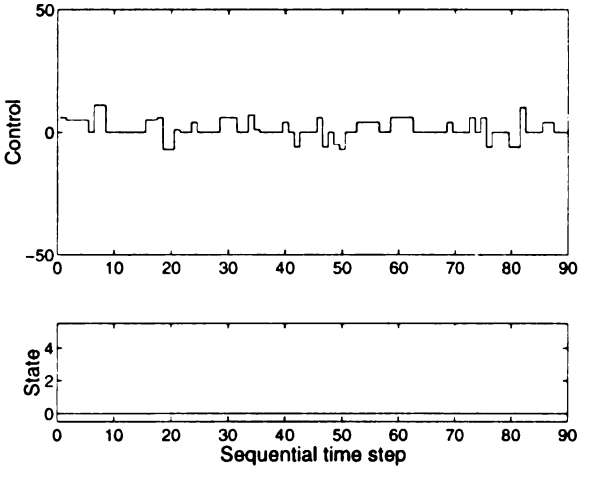
(a) Around corner 2 ("L")



(b) Around corner 4 ("RZ")



(c) Around corner 6 ("L")



(d) Between corners 6 and 1 ("S")

Figure 4.9: Some sample control signals and the associated states. The control signals plotted in this set of figures are corrected heading directions in degrees. (a) and (c) are cases of left turn. (b) is a "RZ", which has a right turn followed by a left turn. (c) is a straight section between corners 6 and 1. We can see that the change in corrected heading along straight section is smaller.

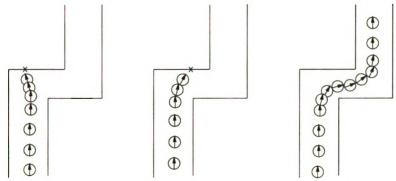


Figure 4.10: Sample trajectories around the 4th corner show how states can help the robot in navigation. (a) and (b) show two failure cases when states were not used. "X" in (a) or (b) indicates an upcoming collision with the wall. (c) shows the robot successfully made the turn when states were used.

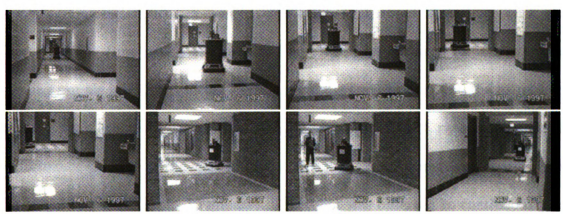


Figure 4.11: Rome navigates autonomously around corner 5. The first five images are video sequences taken from behind the robot. The last three images are taken from front of the robot.

successfully made the turn with state-based SHOSLIF. The states were very helpful in disambiguating critical scenarios.

Fig. 4.11 shows a sample run around the fifth corner. Fig. 4.12 shows the mobile robot's view sequence during real navigation around corner 5. Each image was grabbed at every single time step.

The content-based retrieval of each (I_t, s_{t-1}), a pair of current image and previous state, was performed at a frequency of 6 HZ on the onboard Sun Sparc-1 of the

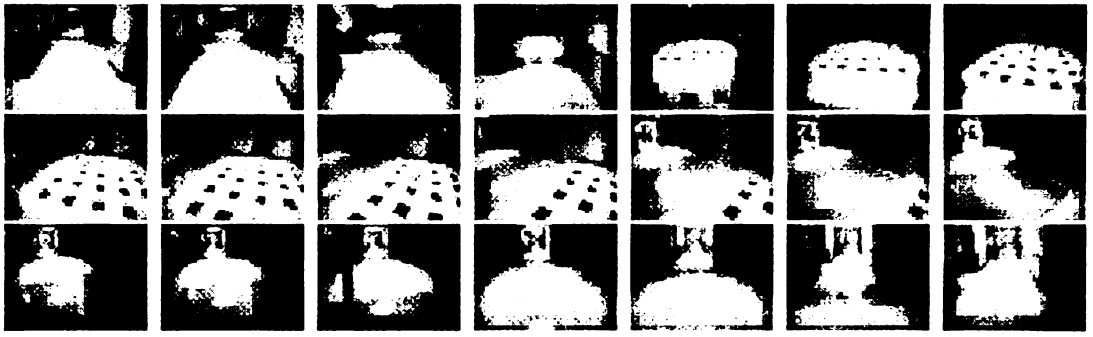


Figure 4.12: The mobile robot's view sequence around corner 5 during a real navigation. Each image was taken at every time step. This sequence shows the robot approached the fifth corner, successfully made the turn and entered straight section.

robot. The incremental learning was conducted using the onboard SPARC-1. In a batch training experiment, a set of 272 samples took about 26 seconds on a SUN Sparc-10. The time record is shown in Fig. 4.13. The maximal response time per learning sample is within 1 second. For most of the training samples, the incremental learning took less than 0.3 second. We have conducted test runs to observe the performance stability of the learned robot. One pass of autonomous navigation along the tested loop took about 20 minutes. The robot was observed to continuously run for longer than 5 hours several times before the onboard batteries which provided the power were low. In dozens of such tests conducted, so far the robot all performed flawlessly.

4.6 Comparison with Two ANN-based Approaches

To show how SHOSLIF performs compared with other alternative methods, we compared it with feedforward neural networks and radial basis function networks. We

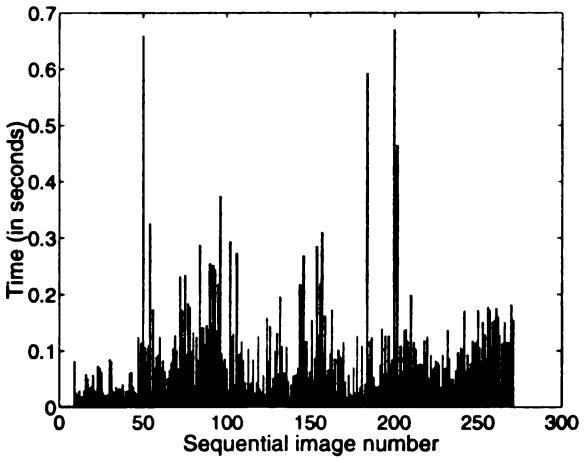


Figure 4.13: Time record of incremental learning a set 272 training images. The learning time was recorded on a SPARC-10.

used two separate sets of data for comparison. Both sets of data were collected on two separate time on the 3rd floor of our Engineering Building. Training with either set of training sample results a tree which leads to successful performance when tested in the trained loop. Set 1 and Set 2 contain about 500 and 300 samples of (I_t, s_{t-1}, s_t, c_t) respectively. Set 1 is more redundant and covers more scenarios.

To map (I_t, s_{t-1}) into current heading c_t, we used the same output scheme as ALVINN and ROBIN: the output pattern of each training sample is a Gaussian distribution peaked at the desired corrected heading. In our simulation, the output layer has 21 to 31 nodes with a resolution of two degrees apart. A sample training input–output pair is shown in Fig. 4.14. After the neural network is trained, the simulated output heading is taken as the peak of a Gaussian fit to the outputs of neurons at output layer.

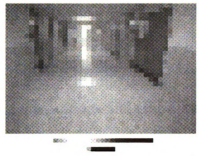


Figure 4.14: Sample training input–output pair: the top part is a 30 × 40 image. The second row is the associated output signal, which is a Gaussian distribution peaked at the corrected heading direction. The last row is the binary state fields, which has a one in first fields and zero in other fields.

4.6.1 Simulation of a feedforward neural network

For simulation of a two-layer feedforward neural network, we used "trainbpx" with adaptive learning rate available in the MATLAB neural network toolbox. Similar to ALVINN, we used four to nine hidden nodes in our simulation. Our simulation showed that it was very unlikely for multilayer perceptron (MLP) to converge to reasonable weights if starting from initial random weights.

Three sets of data in Section 2.7 are used to study the behavior of a two-layer perceptron.

Set A Real images from straight hallway sections: a set of 100 (30 × 40)-pixel images with five different corrected heading directions: $-10°$, $-5°$, $0°$, $5°$ and $10°$. Each of these five classes has 20 samples.

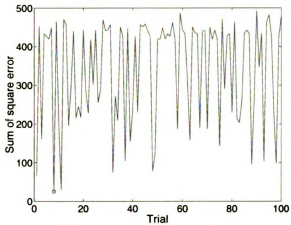
Set B A full set of 318 (30 × 40)-pixel real images are used. This set of images are used in our training of SHOSLIF-N for vision-based navigation. It includes 210 images from straight hallway sections and 108 images from corner sections. Images were collected from corners 1 and 2 and three straight sections connected

to these two corners.

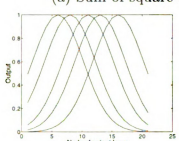
Set C Synthesized data set: Given a trajectory map, synthesized sample images are generated with different orientations and translations. For a square trajectory, a typical set consists of 100 (30×40)-pixel synthetic images with five different headings: $-10°$, $-5°$, $0°$, $5°$ and $10°$.

When the size of problem was small, i.e. the number of images was small and the size of image was also small, the training of feedforward network went successfully: it found a reasonable solution within a few trials starting from initial random guesses. But we found that when the input dimension was high, e.g. 30×40, the neural network would not converge to reasonable solutions with random initial weights. We ran our MATLAB script files for 100 times, each time with a different random guess of initial weights. The training epoch was set to 10,000, large enough to converge to a local minimum. Fig. 4.15 shows the training record for Set A. Here we use four hidden nodes as indicated in [95]. Fig. 4.15 shows that the weights do sometimes converge to a good solution for the training set, although only a few trials provide solutions with small sum-of-squares (SSE) errors as shown in Fig. 4.15(a). Our simulation with Sets B and C got consistent results: the learning process can easily get stuck to poor local minimums.

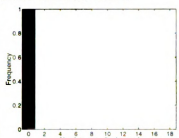
Therefore we used principle component regression (PCR) [35] to provide the initial weights: the weight of first layer was initialized with the principal components of the input patterns, while the second layer was initialized with linear regression between desired outputs and the response of hidden layer. Initialization MLP using PCR led



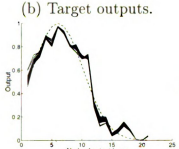
(a) Sum of square error for 100 trials.



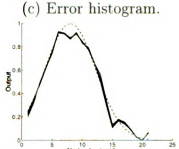
(b) Target outputs.



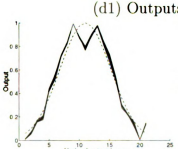
(c) Error histogram.



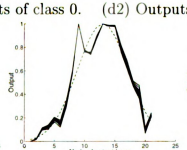
(d1) Outputs of class 0.



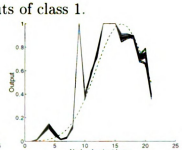
(d2) Outputs of class 1.



(d3) Outputs of class 2.



(d4) Outputs of class 3.



(d5) Outputs of class 4.

Figure 4.15: Training with Set A. 100 trials with different random guesses of initial weights. (a) shows the record of sum of square error (SSE). The trial with minimal SSE is marked with a circle. (b) shows the target output for five classes of training samples with different corrected headings: $-10°$, $-5°$, $0°$, $5°$ and $10°$. (c) is the error histogram of retrieved heading directions. (d1) to (d5) show the network output overlaid on target output for each class: simulated outputs and target outputs are plotted in solid lines and dotted lines respectively.

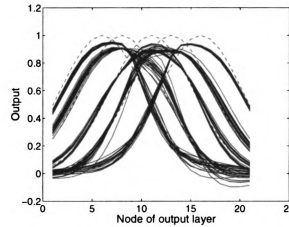
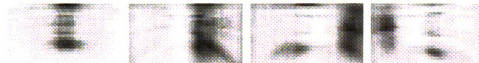


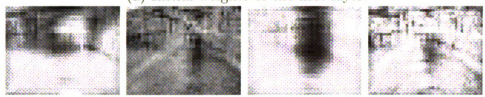
Figure 4.16: Training with Set A using PCR for initialization of weight. The network outputs 100 training images are plotted with solid lines, while the desired target outputs are plotted in dashed lines for comparison.

to much better convergence in our experiments. One example is shown in Fig. 4.16. Set A was trained with 1000 epoches. The learning process converged quickly and resulted in very small sum-of-square error. We also notice that the network outputs initialized with PCR are much smoother and the network output corresponding to each class forms a compact class.

For comparison, the weight images of a two-layer perceptron are shown in Fig. 4.17. The set of 318 images used in chapter 2 was used for training. Four hidden nodes and twenty-one output nodes are used. The initial weights of the first layer using PCR are the first four principle components of input patterns, shown in Fig. 4.17(a). The final weights of the first layer after back-propagation learning are shown in Fig. 4.17(b). For comparison, the first four eigenvectors from linear discriminant analysis are shown in Fig. 4.17(c). It is clear that learning with back-propagation can get similar results as linear discriminant analysis, if the two-layer feedforward network is properly initialized.



(a) Initial weights of the first layer.



(b) Final weights of the first layer after back-propagation learning.

(c) First five eigenvectors of linear discriminant analysis

Figure 4.17: Comparison of a two-layer perceptron with principle component analysis and linear discriminant analysis. The two-layer perceptron is initialized with principle component regression approach described in the text.

4.6.2 Simulation of a radial basis function network

The implementation of ROBIN is different from a typical RBF network in the following aspects: (1) Normalization. The responses of the receptive fields are normalized to have unit sum response for all neurons in the first layer. This normalization improves the interpolation capability, as shown by references sited in [106]. (2) Center selection. Rosenblum and Davis [106] used K-means clustering algorithm for center selection but the results were not satisfactory. So they used "forced-clustering" by manually assigning subset of input patterns as centers. The author did the same way for comparison. Here orthogonal least squares (OLS) [10] is also used to do the automatic center selection for better performance.

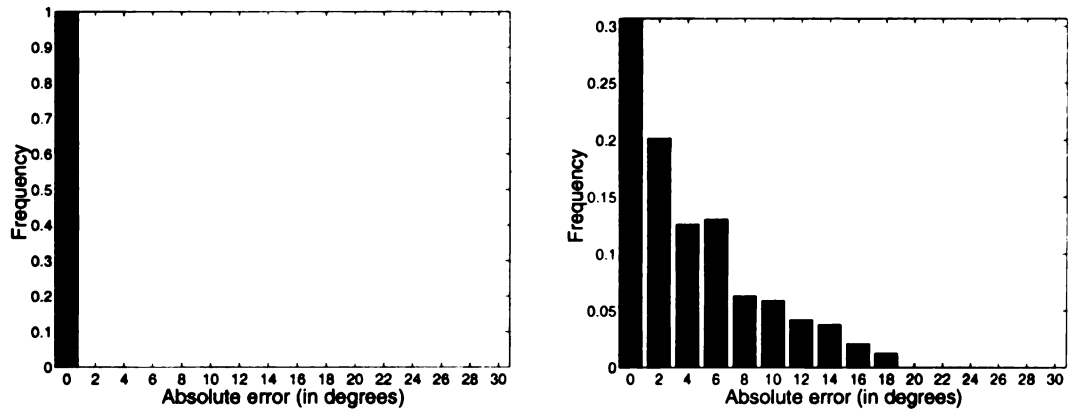
4.6.3 Comparison with two ANN-based approaches

We used Set 1 to for training and Set 2 for testing. The error histograms are plotted in Fig. 4.18, for all three different approaches. SHOSLIF always gets perfect retrieval for the training set, so the error histogram concentrates on a single bin with zero error.

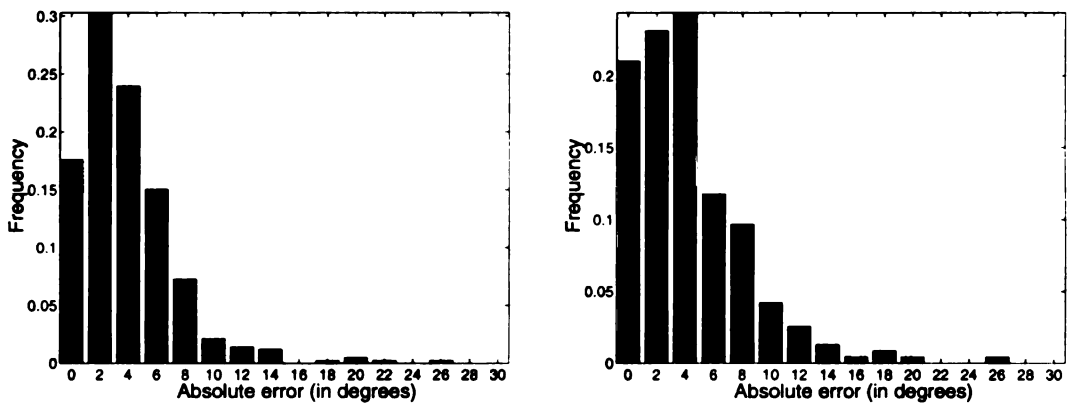
Since we need to study both the accuracy of retrieved heading and that of state-prediction, the accuracy of state-prediction is studied for each of the three approaches. Here the current image and previous state (I_t, s_{t-1}) are mapped into current state s_t. This mapping is classification, rather than approximation which maps (I_t, s_{t-1}) into current heading c_t. Therefore networks for classification, instead of approximation, should be used. A MLP with a linear output layer [132] is known to perform discriminant analysis or classification. We used a three-layered perceptron, with two hidden layers, for state prediction. The number of output nodes is the same as the number of states, which are coded as binary patterns. For network output, the node with the highest response corresponds to the predicted state. The number of nodes in first hidden layer ranges from 4 to 15, while the number of nodes in second hidden layer is smaller, with a range from 4 to 10.

RBF can also be used for classification. Similar to MLP, the states are coded as binary patterns and the number of output nodes is the same as the number of states. The output node with the highest response gives the predicted state. The centers are again selected with orthogonal least squares [10].

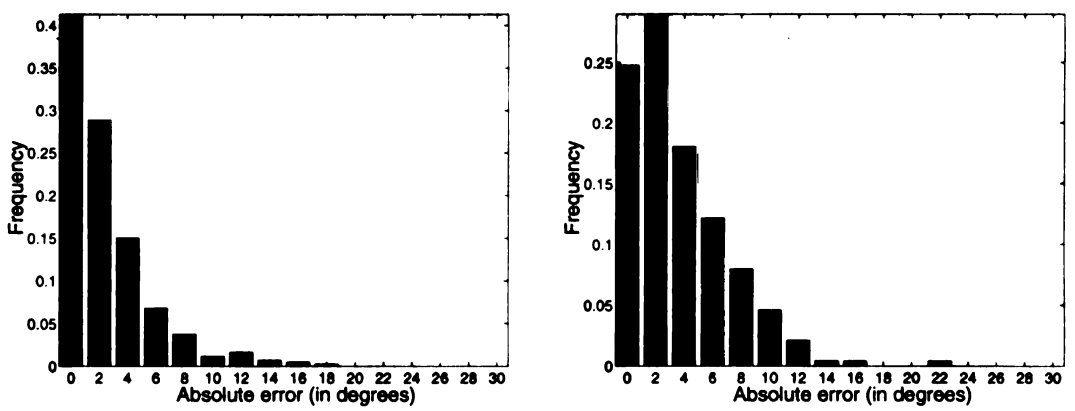
We used one set of data for training and the other for testing, then reverse the



(a) State-based SHOSLIF.



(b) A feedforward network.



(c) A RBF network.

Figure 4.18: Comparison of three approaches: SHOSLIF, the best feedforward network and the best RBF network we obtained. The error histograms for the training set are shown in the left column while the error histograms for the test set are shown in the right column.

order of the training set and the test set for cross-validation. The accuracy of state prediction is reported in Table 4.1. For MLP, it has a tendency to overtrain with the training set, which usually results in poor generalization performance for the test set. Therefore, the number of epochs is determined by a simple cross-validation process: check the state prediction accuracy of both the training set and the test set. The number of epochs which leads to balanced performance for both the train and test sets was chosen. The images of Set 2 are shown in appendix B. Since Set 1 is more redundant than Set 2, training with Set 1 leads to better performance when tested with the disjoint set. From Table 4.1 it is clear to see that state-based SHOSLIF performs better than both MLP and RBFN. A conclusion can be drawn: state-based SHOSLIF is better suited for the tasks here.

Train	Test	MLP	RBFN	SHOSLIF
Set1	Set1	80.71%	90.64%	100.00%
Set1	Set2	73.16%	80.88%	89.08%
Set2	Set1	66.10%	79.59%	87.09%
Set2	Set2	85.66%	84.56%	100.00%

Table 4.1: Comparison of three approaches in state prediction.

The response speeds of these three approaches are reported in Table 4.2. After training with each learning mechanism with Set 2, the response time in milliseconds was recorded on a SUN SPARC-10. The recorded time is the CPU time spending on mapping into to output, not including time spent on grabbing images. When using ANN-based approaches, the learned parameters were loaded from MATLAB, but the response time was obtained using C programs for network mapping.

A qualitative comparison among MLP, RBFN and SHOSLIF is summarized in

	MLP	RBFN	SHOSLIF
Response time	5.9 ms	84.0 ms	28.3 ms

Table 4.2: Comparison of three approaches in response time.

Table 4.3. All three approaches are suited for hardware implementation. The major power of SHOSLIF comes from its local adaptive nonparametric regression, which is more flexible than global parametric regression. Recursive partitioning provides a good tradeoff between speed and performance. Projection pursuit regression [52] is another well known adaptive algorithm for function approximation, but with very high computational complexity. Friedman [37] provides an excellent discussion on this subject. Both MLP and RBFN use global parametric regression and try to minimize the global least squares error. When a learned MLP is exposed to more scenarios, the network could perform deteriorately for the learned samples. This is termed as "memory loss" problem. RBFN for navigation has been shown to experience fewer memory loss problems [106]. With SHOSLIF, memory loss problem is further alleviated due to its use of local nonparametric regression.

	Local global	Para.	Hierarch.	Time compl.	Space compl.	Corr. conv.	Increm. learning
MLP	global	yes	non/few	$O(d \times h)$	$O(d \times h)$	trials PCR	yes mem. loss
RBFN	global	yes	non/few	$O(d \times c)$	$O(d \times c)$	OLS	hard
SHOSLIF-N	local	no	full	$O(d \times \log n)$	$O(d \times n)$	yes	feasible

Table 4.3: Qualitative comparison of three different approaches. The following notations are used: "local/global" for local regression/global regression; d for input dimension; h for the number of hidden nodes in a MLP; c for the number of centers in a RBFN; n for the number of learned samples. The following abbreviations are also used: "Para." for parametric regression; "compl." for complexity; "Hierach." for hierarchical; "Corr. conv." for correct convergence; "Increm." for incremental; "mem." for memory.

With MLP or RBFN, our experiments showed that there was few problems in

training the network to make a single turn. But when the network was exposed to several corners or intersections, things were different. Both MLP and RBFN had very hard time in navigating along the tested loop, especially around corners 3 and 4 in Fig. 4.6. ALVINN has experienced similar problems when exposed to various road conditions. Pomerleau [95] used a rule-based method to arbitrate individual networks trained for specific roads. The integration or arbitration of several networks is a difficult issue. Here we used state information to solve the problem in our navigation. No explicit rule for network arbitration was involved.

According to our experience in simulation, SHOSLIF is advantageous in at least two aspects: (1) ease of training; (2) good performance after it is trained. Although the feedforward network could give reasonable solution, many random trials are needed for initial weights if random initial weights are used. The RBF network does provide comparable performance. When most of the training samples are used as centers, the RBF network will act like a nearest neighbor estimator. But then the computation complexity will be prohibitive when the number of training samples is large. On the other hand, our SHOSLIF-N has to pay extra cost in storing training samples. It seems that the payoff from extra storage enables state-based SHOSLIF to reach a good performance accuracy with a real-time speed. The training is also faster.

4.7 Conclusions and Future Work

The characteristics of the proposed algorithm can be summarized as follows: (1) An observation-driven Markov model for vision-based navigation. The state information and visual attention have been incorporated into the framework to deal with more complex scenes. (2) The ODMM is realized by a nonparametric approach—the nearest neighbor regression. A stochastic recursive partition tree is used for efficient nearest neighbor estimation for the high-dimensional visual data in real-time. The overall learning algorithm is a local nonparametric adaptive regression, which exhibits more flexibility than global parametric regression used in both MLP and RBFN. (3) The appearance-based method. Compared with other navigation approaches using sonar or odometer sensors, the use of visual image provides richer information and hence the number of states is greatly reduced.

Some future researches will be conducted: (1) More extensive tests will be conducted, especially under different navigation environments. (2) The use of direction commands which allows to make different turns at the same intersection. This can be done using our current framework by allowing states to include information about the desired turning direction. The feasibility of the framework can be further utilized in these future studies.

Chapter 5

Conclusions

5.1 Summary

This dissertation presents a learning-based visual navigation scheme. The navigator is treated as a content-based retrieval system which directly maps a preprocessed input image into an output steering signal. In contrast with global parametric regression used by current ANN-based approaches, nonparametric recursive partitioning regression is used in this dissertation. Nonparametric recursive partitioning regression is more flexible than global parametric regression. Projection pursuit regression can also be used to approximate the input-to-output mapping but has a very high computational complexity. Nonparametric recursive partitioning regression [37] used in this dissertation is a good tradeoff between speed and performance.

The nonparametric recursive partitioning regression is realized by a recursive partition tree (RPT). In the training phase, principle component analysis or linear discriminant analysis is used at each inner node to automatically derive the best feature

for further partition. In the autonomous navigation mode, each newly grabbed image is used to do the content-based retrieval. The steering control signal associated with the best matching learned sample is used as the next control signal. The system has been successfully tested in indoor navigation.

Since only the most dominant eigenvector of PCA or LDA is needed in binary partition, an on-line incremental learning scheme is developed with very low computational complexity. This online incremental learning algorithm greatly facilitates the training, compared with previous batch-training mode.

State information is quite useful in navigation. In chapter 4 state information is incorporated into the system, termed as "state-based SHOSLIF-N." State information and a simple visual attention mechanism are naturally incorporated into the system. The system is modeled as an observation-driven Markov model (ODMM), which is again realized by a recursive partitioning regression through an RPT. Using a set of fewer than 300 learning samples, the state-based SHOSLIF-N has been successfully tested on the second and third floors of our Engineering Building.

5.2 Contributions

The major contribution of this dissertation can be summarized as follows:

- Nonparametric recursive partitioning regression which directly maps preprocessed input images into output steering signals for vision-based navigation.

  Recursive partitioning regression of SHOSLIF-N is different from global parametric regression used in either ALVINN or ROBIN. Recursive partitioning

regression uses local regression splines and exhibits more flexibility.

- An online incremental learning algorithm with low computational complexity.

  In SHOSLIF-N, only the most dominant eigenvector of principle component analysis or linear discriminant analysis is computed at each inner node of RPT. This fact is used in Chapter 3 to develop an efficient incremental learning algorithm which greatly eases the training for vision-based navigation.

- State-based SHOSLIF-N incorporates state information and a simple visual attention mechanism.

  State-based SHOSLIF-N is modeled as an observation-driven Markov model (ODMM) which is implemented via a recursive partition tree. State information helps the system in disambiguating scenarios with similar visual appearances but different steering signals in autonomous navigation. State information and visual attention provide the possibility for the system to work in more complicated situations. They are naturally incorporated into the system. State-based SHOSLIF-N does not need an extra arbitration layer to determine which feature tracking algorithm or which navigation network suited for a trained path should be used, as in YARF [119] or MANIAC [56].

5.3 Future Work

The research work done in this dissertation shows that direct input-to-output mapping using nonparametric recursive partitioning regression is successful in indoor naviga-

tion. However, some issues remain to be explored:

- Obstacle avoidance.

  In this dissertation, the author concentrates on path-following for indoor navigation. One natural question is how to extend the framework to make the robot avoid obstacles during navigation. In practice, depth information can be more useful in obstacle avoidance. For example, a stereo algorithm or direct range measures from sonar or infrared sensors may be more appropriate for obstacle avoidance.

- The lighting problem and outdoor navigation.

  Lighting has been a major challenge for vision-based navigation, especially for outdoor navigation. In our tests of indoor navigation, the ambient lighting does not change much. A simple normalization scheme which rescales each input image to zero mean and a unit variance is used to deal with possible changes in ambient lighting. The normalization scheme works for our indoor navigation. To deal with much greater variations in lighting conditions, more elaborate mechanisms should be used. In our indoor navigation, only a black and white visual image is used. For outdoor navigation, typically color images are preprocessed to enhance the contrast between road and nonroad areas [119]. Since our Robot Rome is designed to work only in indoor environment, only limited outdoor experiments have been conducted. Rome has a small wheel diameter and low clearance. It would be interesting to see how the algorithms perform in outdoor navigation. If similar preprocessed inputs like those used in

ALVINN [95] are fed to the system, good performance could be expected.

- Automatic design of states.

  In Chapter 4, those states used in state-based SHOSLIF-N are manually designed by the author for indoor navigation. For an autonomous navigation system, this kind of hand-crafted design should be avoided. Some of the ongoing efforts [134] using spatio-temporal clustering aim for automatic derivation of states for visual navigation.

- Visual attention.

  Right now only a simple visual attention mechanism is employed by the system. The system uses attention windows with fixed positions, and the only decision is whether an attention window or a global view should be used. Visual attention in state-based SHOSLIF-N is used only in discriminating those scenarios which have similar global views but different local appearances in attention windows and requires different steering signals. For a practical autonomous navigation system, more elaborate visual attention mechanisms could be used.

  Some efforts have been made by the Navlab group at Carnegie-Mellon University. Jochem [58] used virtual cameras, or attention windows which are extracted from a global scene. Virtual cameras have been shown useful in lane transition and intersection navigation. Dickmanns' 4D approach [26, 32] naturally incorporates visual attention into the system. The internal state of the 4D approach includes state information for visual attention. This is one of the reasons that the 4D approach is successful. But the 4D approach relies on the

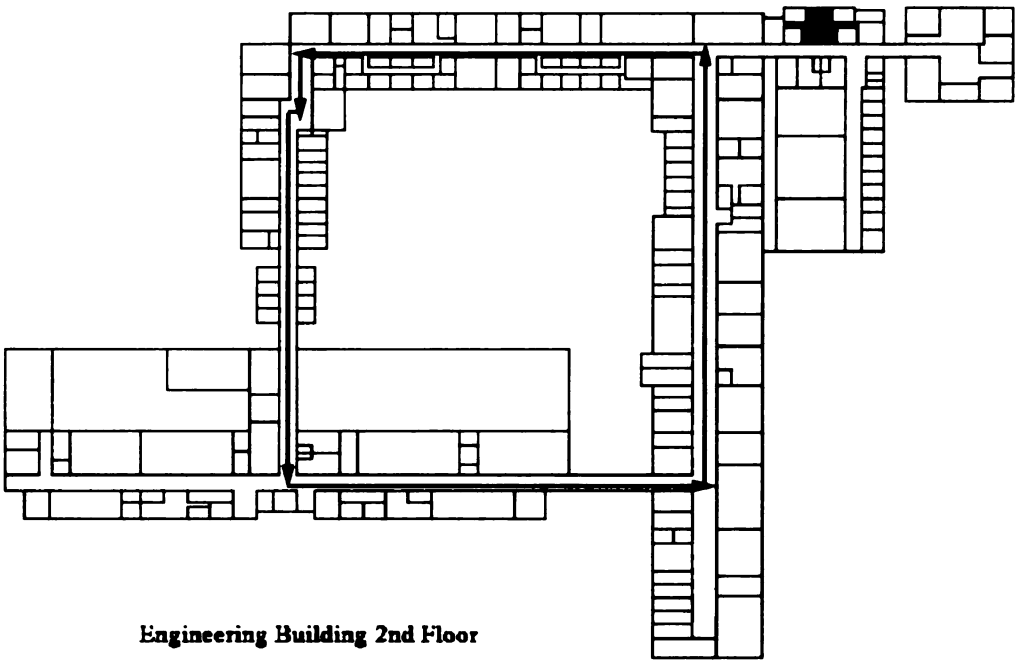
model-driven object tracking. There is still much space for improvement in visual attention for autonomous navigation.

Appendices

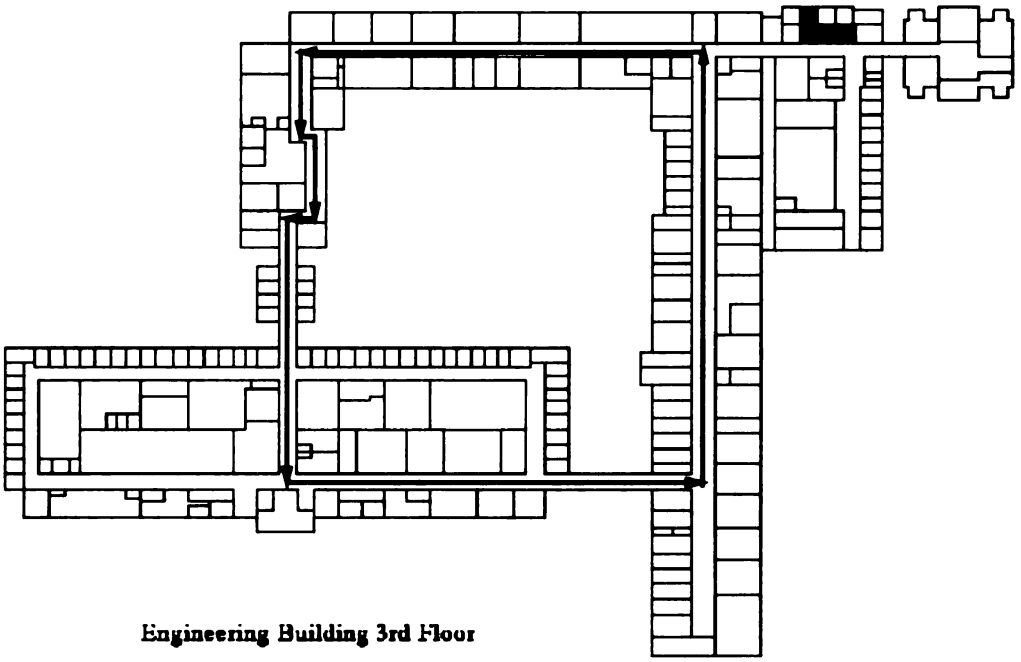
Appendix A

Maps of Test Sites

The second- and third- floors of our Engineering Building serve as the test sites of the thesis work. The tested loops are marked with blue lines.



Engineering Building 2nd Floor



Engineering Building 3rd Floor

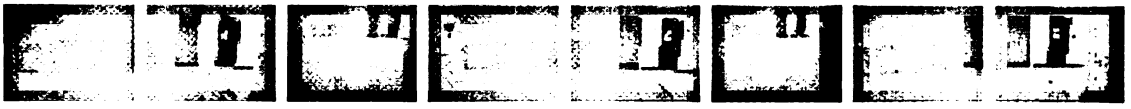
Appendix B

A Set of Training Samples Used in State-Based SHOSLIF-N

This set of training images was collected along the tested loop shown in Fig. 4.6 with a Panasonic GP-KR202 CCD camera, with a 3.6mm fish-eye lens. It includes 273 training samples which were taken during the incremental training on the 3rd floor of the Engineering Building. This set of training samples was used in the study of state-based SHOSLIF-N described in chapter 4. Each training image is displayed with its associated training data (previous state, current state, heading correction).

(0, 0, 0°) (0, 0, −8°) (0, 0, 9°) (0, 1, 0°) (0, 1, −7°) (0, 1, 10°) (1, 2, 0°) (1, 2, 0°)

(1, 2, 0°) (1, 2, 8°) (1, 2, 8°) (1, 2, 8°) (1, 2, 5°) (1, 2, 5°) (1, 2, 5°) (1, 1, 0°)



(1, 1, 0°) (1, 1, 0°) (1, 1, −7°) (1, 1, −7°) (1, 1, −7°) (1, 1, 4°) (1, 1, 4°) (1, 1, 4°)

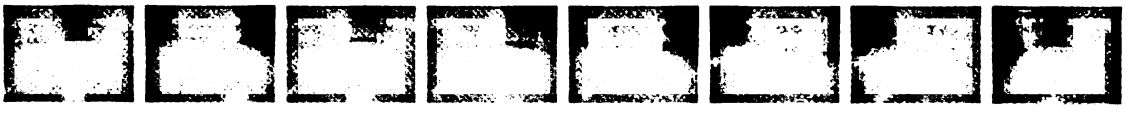
(2, 2, 11°) (2, 2, −7°) (2, 2, 0°) (2, 2, 8°) (2, 2, 10°) (2, 2, 20°) (2, 2, 20°) (2, 2, −11°)

(2, 2, 5°) (2, 2, 12°) (2, 2, −7°) (2, 2, 10°) (2, 0, 6°) (2, 0, −9°) (0, 0, 0°) (0, 0, −10°)

(0, 0, 12°) (0, 0, 0°) (0, 0, −8°) (0, 0, 7°) (0, 0, 0°) (0, 0, −9°) (0, 0, 10°) (0, 1, 5°)

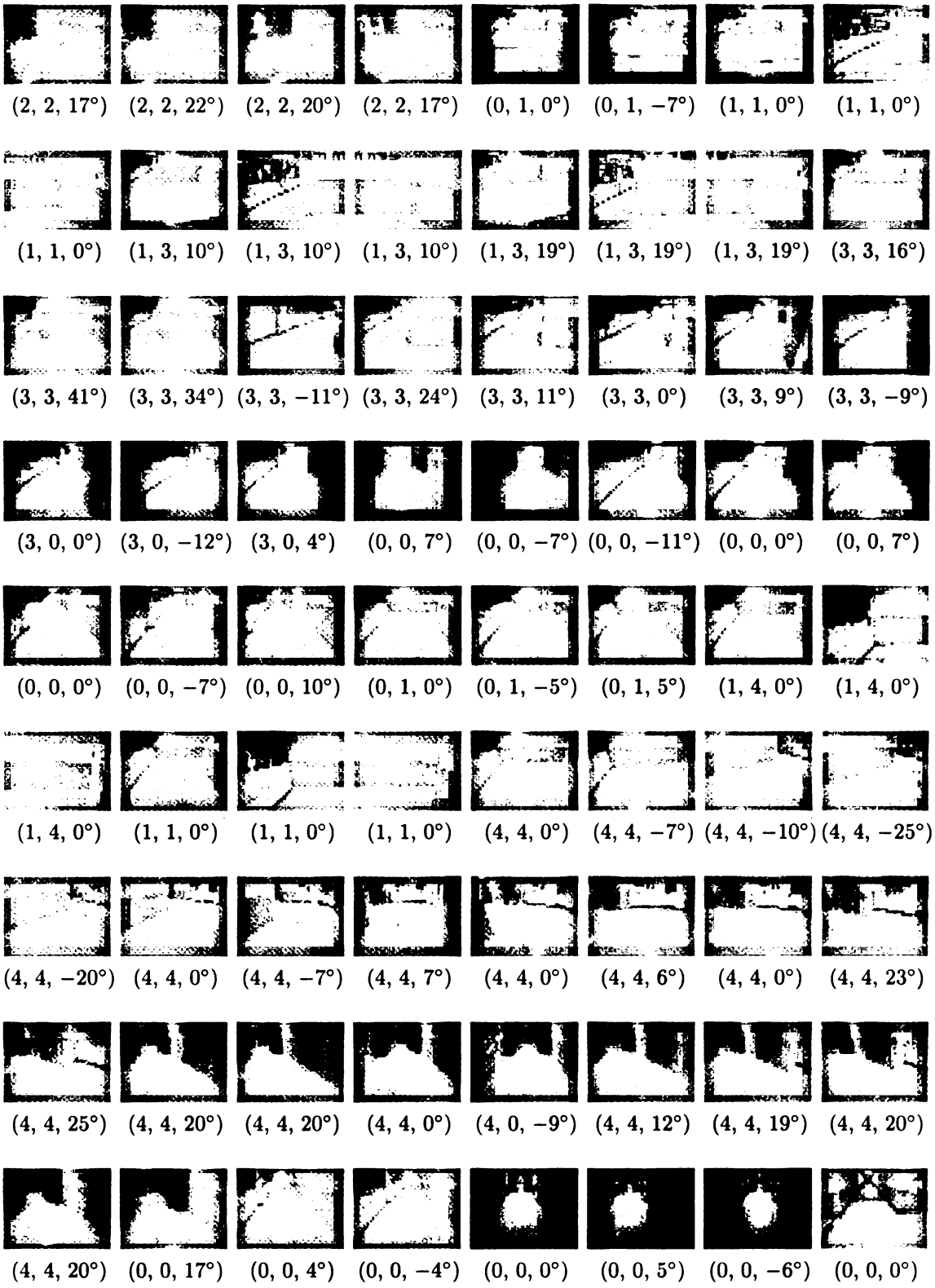
(0, 1, 10°) (0, 0, 5°) (0, 0, 0°) (0, 1, 0°) (0, 1, −7°) (1, 1, −8°) (1, 1, −8°) (1, 1, −8°)

(1, 2, 6°) (1, 2, 6°) (1, 2, 6°) (1, 2, 7°) (1, 2, 7°) (1, 2, 7°) (2, 2, 5°) (2, 2, 5°)



(2, 2, 14°) (2, 2, 10°) (2, 2, 14°) (2, 2, 12°) (2, 2, 10°) (2, 2, 0°) (2, 2, 11°) (2, 0, 10°)

(2, 0, −2°) (0, 0, 7°) (0, 0, 0°) (0, 0, 8°) (0, 0, 0°) (0, 0, −3°) (0, 0, 8°) (2, 2, 16°)


(2, 2, 17°) (2, 2, 22°) (2, 2, 20°) (2, 2, 17°) (0, 1, 0°) (0, 1, −7°) (1, 1, 0°) (1, 1, 0°)
(1, 1, 0°) (1, 3, 10°) (1, 3, 10°) (1, 3, 10°) (1, 3, 19°) (1, 3, 19°) (1, 3, 19°) (3, 3, 16°)
(3, 3, 41°) (3, 3, 34°) (3, 3, −11°) (3, 3, 24°) (3, 3, 11°) (3, 3, 0°) (3, 3, 9°) (3, 3, −9°)
(3, 0, 0°) (3, 0, −12°) (3, 0, 4°) (0, 0, 7°) (0, 0, −7°) (0, 0, −11°) (0, 0, 0°) (0, 0, 7°)
(0, 0, 0°) (0, 0, −7°) (0, 0, 10°) (0, 1, 0°) (0, 1, −5°) (0, 1, 5°) (1, 4, 0°) (1, 4, 0°)
(1, 4, 0°) (1, 1, 0°) (1, 1, 0°) (1, 1, 0°) (4, 4, 0°) (4, 4, −7°) (4, 4, −10°) (4, 4, −25°)
(4, 4, −20°) (4, 4, 0°) (4, 4, −7°) (4, 4, 7°) (4, 4, 0°) (4, 4, 6°) (4, 4, 0°) (4, 4, 23°)
(4, 4, 25°) (4, 4, 20°) (4, 4, 20°) (4, 4, 0°) (4, 0, −9°) (4, 4, 12°) (4, 4, 19°) (4, 4, 20°)
(4, 4, 20°) (0, 0, 17°) (0, 0, 4°) (0, 0, −4°) (0, 0, 0°) (0, 0, 5°) (0, 0, −6°) (0, 0, 0°)

(0, 0, −6°) (0, 0, 5°) (0, 0, 0°) (0, 0, 5°) (0, 0, −6°) (0, 0, 0°) (0, 0, 8°) (0, 1, 0°)

(0, 1, 8°) (0, 1, −6°) (1, 1, 0°) (1, 1, 0°) (1, 1, 0°) (1, 5, 0°) (1, 5, 0°) (1, 5, 0°)

(5, 5, 0°) (5, 5, 7°) (5, 5, −8°) (5, 5, 15°) (5, 5, 23°) (5, 5, 5°) (5, 5, 5°) (5, 5, 20°)

(5, 5, 15°) (5, 5, 20°) (5, 5, 22°) (5, 0, 6°) (5, 0, 11°) (5, 0, −10°) (0, 0, −8°) (0, 0, 13°)

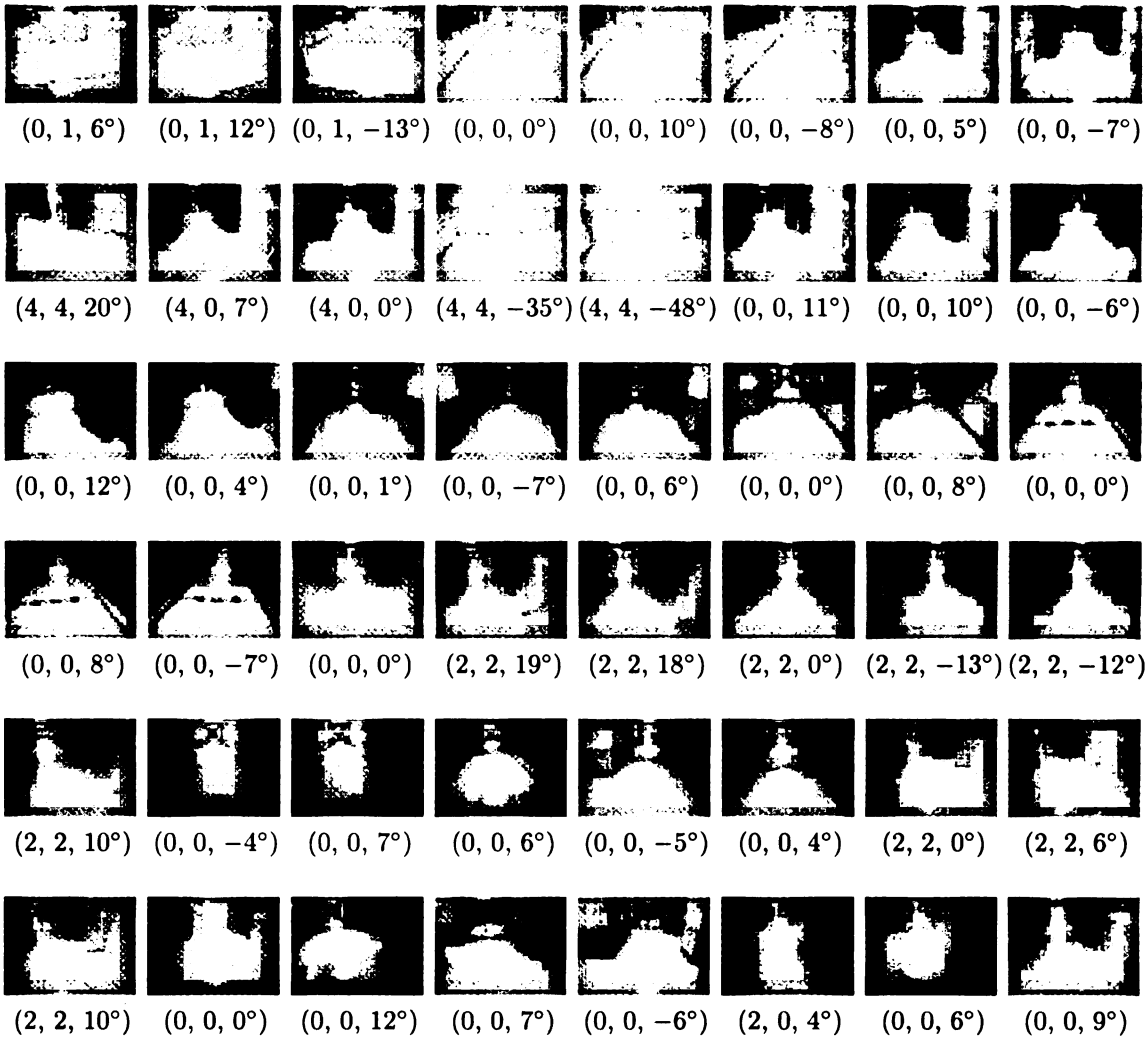
(0, 0, 19°) (0, 0, 11°) (0, 0, −8°) (0, 0, 6°) (0, 0, 10°) (0, 1, 0°) (0, 1, 7°) (0, 1, −5°)

(1, 2, 0°) (1, 2, 0°) (1, 2, 0°) (2, 2, 10°) (2, 2, 15°) (2, 2, 0°) (2, 2, 15°) (2, 2, 28°)

(2, 2, 24°) (2, 0, −8°) (0, 0, 11°) (0, 0, 6°) (0, 0, 0°) (0, 0, 5°) (0, 0, 4°) (2, 2, −7°)

(2, 2, −13°) (2, 2, 25°) (2, 0, 13°) (2, 0, 6°) (0, 0, 0°) (0, 0, 4°) (0, 0, −6°) (0, 0, 0°)

(0, 0, 7°) (0, 0, 0°) (0, 0, 7°) (0, 0, −6°) (0, 0, 4°) (0, 0, −7°) (0, 1, −5°) (0, 1, 0°)



The state transitions of this training set are summarized in Table B.1.

p. state/c. state	0	1	2	3	4	5	sum
0	94	23	0	0	0	0	117
1	0	21	18	6	3	3	51
2	8	0	44	0	0	0	52
3	3	0	0	9	0	0	12
4	3	0	0	0	23	0	26
5	3	0	0	0	0	11	14
sum	111	44	62	15	26	14	272

Table B.1: Summary of state transitions of the training set, where "p. state/c. state" stands for previous state/current state.

Bibliography

[1] N. Ayache and O. D. Faugeras, "Maintaining representations of the environment of a mobile robot", *IEEE Trans. Robotics and Automation*, Vol. 5, No.6, pp. 804-819, 1989.

[2] M. Bertozzi and A. Broggi, "Vision-Based Vehicle Guidance", *Computer*, Vol 30, No. 7, July 1997, pp. 49-55.

[3] M. Bertozzi and A. Broggi, "GOLD: A Parallel Real-Time Stereo Vision System for Generic Obstacle and Lane Detection", *IEEE Trans. Image Processing*, Vol. 7, No. 1, 1998, pp. 62-81.

[4] A. F. Bobick and J. W. Davis, "An appearance-based representation of action", *MIT Media Lab Perceptual Computing Section Technical Report*, No. 369, 1996.

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

[6] R. A. Brooks, "Intelligence without reason", *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 569-595, Sydney, Australia, August, 1991.

[7] C. J. Chappel and J. G. Taylor, "The temporal Kohonen map", *Neural Networks*, 6:441-445, 1993.

[8] J.-L. Chen and A. Kundu, "Rotation and gray scale transform invariant texture identification using wavelet decomposition and hidden Markov model", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 16, No. 2, pp. 208-214, 1994.

[9] M.-Y. Chen, A. Kundu, and J. Zhou, "Off-line handwritten word recognition using hidden Markov model type stochastic network", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 16, No. 5, pp. 481-496, 1994.

[10] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithms for radial basis function networks", *IEEE Trans. Neural Networks*, Vol. 2, No. 2, pp. 302-309, 1991.

[11] S. Chen and J. Weng, "SHOSLIF-N: SHOSLIF for autonomous navigation (Phase I)", Technical Report CPS-94-62, Department of Computer Science, Michigan State University, East Lansing, MI, December, 1994.

[12] S. Chen and J. J. Weng, "Incremental Learning for Vision-based Navigation", Technical Report CPS 96-14, Department of Computer Science, Michigan State University, 1996.

[13] S. Chen and J. J. Weng, "On-Line Incremental Learning for Vision-Guided Real-Time Navigation Using Improved Updating", in *The 10th Scandinavian Conf. on Image Analysis*, Lappeenranta, Finland, June, 1997.

[14] S. Chen and J. J. Weng, "State-based SHOSLIF for indoor visual navigation", to appear in *Proc. 14th Int'l Conf. on Pattern Recognition*, Brisbane, Australia, August, 1998.

[15] X. Chen, E. Dagless, S. Zhang, and B. Thomas, "A real-time plane-view method for following bending roads", *1993 IEEE Symposium on Intelligent Vehicles*, July 14-16, 1993, Tokyo, Japan, pp. 219-224.

[16] P. A. Chou, "Optimal partitioning for classification and regression trees", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 13, No. 4, pp. 340-354, 1991.

[17] J. D. Courtney, *Mobile robot localization using classification techniques*, Master's thesis, Department of Computer Science, Michigan State University, East Lansing, Michigan, 1993.

[18] T. M. Cover, "Estimation by the nearest neighbor rule", *IEEE Transactions on Information Theory*, Vol.IT-14, No.1, pp.50-55, Jan. 1968.

[19] D. R. Cox, "Statistical analysis of time series: some recent developments", *Scand. J. Statist.* , Vol. 8, No. 2, pp. 93-115, 1981.

[20] J. Crisman and C. Thorpe, "Color vision for road following", in *Vision and Navigation: The Carnegie Mellon Navlab*, C. Thorpe, ed., Kluwer, Norwell, Mass., pp. 9-23, 1990.

[21] Y. Cui, D. Swets, and J. Weng, "Learning-based hand sign recognition using SHOSLIF-M", in *Proc. Int'l Conf. Computer Vision*, MIT, MA, pp. 631-636, June 20-23, 1995.

[22] T. J. Darrell, I. A. Essa and A. P. Pentland, "Task-specific gesture analysis in real-time using interpolated views", *MIT Media Lab Perceptual Computing Section Technical Report*, No. 364, 1995.

[23] S. Das and M. Mozer, "Dynamic on-line clustering and state extraction: an approach to symbolic learning", *Neural Networks*, Vol. 11, No. 1, pp. 53-64, 1998.

[24] P. A. Devijver and J. Kittler, *Pattern Recognition*, Prentice Hall, 1982.

[25] E. D. Dickmanns and A. Zapp, "A curvature-based scheme for improving road vehicle guidance by computer vision", in *Proc. SPIE Mobile Robot Conf.*, Cambridge, MA, pp. 161-168, Oct. 1986.

[26] E. D. Dickmanns, "Machine perception exploiting high-level spatio-temporal models", *AGARD Lecture Series 185 'Machine Perception'*, Sept,/Oct., 1992.

[27] E. D. Dickmanns and B. D. Mysliwetz, "Recursive 3-D road and relative ego-state recognition", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 14, No. 2 , pp. 199-213, 1992.

[28] E. D. Dickmanns, "Active bifocal vision", *7th International Conference on Image Analysis and Processing*, Monopoli, Italy, September 20-22, 1993.

[29] E. D. Dickmanns, R. Behringer, D. Dickmanns, T. Hidebrant, M. Maurer, F. Thomanek, and J. Schielen, "The seeing passenger car 'VaMors-P"', in *1994 IEEE Symposium on Intelligent Vehicles*, pp. 68-73.

[30] E. D. Dickmanns, "Performance improvements for autonomous road vehicles", *Intelligent Autonomous Systems-4*, March 1995, Karlsruhe, Germany.

[31] E. D. Dickmanns, "Improvements in visual autonomous road vehicle guidance 1987-1994", in *Visual Navigation: from Biological Systems to Unmanned Ground Vehicles* (Ed. Y. Aloimonos), Larrence Erlbaum Associates, Publishers, New Jersey, 1997.

[32] E. D. Dickmanns, "Vehicles Capable of Dynamic Vision", *Proc. 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya. Japan, August 23-29, 1997.

[33] H. S. Dulimarta and A. K. Jain, "A client/server control architecture for robot navigation", *Pattern Recognition*, Vol. 29, No. 8, pp. 1259-1284, 1996.

[34] Y. T. Feng and D. R. Owen, "Conjugate gradient methods for solving the smallest eigenpair of a large symmetric eigenvalue problems", *Int'l J. Numer. Methods Eng.*, Vol. 39, pp. 2209-2229, 1996.

[35] I. E. Frank and J. H. Friedman, "A statistical review of some chemometrics regression tools" *Technometrics*, Vol. 35, No. 2, pp. 109-148, 1993.

[36] J. H. Friedman, J. L. Bentley and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time", *ACM Trans. on Mathematical Software*, Vol. 3, No. 3, pp. 209-226, 1977.

[37] J. H. Friedman, "Multivariate adaptive regression splines (with discussion)", *The Annals of Statistics*, Vol. 19, No. 1, pp. 1-141, 1991.

[38] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd Ed. Academic, San Diego, 1990.

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

[40] S. Grossberg, "Adaptive pattern classification and universal recoding: I. Parallel Development and coding of neural feature detectors", *Biol. Cybernetics* 23, pp. 121-134, 1976.

[41] S. Grossberg, "Adaptive pattern classification and universal recoding: II. Feedback, expectation, olfaction, illusions", *Biol. Cybernetics* 23, pp. 187-202, 1976.

[42] S. Grossberg, "Content-addressable memory storage by neural networks: A general model and global Liapunov method", in *Computational Neuroscience* (E.L. Schwartz, ed.), pp. 56-65, Cambridge, MA: MIT Press, 1990.

[43] M. Gu and S. C. Eisenstat, "A stable and fast algorithm for updating the singular value decomposition", Technical Report YALE/DCS/RR-966,Yale University, New Haven, CT, 1994.

[44] J. Hancock and C. E. Thorpe, "ELVIS: eigenvectors for land vehicle image system", CMU-RI-TR-94-43, Dec. 1994.

[45] D. J. Hand, *Discrimination and Classification*, Wiley, Chichester, 1981.

[46] T. Hastie and W. Stuetzle, "Principle curve", *J. American Statistical Association*, Vol. 84, No. 406, 1989.

[47] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing, 1994.

[48] M. Hebert, "Building and navigating maps of road scenes using an active sensor", in *Proc. IEEE Int'l Conf. Robotics and Automation*, Philadelphia, PA, pp. 1136-1142, April, 1988.

[49] J. Heikkonen, P. Koikkalainen, and E. Oja, "Self-organizing maps for collision-free navigation", in *Proc. World Congress on Neural Networks*, Protland, OR, Vol. 3, pp. 141-144, 1993.

[50] X. D. Huang, Y. Ariki and M. A. Jack, *Hidden Markov Models for Speech Recognition*, Edinburgh University Press, 1990.

[51] X. Huang, "Phoneme classification using semicontinuous hidden Markov models", *IEEE Trans. Signal Processing*, Vol. 40, No. 5, pp. 1062-1067, 1992.

[52] P. J. Huber, "Projection pursuit", *The Annals of Statistics*, Vol. 13, No. 2, pp. 435-475, 1985.

[53] R. M. Inigo, E. S. McVey, B. J. Berger, and M. J. Mirtz, "Machine vision applied to vehicle guidance", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 6, No. 6, pp. 820-826, 1984.

[54] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, 1988.

[55] D. L. James and R. Miikkulainen, "SARDNET: A Self-Organizing Feature Map for Sequences", in G. Tesauro, D. S. Touretzky and T. K. Leen (editors) *Advances in Neural Processing Systems 7*, 1995.

[56] T. M. Jochem, D. A. Pomerleau and C. E. Thorpe "MANIAC: a next generation neurally based autonomous road follower", *Proceedings of the Image Understanding Workshop*, Washington D. C., April 1993.

[57] T. M. Jochem, D. A. Pomerleau and C. E. Thorpe, "Vision-based neural network road and intersection detection and traversal", *Proc. IEEE Symposium on Intelligent Vehicles*, pp. 344-349, Detroit, Michigan, Sept. 25-26, 1995.

[58] T. M. Jochem, "Vision-based tactical driving", Technical Report CMU-RI-TR-96-14, The Robotics Institute, Carnegie-Mellon University, January 1996.

[59] J. Kangas, "Time-dependent self-organizing maps for speech recognition", in *Proc. of the Intern. Conf. on Artificial Neural Networks*, 1591-1594, 1991.

[60] L. V. Kantorovich and G. P. Akilov, *Functional Analysis in Normed Spaces*, Pergamon, Oxford, 1964.

[61] J. Karhunen and E. Oja, "New methods for stochastic approximation of truncated Karhunen-Loeve expansions", in *Proceedings of the 6th Int'l Conf. on Pattern Recognition*, pp. 550-553, Munich, Germany, Oct. 1982.

[62] S. K. Kenue, "Lanelok: detection of lane boundaries and vehicle tracking using image-processing techniques ", *SPIE Conference On Aerospace Sensing, Mobile Robots IV*, Nov. 1989.

[63] M. Kirby and L. Sirovich, "Application of the Karhunen-Loeve procedure for the characterization of human faces", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 12, No. 1, pp. 103-108, 1990.

[64] K. Kluge and C. Thorpe, "Explicit models for robot road following", in *Vision and Navigation: The Carnegie Mellon Navlab*, C. Thorpe, ed., Kluwer, Norwell, Mass., pp. 25-38, 1990.

[65] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, Addison-Wiley, 1973.

[66] S. Koenig and R. G. Simmons," A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models", in *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, D. Kortenkamp, R. P. Bonasso, R. Murphy (eds.), MIT Press, 1997.

[67] T. Kohonen, "Dynamically expanding context, with application to the correction of symbol strings in the recognition of continuous speech", *Proc. of 8th Int. Conf. Pattern Recognition*, pp. 1148-1151, 1986.

[68] T. Kohonen, "Improved versions of learning vector quantization", *Proc. of Int. Joint Conf. Neural Networks*, Vol. I, pp. 545-550, 1990.

[69] T. Kohonen, "Self-organizing maps: optimization approaches", in T. Kohonen, K. Mäkisara and J. Kangas, editors, *Artificial Neural Networks*, Evsevier Science Publishers, pp. 981-990, 1991.

[70] T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas, "Engineering applications of the self-organizing map", *Proc. IEEE*, Vol. 84, No. 10, pp. 1358-1384, 1996.

[71] D. J. Kriegmaan, E. Triendl and T. O. Binford, "Stereo vision and navigation in buildings for mobile robots", *IEEE Trans. Robotics and Automation*, Vol. 5, No.6, pp. 792-803, 1989.

[72] D. Kuan, G. Philips, and A. Hsueh, "Autonomous land vehicle road following", in *Proc. Int'l Conf. Computer Vision*, IEEE, Piscataway, N. J., 1987.

[73] J. Kuczynski and H. Wozniakowski, "Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start", *SIAM J. Matrix Anal. Appl.*, Vol. 13, No. 4, pp. 1094-1122, 1992.

[74] X. Lebesgue and J. K. Argarwal, "Significant line segments for an indoor mobile robot", *IEEE Trans. Robotics and Automation*, Vol. 9, No.6, pp. 801-816, 1993.

[75] M. M. Loeve, *Probability Theory*, NJ: Van Nostrand, 1955.

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

[77] D. G. Luenberger, *Linear and nonlinear programming* (2nd ed.), Addison-Wesley Publishing Company, 1984.

[78] B. S. Manjunath, S. Chandrasekaran and Y. F. Wang, "An eigenspace update algorithm for image analysis", *Proc. IEEE Int'l Symposium on Computer Vision*, Coral Gables, FL, pp. 551-556, Nov. 20-22, 1995.

[79] J. Mäntysalo, K. Toekkola, and T. Kohonen, "Mapping context dependent acoustic information into context independent from by LVQ", *Speech Communication*, Vol. 14, pp. 119-130, 1994.

[80] J. Mao and A. K. Jain, "Artificial neural networks for feature extraction and multivariate data projection", *IEEE Trans. Neural Network*, Vol. 6, No. 2, pp. 296-317, 1995.

[81] M. Meng and A. C. Kak, "Mobile robot navigation using neural networks and nonmetrical environment models", *IEEE Control Systems*, pp. 31-42, August, 1993.

[82] H. P. Moravec, "Obstacle avoidance and navigation in the real world by seeing a robot rover", Ph.D. dissertation, Stanford Univ., Stanford, CA, Sept. 1980. (Reprinted as *Robot Rover Visual Navigation.* Ann Arbor, MI: UMI Research Press, 1981.)

[83] H. P. Moravec , "The Stanford Cart and the CMU Rover", *Proc. IEEE*, Vol. 71, No. 7, pp. 872-884, 1983.

[84] J. K. Mui, and K. S. Fu, "Automated classification of nucleated blood cells using a binary tree classifier", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 2, No. 5, pp. 429-443, 1980.

[85] H. Murakami and B. V. K. Kumar, "Efficient calculation of primary images from a set of images", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. PAMI-4, No. 5, pp. 511-515, 1982.

[86] H. Murase and S. K. Nayar, "Illumination planning for object recognition in structured environments", in Proc *IEEE Conf. Computer Vision and Pattern Recognition*, Seattle, WA, pp. 31 - 38, June 1994.

[87] H. Murase and S. K. Nayar, "Visual learning and recognition of 3D objects from appearance", *International Journal of Computer Vision*, Vol .14, No. 1, pp. 5-24, 1995.

[88] H. Murase and M. Lindenbaum, "Partial eigenvalue decomposition of large images using the spatial temporal adaptive method", *IEEE Trans. Image Processing*, pp. 620-629, Vol. 4, No. 5, 1995.

[89] S. K. Nayar, H. Murase, and S. A. Nene, "Learning, positioning, and tracking visual appearance", in *Proc. IEEE Int'l Conf. Robotics and Automation*, San Diego, CA, May 1994.

[90] S. K. Nayar, S. A. Nene, and H. Murase, "Real-time 100 object recognition system", in *Proc. IEEE Int'l Conf. Robotics and Automation*, Twin Cities, MN, May 1996.

[91] S. A. Nene and S. K. Nayar, "A simple algorithm for nearest neighbor search in high dimensions", Technical Report No. CUCS-030-95, Department of Computer Science, Columbia University, 1995.

[92] E. Oja and J. Karhunen, "On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix", *J. of Math. Analysis and Applications*, Vol. 106, pp. 69-84, 1985.

[93] E. Oja, "Principle components, minor components, and linear neural networks", *Neural Networks*, Vol. 5, pp. 927-936, 1992.

[94] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation", *Neural Computation*, Vol. 3, No. 1, pp. 88-97, 1991.

[95] D. A. Pomerleau, "Neural Network Perception for Mobile Robot Guidance", Klumer Academic Publishers, 1993.

[96] D. A. Pomerleau, "RALPH: rapidly adapting lateral position handler", *Proc. IEEE Symposium on Intelligent Vehicles*, Detroit, Michigan, Sept. 25-26, 1995.

[97] T. Poggio and F. Girosi, "Networks for approximation and learning", *Proceedings of the IEEE*, Vol. 78, No. 9, pp. 1481-1497, 1990.

[98] T. Poggio, "A theory of how the brain might work", *Cold Spring Habor Symposium on Qualitative Biology*, Vol. LV, pp. 899-910, 1990.

[99] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C (2th Ed.)*, Cambridge University Press, 1992.

[100] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition", *Proceedings of the IEEE*, Vol. 77, No. 2, pp. 257-286, 1989.

[101] R. Reddy, "Foundations and grand challenges of artificial intelligence", *AI Magazine*, pp. 9-21, Winter 1988.

[102] R. Reddy, "To dream the possible dream", *Comm. ACM*, pp. 105-112, May 1996.

[103] R. Reddy, "The challenge of artificial intelligence", *Computer*, pp. 86-98, Oct. 1996.

[104] P. Roberti, "The accelerated power method", *Int'l J. Numer. Methods Eng.*, Vol. 20, pp. 1179-1191, 1984.

[105] M. Rosenblum and L. S. Davis, "The use of a radial basis function network for visual autonomous road following", Univ. of Maryland Center Automat. Res., Tech. Rep. CAR-TR-666, May, 1993.

[106] M. Rosenblum and L. S. Davis, "An improved radial basis function network for visual autonomous road following", *IEEE Trans. on Neural Networks*, Vol. 7, No. 5, pp. 1111-1120, 1996.

[107] I. K. Sethi, and G. P. R. Savarayudu, "Hierarchical classifier design using mutual information", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 4, No. 4, pp. 441-445, 1982.

[108] I. K. Sethi, "Decision tree performance enhancement using an artificial neural network implementation", in *Artificial Neural Networks and Statistical Pattern Recognition* (eds. I. K. Sethi and A. K. Jain), pp. 71-88, Amsterdam: North-Holland, 1991.

[109] S. Shah and J. K. Aggarwal, "Mobile Robot Navigation and Scene Modeling Using Stereo Fish-Eye Lens System", *Machine Vision Applications*, Vol. 10, No. 4, 1997, pp. 159-173.

[110] H.-Y. Shum, K. Ikeuchi, and R. Reddy, "Principle component analysis with missing data and its application to polyhedral object modeling", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 17, No. 9, pp. 854-867, 1995.

[111] R. Simmons and S. Koenig, " Probabilistic Robot Navigation in Partially Observable Environments", in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1080-1087, 1995.

[112] L. Sirovich and M. Kirby, "Low-dimensional procedure for the characterization of human faces", *Journal of the Optical Society of America A*, Vol. 4, No. 3, pp. 519-524, 1987.

[113] S. E. Stead, "Smooth multistage multivariate approximation", Ph.D. Dissertation, Department of Math., Brown University, 1983.

[114] G. Struck, F. Geisler, H. Laubenstein, H. Nagel and G. Siegle, "Interaction between digital road map system and trinocular autonomous driving", *1993 IEEE Symposium on Intelligent Vehicles*, July 14-16, 1993, Tokyo, Japan, pp. 461-466.

[115] R. S. Sutton, Guest Editor, Special Issue on Reinforcement Learning, *Machine Learning*, Vol. 8, No. 3/4, May 1992.

[116] A. M. Thompson, "The navigation system of the JPL robot", in *Proc. Fifth IJCAI*, pp. 745-757, 1977.

[117] C. Thorpe, "FIDO: Vision and navigation for a mobile robot", Ph.D. dissertation, Dept. Comput. Sci., Carnegie-Mellon Univ., Dec, 1984.

[118] C. Thorpe, M. H. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the Carnegie-Mellon Navlab", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 10, No. 3, pp. 362-373, May 1988.

[119] C. Thorpe, ed., *Vision and Navigation: The Carnegie Mellon Navlab*, Kluwer, Norwell, Mass., pp. 9-23, 1990.

[120] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Toward autonomous driving: The CMU Navlab", *IEEE Expert*, pp. 31-42, August, 1991.

[121] TRC, *LABMATE User Manual. Version 5.21L-e.*, Transitions Research Corporation, 1991.

[122] S. Tsuji, J. Y. Zheng, and M. Asada, "Stereo vision of a mobile robot: World constraints for image matching and interpretation", in *Proc. IEEE Int. Conf. Robotics and Automation*, San Francisco, CA, Apr. 1986, pp. 1594-1599.

[123] M. A. Turk, D. G. Morgenthaler, K. D. Gremban, and M. Marra, "VITS–A vision system for autonomous land vehicle navigation", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 10, No. 3, pp. 342-361, 1988.

[124] M. Turk and A. Pentlend, "Eigenfaces for recognition", *Journal of Cognitive Neuroscience*, Vol. 3, No. 1, pp. 71-86, 1991.

[125] R. Wallace, K. Matsuzaki, J. Crisman, Y. Goto, J. Webb, and T. Kanade, "Progress in robot road-following", in *Proc. IEEE Int'l Conf. Robotics and Automation*, San Francisco, CA, pp. 1426-1432, April 1986.

[126] S. J. Walsh, *Indoor Robot Navigation using a symbolic landmark map*, PhD thesis, Department of Computer Science, Michigan State University, East Lansing, Michigan, 1992.

[127] D. L. Wang and M. A. Arbib, "Complex temporal sequence learning based on short-term memory", *Proc. IEEE*, Vol. 78, pp. 1536-1543, 1990.

[128] D. L. Wang and M. A. Arbib, "Timing and chunking in processing temporal order", *IEEE Trans. Syst., Man, Cybern.*, Vol. 23, pp. 993-1009, 1993.

[129] D. L. Wang and B. Yuwono, "Anticipation-based temporal pattern generation", *IEEE Trans. Syst., Man, Cybern.*, Vol. 25, pp. 615-628, 1995.

[130] D. L. Wang and B. Yuwono, "Incremental learning of complex temporal patterns", *IEEE Trans. on Neural Networks*, Vol. 7, No. 6, pp. 1465-1481, 1996.

[131] A. M. Waxman, J. J. Lemoigne, L. S. Davis, B. Srinivasan, T. R. Kushner, E. Liang, and T. Siddalingaiah, "A visual navigation system for autonomous land vehicles", *IEEE J. Robotics and Automation*, Vol. RA-3, No. 2, pp. 124-141, 1987.

[132] A. R. Webb and D. Lowe, "The optimized internal representation of multilayer classifier networks performs nonlinear discriminant analysis", *Neural Networks*, Vol. 3, pp. 367-375, 1990.

[133] J. J. Weng, "On Comprehensive Visual Learning", *Proc. NSF/ARPA Workshop on Performance vs. Methodology in Computer Vision*, pp. 152-166, Seattle, WA, June 24-25, 1994.

[134] J. J. Weng, "The developmental approach to intelligent robots", *in Proc. 1998 AAAI Spring Symposium Series*, Integrating Robotic Research: Taking The Next Leap , Stanford University, March 23-25, 1998.

[135] J. J. Weng, N. Ahuja, and T. S. Huang, "Learning recognition using the Cresceptron", *Int'l Journal of Computer Vision* , 1996. Accepted and to appear.

[136] J. J. Weng and S. Chen, SHOSLIF Convergence Properties and MDF Version of SHOSLIF-N, Technical Report CPS-95-22, Department of Computer Science, Michigan State University, East Lansing, MI, May 1995.

[137] J. J. Weng and S. Chen, "SHOSLIF-N: SHOSLIF for autonomous navigation (Phase II)", Technical Report CPS-95-22, Department of Computer Science, Michigan State University, East Lansing, MI, May 1995.

[138] J. J. Weng and S. Chen, "Autonomous navigation through case-based learning", in *Proc. IEEE Int'l Symposium on Computer Vision*, Coral Gables, FL, pp. 359-364, Nov. 20-22, 1995.

[139] J. J. Weng and S. Chen, "Incremental learning for vision-based navigation", in *Proc. Int'l Conf. on Pattern Recognition*, Vol. IV, pp. 45-49, Vienna, Austria, Aug. 1996.

[140] J. J. Weng and S. Chen, "Vision-guided navigation using SHOSLIF", *Neural Networks*, Vol. 11, pp. 1511-1529, 1998.

[141] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, 1965.

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

[143] A. Wilson and A. F. Bobick, "Using configuration states for representation and recognition of gesture", *MIT Media Lab Perceptual Computing Section Technical Report*, No. 308, 1995.

[144] S. Yakowitz, "Nearest-neighbor methods for time series analysis", *J. Time Ser. Anal.*, Vol. 8, No. 2, pp. 235-247, 1987.

[145] S. Yakowitz, "Nonparametric density and regression estimation for Markov sequences without mixing assumptions", *Journal of Multivariate Analysis*, Vol. 30, pp. 124-136, 1989.

[146] X. Yang, T. K. Sarkar, and E. Arvas, "A survey of conjugate gradient algorithms for solution of extreme eigen-problems of a symmetric matrix", *IEEE Trans. ASSP*, Vol. 37, No. 10, pp. 1550-1555, 1989.

[147] H. Yang, "Conjugate gradient method for the Rayleigh quotient minimization of generalized eigenvalue problems", *Computing*, Vol. 51, pp. 79-94, 1993.

[148] J. A. Zandhuis, "Storing sequential data in self-organizing feature maps", Internal Report MPI-NL-4/92, Max-Planck-Institute fur Psycholinguistik, Nijmegen, the Netherlands.

[149] S. L. Zeger and B. Qaqish, "Markov regression models for time series: a quasi-likelihood approach", *Biometrics*, Vol. 44, pp. 1019-1031, 1988.