Autonomous Mobile Robot Global Self-Localization Using Kohonen and Region-Feature Neural Networks

Jason A. Janet*
Ricardo Gutierrez
Troy A. Chase
Mark W. White
John C. Sutton, III
Center for Robotics and Intelligent Machines
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27695-7911
e-mail:jajanet@eos.ncsu.edu

Accepted October 29, 1996

This article presents and compares two neural network-based approaches to global self-localization (GSL) for autonomous mobile robots using: (1) a Kohonen neural network; and (2) a region-feature neural network (RFNN). Both approaches categorize discrete regions of space (topographical nodes) in a manner similar to optical character recognition (OCR). That is, the mapped sonar data assumes the form of a character unique to that region. Hence, it is believed that an autonomous vehicle can determine which room it is in from sensory data gathered from exploration. With a robust exploration routine, the GSL solution can be time-, translation-, and rotation-invariant. The GSL solution can also become independent of the mobile robot used to collect the sensor data. This suggests that a single robot can transfer its knowledge of various learned regions to other mobile robots. The classification rate of both approaches are comparable and, thus, worthy of presentation. The observed pros and cons of both approaches are also discussed.

この発表では、Kohonenニューラル・ネットワークと領域特性ニューラル・ネットワーク（RFNN；region-feature neural network）を使って、自動モービル・ロボットのグローバルな自己局在化（GSL；global self-

*To whom all correspondence should be addressed.

Journal of Robotic Systems 14(4), 263–282 (1997)

CCC 0741-2223/97/040263-20

localization）を行う、ニューラル・ネットワークをベースにした2つの方法について説明して比較する。2つの方法とも、光学式文字認識（OCR）と同じ方法で、空間の離散領域（地形上の接合点）を分類する。そして、マッピングされたソナー・データは、その領域に対して固有な文字であると仮定される。したがって、自動車両は、探索によって収集した認識データから、現在いる場所を特定できる。堅牢な探索ルーチンによって、GSLの解は、時間、平行移動、回転に対して不変である。また、GSLの解は、センサー・データを収集するために使われるモービル・ロボットからは独立している。そのため、1台のロボットが学習した様々な領域の情報を、他のモービル・ロボットに転送することができる。2つの方法における分類の割合は比較できるので、それも説明する。さらに、2つの方法の長所と短所も考察する。

1. INTRODUCTION

Our objective is to endow mobile robots with the ability to perform self-localization on a *global* level. That is, a mobile robot should be able to use sensor data to autonomously determine which region of space it is in without knowing how it got there. Specifically, we treat the global self-localization (GSL) of a mobile robot as an optical character recognition (OCR) problem.[1,2,3,4] That is, natural landmark signatures can, through computer vision and / or mapped sonar data, assume character-like configurations unique to those landmarks. The ability to identify each *room character* helps a mobile robot determine its global location by association. Since most indoor environments can be easily segmented into *rooms*, different room configurations will define discrete regions of space (topographical nodes). The 10 rooms we used (shown in Fig. 1) have dimensions ranging up to 12.5 m in the *y*-direction and up to 15.5 m in the *x*-direction.

1.1. Time-, Translation-, and Rotation-Invariance

To be truly robost, a GSL technique should have the following characteristics. First, it should be time-invariant for the simple reason that no two robots will explore a room the same way. In fact, a single robot will likely not repeat the same trajectory. Second, it should be translation-invariant because the robot is assumed to not know the actual global coordinates of the region of space it is in, much less the sensor data it collects. Third, it should be rotation-invariant because, through the course of becoming lost, a robot can also become disoriented.

1.2. Using Neural Networks to Solve the GSL Problem

Several approaches to solving the optical character recognition (OCR) problem with neural networks have been investigated. In this research, we compare a Kohonen neural network approach and a region-feature neural network (RFNN) approach to recognizing room characters associated with particular rooms. Both neural nets are known for their abilities to perform classification, recognition, data compression, and association in a self-organizing manner, and neither requires *a priori* knowledge of a sensor data point's affiliation to a particular feature. For the Kohonen, all that must be estimated beforehand is the maximum expected number of features (neurons) per room. When a Kohonen is mature, the point density of the weight vectors approximate the probability distribution of the input space.

The RFNN, on the other hand, is a multi-layered, feed-forward neural network that uses receptive fields and weight sharing to compensate for noise, minor phase shifts, and occlusions.[5] The RFNN also utilizes greedy adaptive learning rates and mature feature preservation to expedite the overall training process, especially when new rooms are added.[3,6] A novel *ad hoc* approach called ''shocking'' is used to solve the instability problem inherent to greedy adaptive learning rates. The RFNN ''feature'' is grounded in computer vision morphology in that the neural network autonomously learns subpatterns unique to various problems. The RFNN is a reliable pattern analysis engine that is easy to implement, compact, and fast. The RFNN requires that we estimate the following: (1) which regions of the data space (room character image) are important; (2) the size and overlap of the receptive-field (feature) patches; (3) the maximum expected number of features per region; and (4) the size and (digitized) resolution of the input space.

2. THE DOMAIN

We assume the environment to be in R^2 space. Specifically, as a robot wanders autonomously or is man-

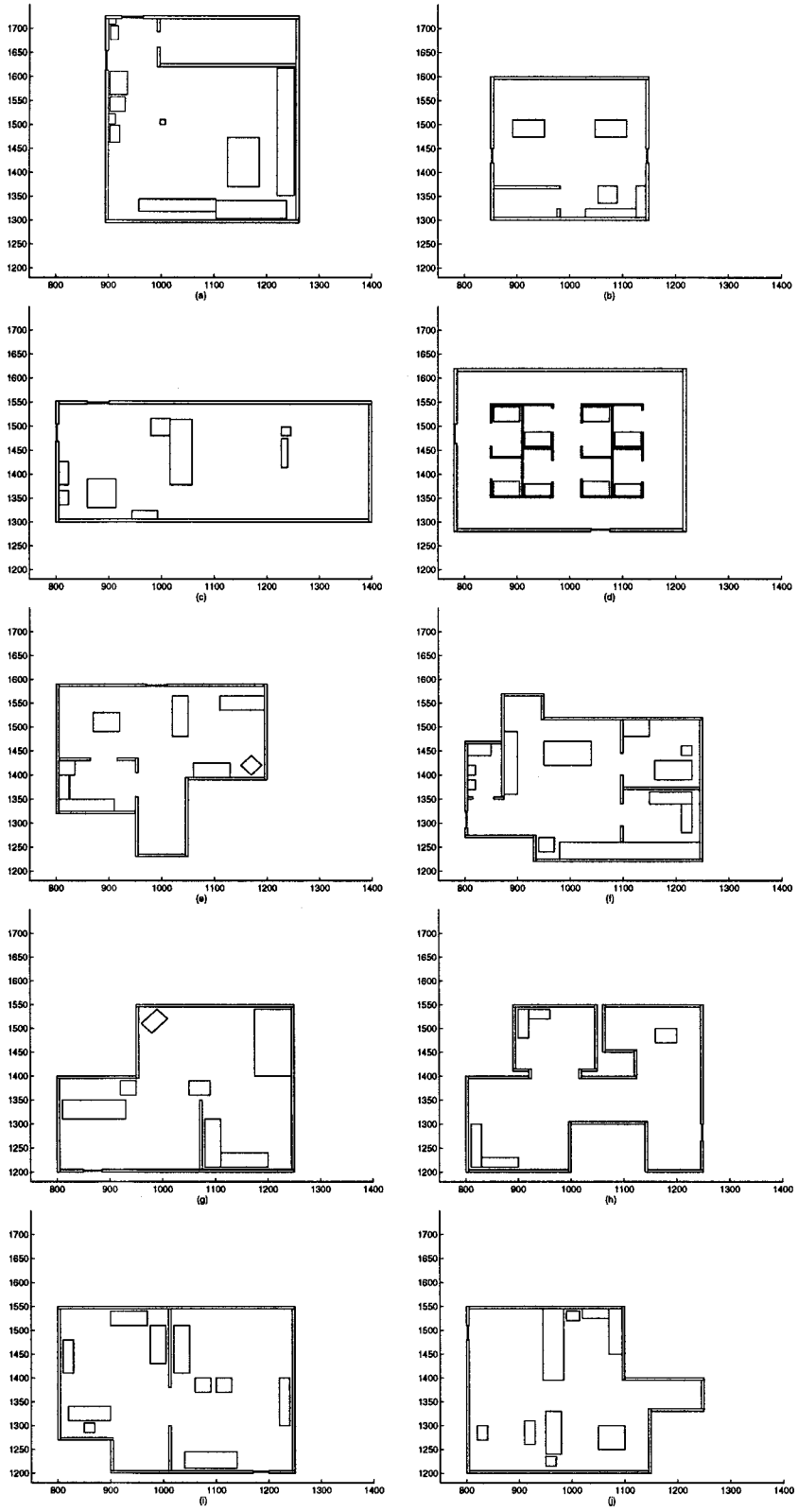

Figure 1. Geometric maps for rooms 1 through 10.

ually driven around a room, its sensor data is mapped to a two-dimensional plane (see Fig. 2). Since ultrasonic sensors are common to nearly all autonomous and semi-autonomous vehicles, it is felt that the sensor used for extracting feature information should be the ultrasonic sensor. This decision is also predicated on the fact that ultrasonic sensors are inexpensive, simple to use, and very reliable.

2.1. The Autonomous Mobile Robot Simulation

Given that the simulation described in[1,2,7,8] has proven accurate at modeling both sonar and robot behavior and that it provides a graphical interface essential to understanding each step of solving this problem, it was considered a suitable platform for creating training and testing data sets. The simulated sonar models are based on actual data observed using narrow beam Polaroid[9] ultrasonic sensors and the wide-beam sonars on our Cybermotion K2A[10] and Nomad 200[11] mobile robots. The simulated dead reckoning models are also based on data observed from the Cybermotion and Nomad mobile robots. Additional rotation errors are based on a host of compasses discussed in ref. 12 and on our Vector 2X compass module.[13] The simulation also ensures that the 160 training sets and 180 test sets are created in a timely fashion, and without the risk of harming either the robot or the environment. Furthermore, the simulation can simultaneously track the dead reckoning coordinates (i.e., where the robot thinks it is) and actual coordinates (i.e., where the robot really is) without needing someone to physically measure and / or estimate the true location of the robot.

2.2. Room Character Generation

A room's unique character is created by clouds of sonar data points collected and mapped by the robot in its travels through a region. Typically, these clouds are clustered near the surfaces encountered by the sonars. With a single-transducer sonar it is difficult to know from a time-of-flight (TOF) reading the specific point that produced an echo because sonars sample a region. Hence, we must assume that each sonar reading occurs along the axis normal to its transducer ($x_S = 0, y_S = TOF$). To map the TOF reading, the point $X_S = [x_S\ y_S]^T$ is transformed from the sonar frame to the global frame according to the robot's dead reckoning. That is, $X = {}^G T_S(X_S)$ where ${}^G T_S$ is the composite coordinate transformation matrix that converts coordinates from the sonar reference frame to the global frame.

2.2.1. Sonar Corruption

Because sonar readings are inherently corrupted by noise, a 6σ Gaussian corruption function is applied to all TOF readings calculated by the simulation. Specifically, for a *calculated* TOF distance, R_{calc}, C_s is a user-specified variable in the closed interval $C_s \in [0, 1]$ that defines the range of noisy *sensed* readings by R_{sens} (see Fig. 3).

2.2.2. Trajectory Corruption

The very motivation for developing low-level sensor-based self-localization techniques stems from the fact that true mobile robot dead reckoning (from odometers, inertial navigation systems, etc.) grows increasingly unreliable with each bump in the floor and turn of the robot. That is, where the robot thinks it is might not be where it actually is. To simulate this phenomenon, a user-defined random Gaussian trajectory corruption is induced as shown in Figure 4. A 6σ Gaussian corruption function is applied to all robot rotations and translations. Rotation corruption is limited to the closed interval $C_r \in [-\pi, \pi]$, and is specified by the user to define the range of potential noise added to each turn. Translation corruption is a variable on the closed interval $C_t \in [-1, 1]$, and is also specified by the user to define the range of potential noise added to each translation. Figure 5 shows mapped sonar data from a simulated robot subjected to different degrees and types of corruption.

2.2.3. Training and Testing Data

A robot can collect data while autonomously driving through sections of a room, carrying out tasks. Trajectories and explored features vary from data set to data set. However, one extremely important requirement of collecting training and testing data is that the robot explore (what it perceives to be) the south and west extremes of each room. This deliberate search is the first step to make the GSL problem translation-independent. That is, to transpose the mapped sonar data as close to the neural network origin as possible, we need mapped data that reflects minimum x- and minimum y-coordinate values (west and south, respectively). Large-scale rotation-invariance is achieved with the assumption that the robot has a compass on board that is accurate to within $\pm 18°$. Everett discusses several off-the-shelf compasses with accuracies better than $\pm 10°$.[12]

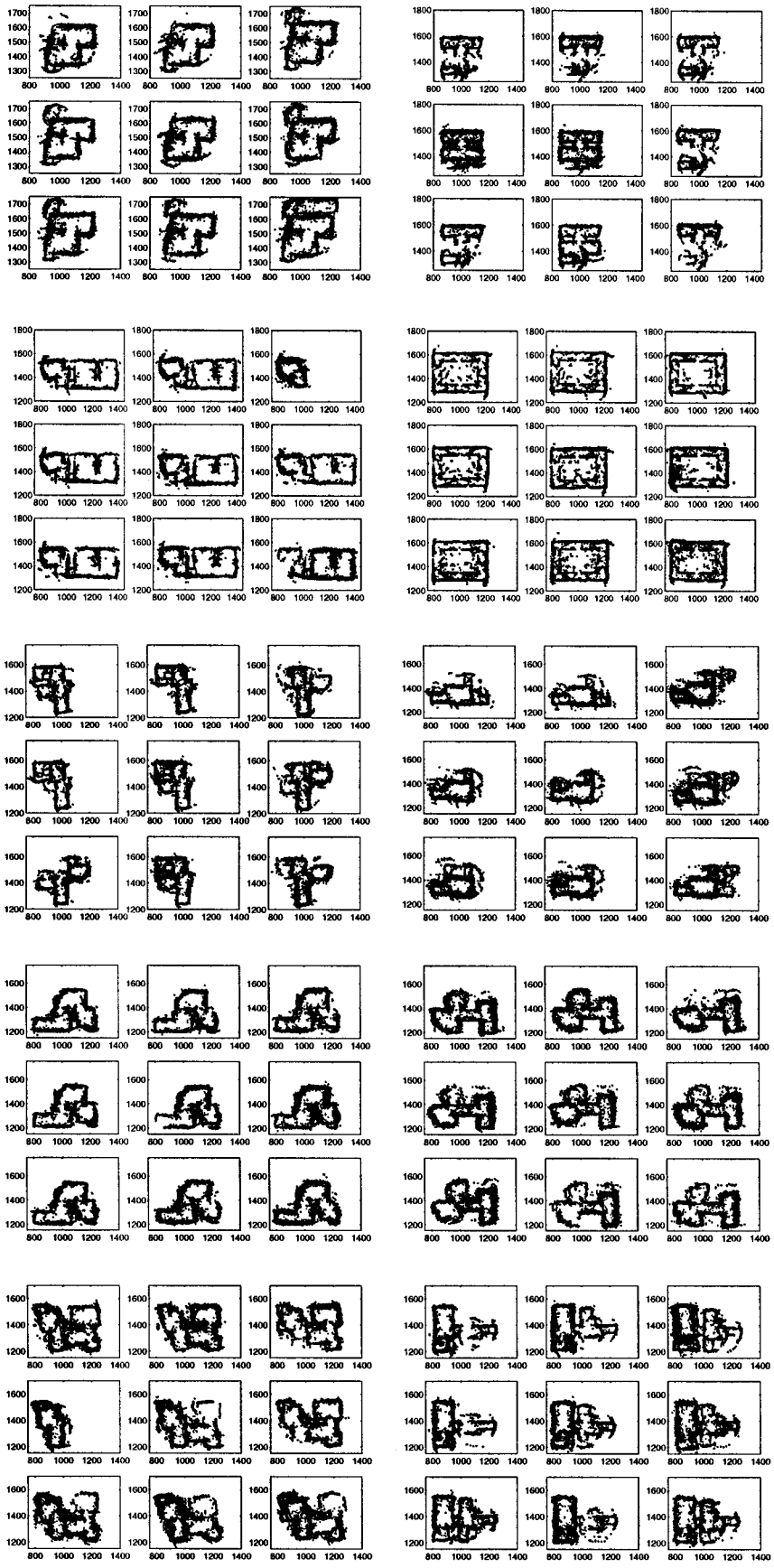

Figure 2. Subset of mapped sonar data from exploring Rooms 1 through 10.

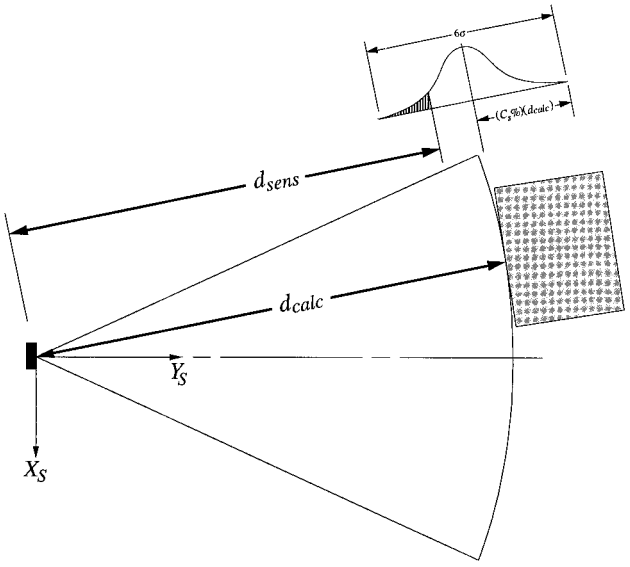

Figure 3. Simulated sonar reading corrupted by white Gaussian noise.

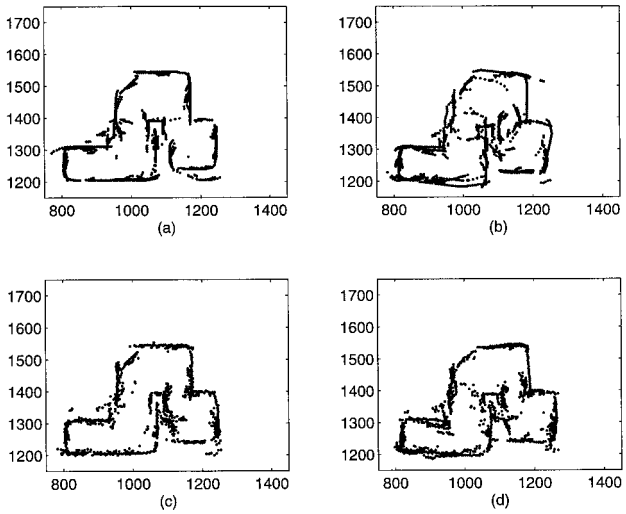

Figure 5. Room character a) $C_s = 0$, $C_r^\circ = 0°$, $C_t = 0$; b) $C_s = 0$, $C_r^\circ = 18°$, $C_t = 10\%$; c) $C_s = 15\%$, $C_r^\circ = 0°$, $C_t = 0$; d) $C_s = 15\%$, $C_r^\circ = 18°$, $C_t = 10\%$.

3. GSL WITH THE KOHONEN NEURAL NETWORK

By design, the neurons of a Kohonen neural network congregate around areas of activity during training (see Fig. 6). During recall, a reactivation of the matured Kohonen nodes by data points in close proximity to those nodes can associate a data set with a class represented by the Kohonen. For the GSL problem, a sonar echo generated by an object's surface is considered an important occurrence since it indicates the presence of certain features. Hence, we desired that the Kohonen organize its neurons such that the configuration of room features (clusters of sonar data) are preserved. The metric we used to determine neuron activation is the Euclidean distance primarily because it is easy to employ. Due to the size of the rooms, it was determined that the initial Kohonen mesh could have 441 neurons (21×21) to be successful at classifying a variety of data sets. However, one of our objectives is to minimize data size, so we also look for opportunities to prune neurons. The method and impact of pruning is discussed in section 3.2.3.

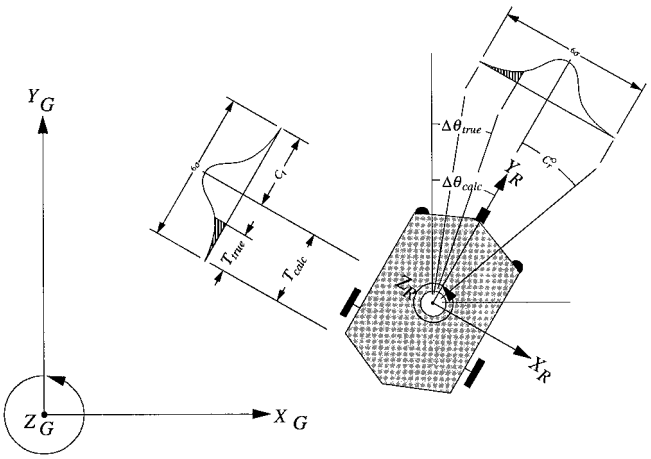

Figure 4. Simulated trajectory corrupted by white Gaussian noise.

3.1. Strategy

A Kohonen neural network will organize a 21×21 mesh of cluster centers to represent the maximum potential 441 features each room is assumed to contain. While training, the Kohonen will make a record of the frequency with which each neuron fires to be used later in pruning redundant and / or deceptive cluster centers. To test the capabilities of the mature Kohonen, test data sets are collected by the simulated robot as it autonomously drives around the room. Generality is tested by allowing test data corruption ranges to be larger than those used for training. Three pre-processing operations will be applied to the Kohonen to reduce the probability of misclassification.

3.1.1. Winner-Take-All Learning

The procedures followed in both training and recall of the Kohonen neural net are fairly standard. The Euclidean distance from each data point X to all of

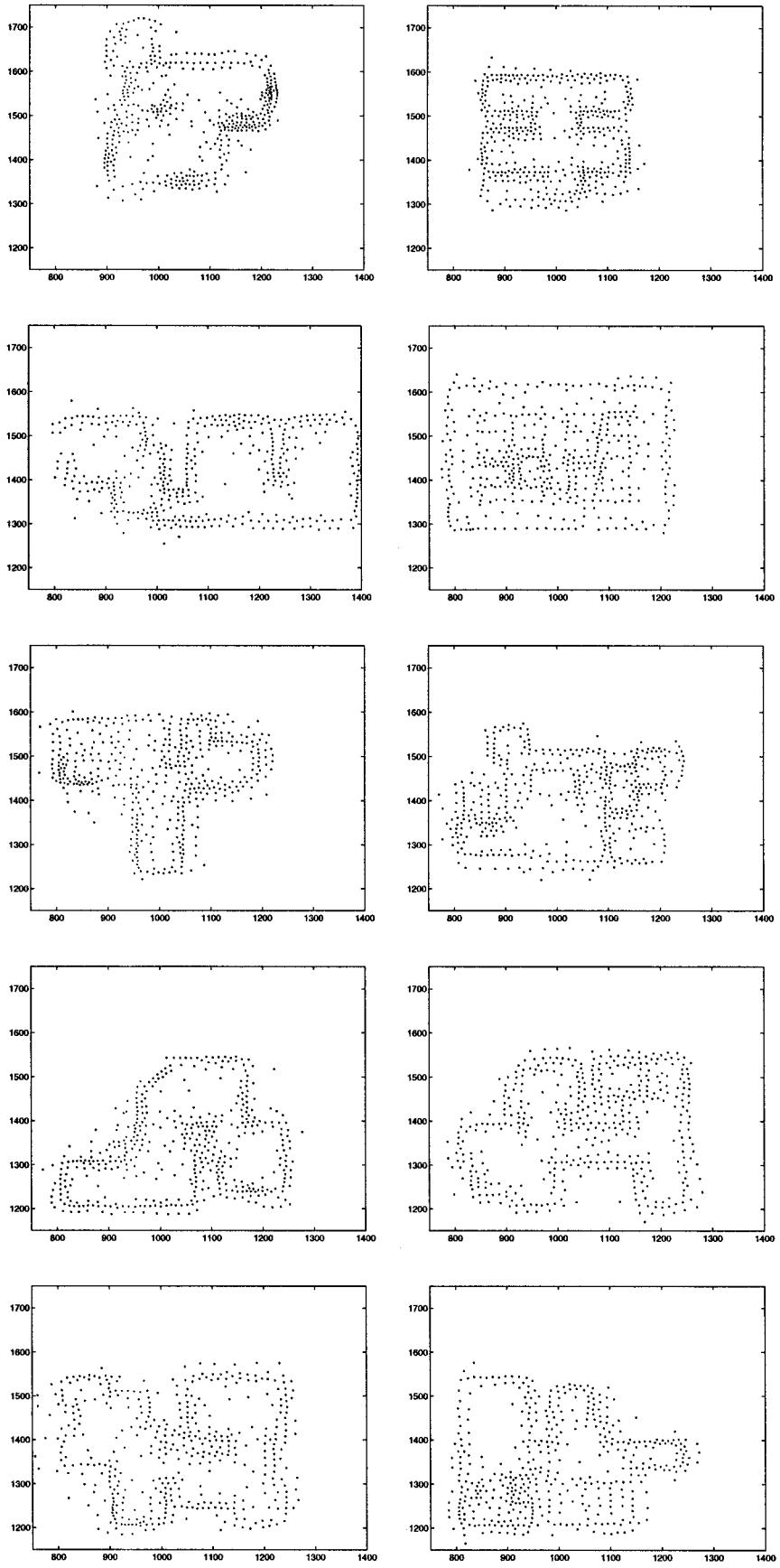

Figure 6. Mature Kohonen node configurations for Rooms 1 through 10.

the Kohonen nodes $\bar{W}_{ij} = [x_{ij}\ y_{ij}]^T$ (for $i, j = 1, 2, \ldots, 21$) is calculated. The initial coordinates of the neurons $\bar{W}_{ij}$ are uniformly distributed over the area bounded by the minimum and maximum x- and y-locations of the collected sonar data. A Kohonen node is the winner $\bar{W}_w$ if

$$\|\bar{X} - \bar{W}_w\| = \min_{i,j=1,2,\ldots,21} \{\|\bar{X} - \bar{W}_{i,j}\|\} \quad (1)$$

$\bar{W}_w$ and its neighbors $\bar{W}_{N_w}$ are then updated based on the learning rate η_ε and the neighborhood size N_ε for a given epoch ε. Our learning rate, initially $\eta_o = 0.5$, is linearly reduced as the Kohonen matures during training to a value of $\eta_f = 0.01$ at epoch $\varepsilon_{max} = 200$.

$$\eta_\varepsilon = \left(\frac{\eta_o - \eta_f}{\varepsilon_{max}}\right)\varepsilon - \eta_o \quad (2)$$

The update rule for the winning node is

$$\bar{W}_w^{k+1} = \bar{W}_w^k + \eta_\varepsilon[\bar{X} - \bar{W}_w^k] \quad (3)$$

The neighbors of the winning node are also updated. Our learning rate for neighbors of the winning node is a function of the indicial distance B from the winning node.

$$\eta_N = (\eta_\varepsilon)(0.7^B) \quad (4)$$

The update rule for neighboring nodes is

$$\bar{W}_{N_w}^{k+1} = \bar{W}_{N_w}^k + \eta_N[\bar{X} - \bar{W}_{N_w}^k] \quad (5)$$

As the Kohonen trains, neighborhood size is gradually reduced. For the first 80 epochs the neighborhood size is a 5×5 window centered over $\bar{W}_w$. Then for the next 80 epochs the neighborhood size is reduced to a 3×3 window. During the final 40 epochs only the winning node is updated. Stop training is based on convergence or $\varepsilon = \varepsilon_{max}$. There are two conditions for convergence: (1) The neighborhood size is only a single neuron; and (2) the change in training error is less than $|\delta|$.

3.1.2. Recall

To test the Kohonen approach to GSL we present previously unseen sensor data to each mature Kohonen. Recall is based on the summation Σ_{rk} of the Euclidean distances between each data point X from the r^{th} data set and the nearest neuron from the k^{th} mature room Kohonen. The resulting Σ_{rk} is a measure of the match between the room data that has just been collected and the mature Kohonen representing a particular room. A Kohonen node mesh is determined to be the winner, Σ_{rw}, if it satisfies

$$\Sigma_{rw} = \min_{k=1,2,\ldots,k_{max}} \{\Sigma_{rk}\} \quad (6)$$

The room w corresponding to Σ_{rw} represents the room that the robot concludes it is currently in. Said differently, if the Kohonen belonging to room w yields the lowest summed Euclidean distance Σ_{rk} (for $k = 1, 2, \ldots k_{max}$) with data set r, then the data in question is classified as belonging to room w.

3.2. Pre-Processing

Four pre-processing procedures are examined in the Kohonen-based approach. First, *gross shifting* is used in all phases of testing to approximate a translationally invariant data space. Second, *fine shifting* is used to improve the fit between a Kohonen and data set, and to further reduce misclassifications caused by translations. Third, *pruning* dormant Kohonen nodes is used to reduce data size and decrease the likelihood of false-positive matches between X and a node whose in-training firing frequency is zero. Fourth, a more aggressive pruning of low-frequency Kohonen nodes is used to further reduce data size and decrease the likelihood of false-positive matches between X and a node whose in-training firing frequency is less than 0.05%.

3.2.1. Gross Shifting

The first step to translation-invariance involves gross shifting. This procedure crudely fits a data set arbitrarily placed in 2D space to a Kohonen node mesh. As Figure 7 shows, minimum x- and y-values of the data and Kohonen are subtracted to transpose both to a common origin. While this transposition addresses the major portion of the translation-invariance problem, it can be seen that a single outlier in either the x- or y-direction can hinder a sound fit between Kohonen and data set.

3.2.2. Fine Shifting

We now need to resolve the less pronounced offset between Kohonen and data set that results from gross shifting. With the minimum x- and y-values trans-

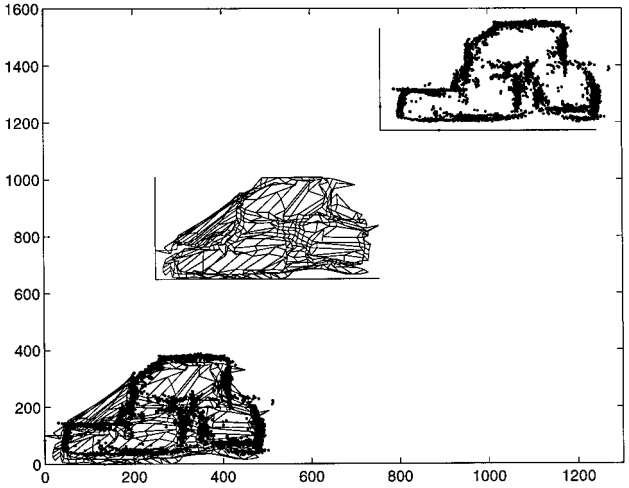

Figure 7. Gross shifting a Kohonen mesh (arbitrarily placed in 2D space) to a room character data set (also arbitrarily placed in 2D space).

lated to the global origin, the Kohonen is finely shifted about the data set to find the best fit. That is, the mature Kohonen mesh is incrementally moved about the data set until the best fit with respect to Σ_{rw} is found. This systematic shifting is done by collectively offsetting the Kohonen nodes *x*- and *y*-values a distance in the range of $[-\rho, \rho]$. Figure 8 shows fine shifting until the best fit. Fine shifting reduces the translation discrepancy caused by outliers in the data set or Kohonen. The range of potential offsets, $\pm\rho$, is sized on the basis of the following issues: the robot's confidence in its ability to find the southwest extreme of a room; the anticipated maximum trajectory deviation; the anticipated maximum sensor corruption; and the expected worst-case room character alteration. Time is the primary reason for (1) bounding the fine shifting range, (2) moving the Kohonen about the data set in increments of 25 cm, and (3) sampling only 1% of the data points (of 2000–4000 per data set). We are presently exploring other optimization routines that can expedite this task of finding the best fit. But, without these constraints, recall is simply too time consuming to be practical.

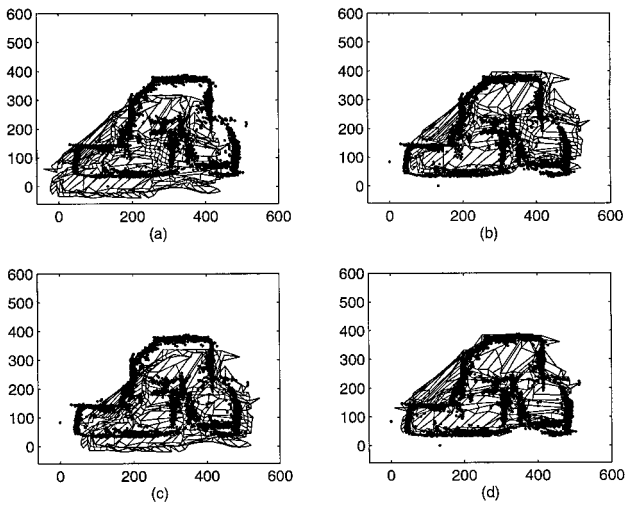

Figure 8. Fine-shifting a Kohonen mesh about a room character data set, $\rho = 1$ meter.

It should also be added that we do not use center-of-gravity to fit Kohonens to data sets because there are times when the robot does not detect every feature. Furthermore, the robot might arbitrarily collect more data in certain regions of a room than in others. The variations of mapped sonar data in Figure 2 support the fact that the center-of-gravity can not be assumed to be consistent from data set to data set. It will be shown in section 3.3 that classification is very sensitive to the Kohonen-data set alignment.

3.2.3. Pruning

Simply put, Kohonen nodes are expected to represent part or all of unique feature in a particular room. If a node exists that does not reflect a feature, it is *pruned*. Pruning a Kohonen node renders that node unavailable to X for use as a nearest neuron. This penalizes the Kohonen since X must then resort to a node that is farther away than the truly closer pruned node. The result is a larger Σ_{rk} for the improperly matched Kohonen-data set pair. Given that Σ_{rk} is the metric with which a data set is associated with a particular room and that errors are emphasized more than *matched* features, the probability of misclassification is expected to decrease.

Two levels of pruning are examined to determine if misclassification errors can be reduced. For the first level, the mature Kohonens are pruned of all nodes whose frequency of being chosen the winning neuron, $f_{w_{ij}}$ (for $i, j = 1, 2, \ldots, 21$), is zero in the last epoch of training. Figure 9 marks with an $\times$ all nodes that are pruned. The second pruning level is more aggressive because we prune all nodes whose frequency of being chosen the nearest or winning neuron, $f_{w_{ij}}$ (for $i, j = 1, 2, \ldots, 21$) is less than 0.05% in the last epoch of training. Figure 9 also shows a plot of all nodes that satisfy the condition $f_{w_{ij}} \leq 0.05\%$.

3.3. GSL Classification with the Kohonen Neural Net

With the 10 Kohonen room networks trained, four pre-processing configurations are tested and com-

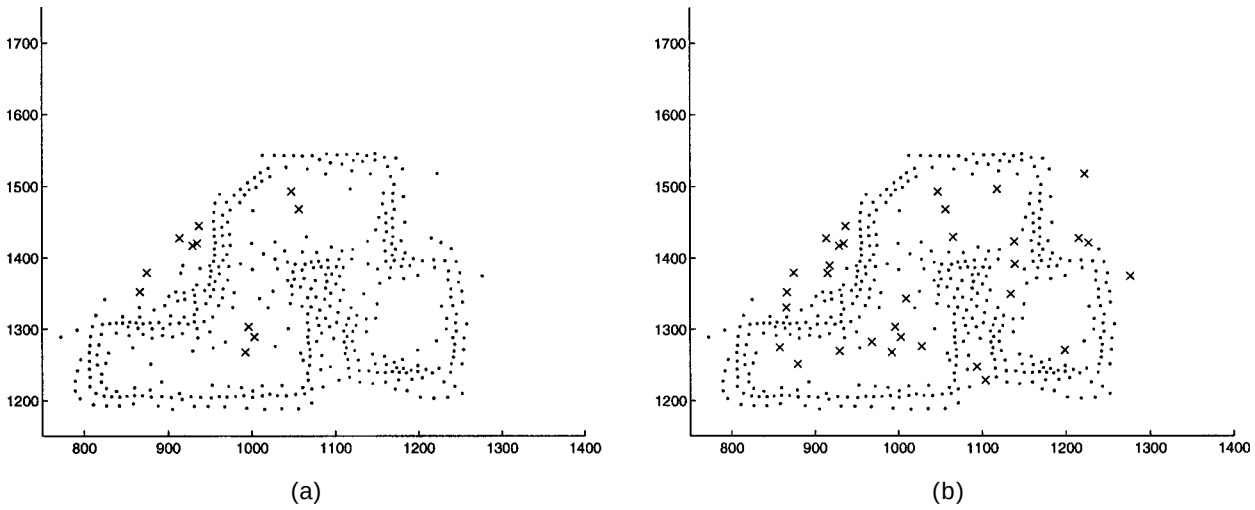

Figure 9. (a) Kohonen nodes pruned for $f_{w_{ij}} = 0$. (b) Kohonen nodes pruned for $f_{w_{ij}} = 0.05\%$.

pared. The results of these systems are used to determine the most reliable approach to room classification. The first configuration uses only gross shifting. The second uses both gross and fine shifting. The third configuration uses gross and fine shifting and pruned $f_{w_{ij}} = 0$ nodes. Configuration four use gross and fine shifting and pruned $f_{w_{ij}} \leq 0.05\%$ nodes. For each room, 18 test sets, each with varying levels and combinations of data corruption, are created and used to test each of the four system configurations.

3.3.1. Gross Shifting Only

Using only Σ_{rk} and gross shifting produced fair results. Figure 10 shows that the misclassification rate for the gross shifting only configuration was 20%. Specific rooms were misclassified more often than others. Room 10 was not misclassified at all while room 4 was misclassified half of the time. This shows that some rooms are distinct enough to be identified just using crude methods like gross shifting. Other rooms, however, are too similar in shape and layout, making recognition more difficult. Although these results were promising, a lower misclassification rate is desired.

3.3.2. Gross Shifting and Fine Shifting

From Figure 11, fine shifting significantly reduced the misclassification rate from 20% to 10.56%. Room 4, however, suffered from more misclassifications. Shown in Figures 2 and 6, room 4 contains data and cluster centers dispersed somewhat evenly throughout the entire room. It is believed that the high misclassification rate associated with room 4 can be attributed to room symmetry and the poor spatial separation of unique features. Regardless, it is clear that fine shifting is an effective pre-processing tool for GSL.

3.3.3. Pruning Dormant Nodes

Kohonen nodes that were placed, through neighborhood organization, in regions where no training data points actually existed ($f_{w_{ij}} = 0$) were apparently troubling from the start. Figure 12 shows the misclassi-

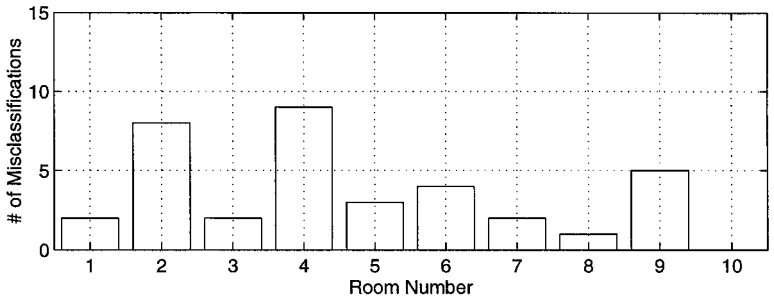

Figure 10. Misclassifications per room for system with *gross-shifting* only.

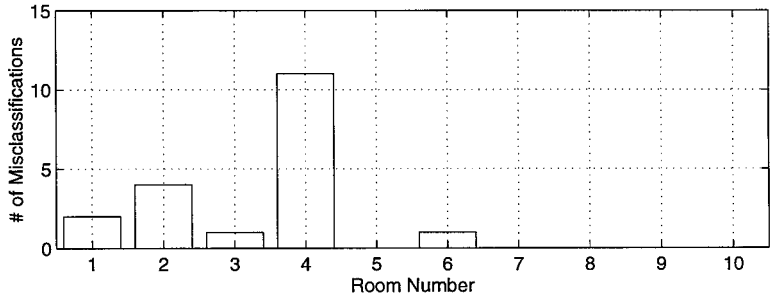

Figure 11. Misclassifications per room for system with *gross-shifting* and *fine-shifting.*

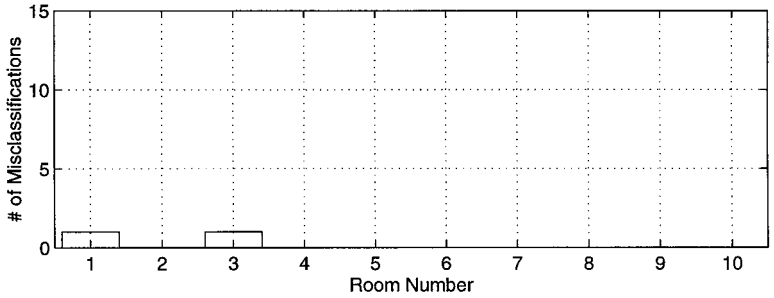

Figure 12. Misclassifications per room for system with *gross-shifting, fine-shifting* and *pruned nodes* for $f_{w_{ij}} = 0$.

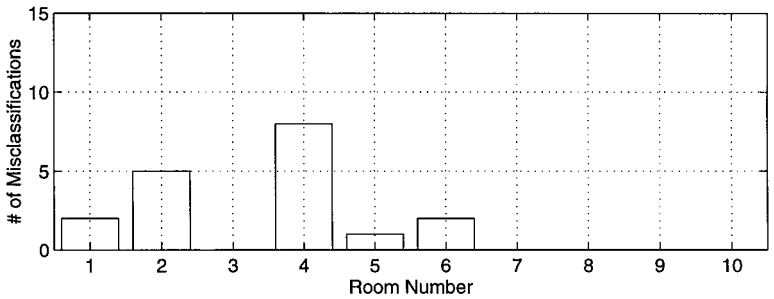

Figure 13. Misclassifications per room for system with *gross-shifting, fine-shifting* and *pruned nodes* for $f_{w_{ij}} \leq 0.05\%$.

fication results for a system with gross shifting, fine shifting, and pruned dormant nodes. There were only two misclassifications (out of 180 test sets) yielding an overall misclassification rate of 1.11%. The performance of this method is a clear improvement over the previous two.

3.3.4. Pruning Low-Frequency Nodes

To see if a more aggressive pruning approach could improve the classifier performance, we tested with gross shifting, fine shifting, and pruned low-frequency nodes ($f_{w_{ij}} \leq 0.05\%$). From Figure 13, it can be seen that, in fact, with a slightly more aggressive pruning approach the classifier performance deteriorated to a misclassification rate of 10%. Hence, it seems the Kohonen is as responsible as the data set to provide feature information. Any loss of true feature information, even from what seems to be outlier data points, can degrade the classifier.

3.3.5. General Comments

Of the four systems, the best classifier utilized gross shifting, fine shifting, and pruning dormant nodes. This is for a static environment, the effects of other

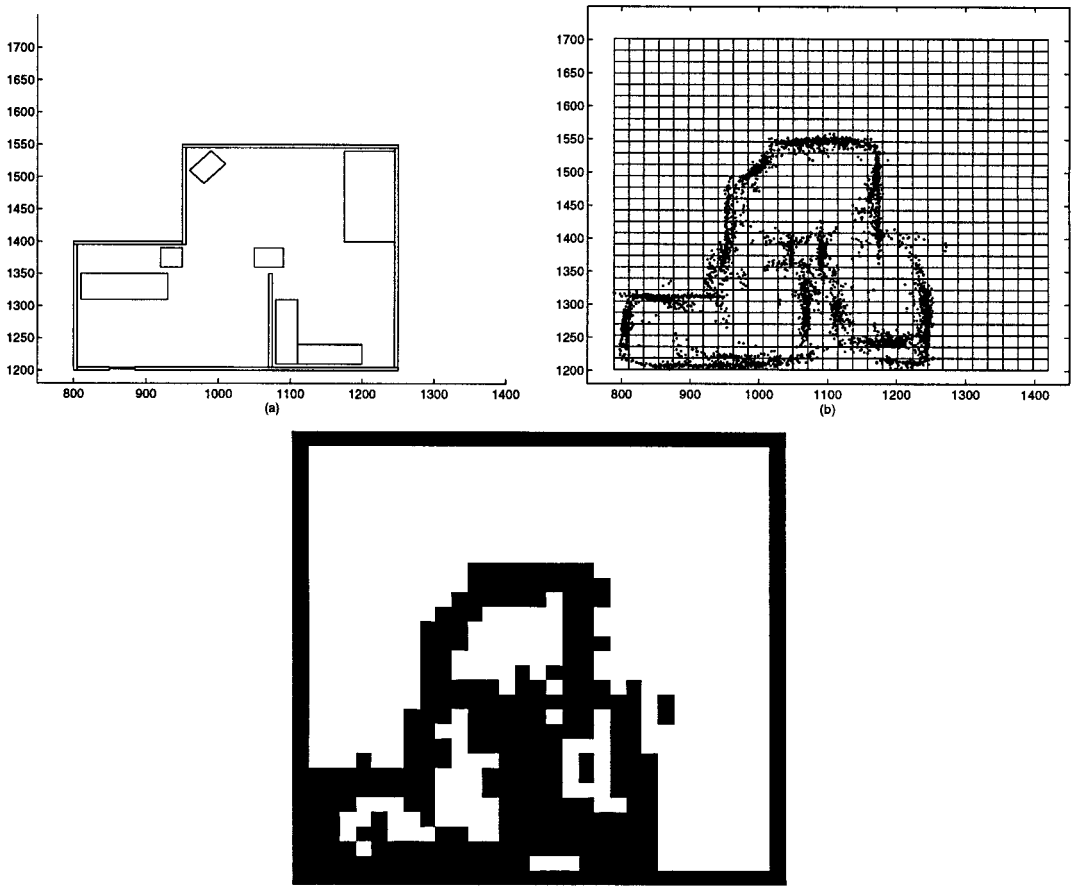

Figure 14. Geometric map 7, sonar data and corresponding digitized room character.

unknown objects on the recognition have not yet been tested. A radical redesign of a room would require the system to relearn the room since the addition of new furniture or the rearrangement of existing furniture would change the signature character of that room. How much the character changes and the effect it has on recognition is situational. This suggests the need for an update learning scheme that notices small physical changes in a previously trained room.

4. GSL WITH THE REGION-FEATURE NEURAL NETWORK

Two very successful and robust implementations of OCR are reported in refs. 14 and 15. Common to both approaches is the use of receptive fields, weight sharing, and at least four hidden layers. While receptive fields and weight sharing help compensate for slight variations in translation and rotation, using four hidden neuron layers results in longer training time, increased computational complexity, and large data sizes.[2] The RFNN utilizes both receptive fields and weight sharing. However, unlike the OCR neural networks in refs. 14 and 15, the RFNN uses only a single hidden layer.

4.1. Pre-Processing

For classification, there are only two pre-processing steps for the RFNN approach. The first is to digitize the sonar data into a 29 $\times$ 29 binary pixel grid. (The number of pixels was determined experimentally to be of sufficient resolution to discriminate between room characters.) The presence of a perceived echo inside a pixel's boundaries activates that pixel. Figure 14 shows one of the 10 geometric rooms, an example simulated sonar data set (generated by a mobile robot in a random wander / explore routine), and the resulting digitized image that functions as the input matrix to the RFNN. The 10 rooms have a data space that covers roughly 16 m by 13 m. That is, it is assumed that the rooms encountered by the robot will not exceed these dimensions. It is already clear that

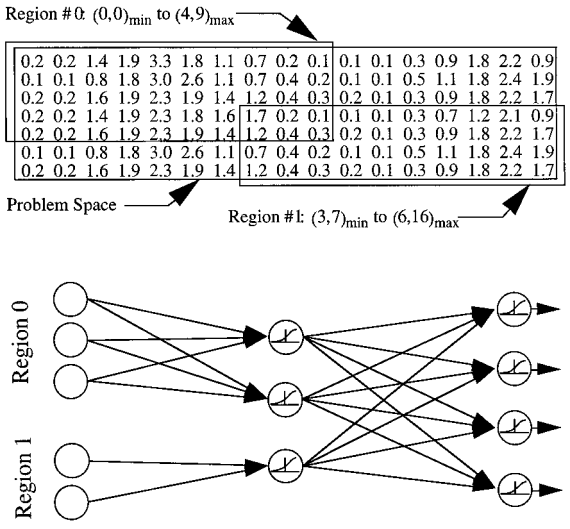

Figure 15. Data space with overlapping regions which define input-to-hidden connectivity and the sections of the data space inside which feature kernels can search for matches.

this is one disadvantage to the RFNN approach to GSL. However, a limit must be placed on the size of the matrix used to input the RFNN. The reason we cannot simply scale each room character to a data space is because the randomness of exploration routines and changing furniture configurations can cause the mobile robot to miss certain features.

The second pre-processing step involves gross shifting the mapped sonar data to achieve large-scale translation-invariance. This is why the grid in Figure 14 begins at the minimum-x and minimum-y points of the mapped sonar data. Large-scale rotation invariance is achieved through the assumption that the robot has a compass accurate to within $\pm 18°$ on board. The RFNN's receptive field architecture is expected to compensate for small-scale variations in translation and rotation.

4.2. Regions: Input-to-Hidden Neuron Connectivity

The input layer is where the training and testing data is introduced to the neural network. The data space can be broken down into two-dimensional regions that can overlap each other or define independent portions of the data space. Figure 15 illustrates an example data space. Each region is defined by its minimum and maximum (*row, col*) locations inside the data space. Figure 15 shows how the arrangement of regions defines the connectivity between the input layer and hidden layer neurons. Specifically, each region has its own dedicated group of hidden layer neurons. If an architecture required fully connected input-to-hidden neurons, the entire set of input data would be confined to only one region. Otherwise, the input-to-hidden neurons can be made partially connected by breaking the data space into more than one region.

4.3. Features: Hidden Neuron Kernels

We use the term *feature* because the RFNN bases its final output decision on the quantity and / or mapping of certain characteristic features unique to an overall larger class of objects, images, or patterns. Some features might be shared by several classes. However, each unique class is characterized by a certain number of particular feature occurrences, or the placement of particular features in the region. A feature is a movable hidden layer neuron with a unique set of input-to-hidden layer synapses. The number of hidden layer neurons for a given region is dictated by the number of features it has. The set of input-to-hidden layer synapses (receptive field) unique to a feature is held in a *feature weight matrix* (FWM). The FWM is convolved with its region to search for a possible feature match. This convolution process is similar to morphological operations in computer vision where a kernel is compared to all parts of an image to look for a pattern match.[16] The number of input-to-hidden layer synapses in each FWM is defined by the feature *patch* size (kernel size). Specifically, a patch is a subset of its respective region and, hence, its size is the number of rows and columns ($\leq$ the size of the region) inside the data space.

4.3.1. Patch Size and Overlap

To determine if, where, and to what degree a feature occurs, the feature patch is placed over portions of its respective region. This operation is like sliding a window over an image where, for each position of the window, the portion of the image inside that window can be measured by the feature neuron for similarity to a previously learned feature (see Fig. 16). The feature's set of input-to-hidden synapses weight the corresponding region elements and measure the sum of weighted inputs with the sigmoid function. Placement of each feature patch depends on the *overlap*. Specifically, the overlap defines the translational change of a patch when it is repositioned. The overlap in the row- and column-directions can be up to one index position less than the size of the patch in both directions.

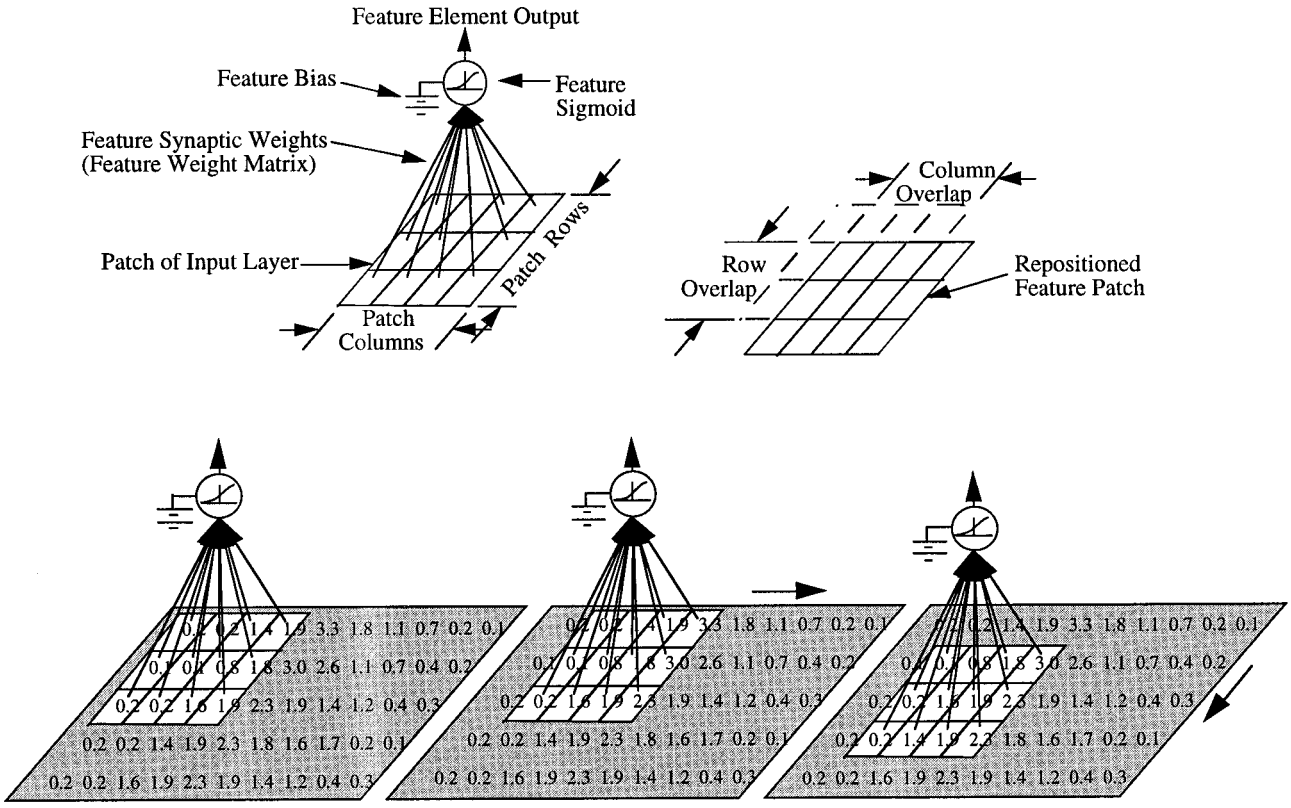

Figure 16. Feature neuron, FWM, patch and overlap. Feature patch convolving with region.

4.3.2. Receptive Fields Help Compensate for Translation and Rotation

Patch overlapping can compensate for slight rotations and translations of a feature by searching areas of the region *at* and *near* the location of the expected feature occurrence. These variations might be caused by noise or a miscalibrated medium used for measuring the data space elements. Looking at successive subsets of the data space in this manner is referred to as using *receptive fields*. The practice of measuring these subsets with a feature whose synaptic weights are position-independent is called receptive field weight-sharing. Unlike computer vision morphology, the neural network decides for itself how a feature should be measured. This is because the neural net not only looks at *whether* a candidate feature does or does not completely match; it also looks at the *degree* of match. In general, we want to minimize the number of features to minimize the overall architecture size.

4.4. Feature Mapping and Hidden-to-Output Connectivity

For each position of a feature inside its region, a unique value associated with that position is produced by the feature neuron. These position-dependent feature values are placed in a *feature element matrix* (FEM) where they are held and later presented to the output neuron layer. Hence, synapses exist between the FEM elements and the output neurons. Each feature has its own FEM. The general idea behind the FEM is to let the output layer know where and to what degree certain features are found. A single feature uses the same input-to-hidden synaptic weights for every position of its patch. Figure 17 shows how a FEM is filled and then connected to an output layer neuron. The set of FEM element-to-output weights is different for each output neuron. This is because each output neuron is entitled to place as much or as little significance on a given type of feature. After all, not all feature types can be expected to be relevant to all output neurons.

4.5. Tailoring the Backprop Algorithm to the RFNN

Due to the uniqueness of receptive-field architectures and weight-sharing, the back-propagation training algorithm must be adapted to the RFNN. For an architecture with $r = 1, 2, \ldots, R$ regions, the r^{th} region is defined in the data space G by $r \in [(row^r_{min}, col^r_{min})$,

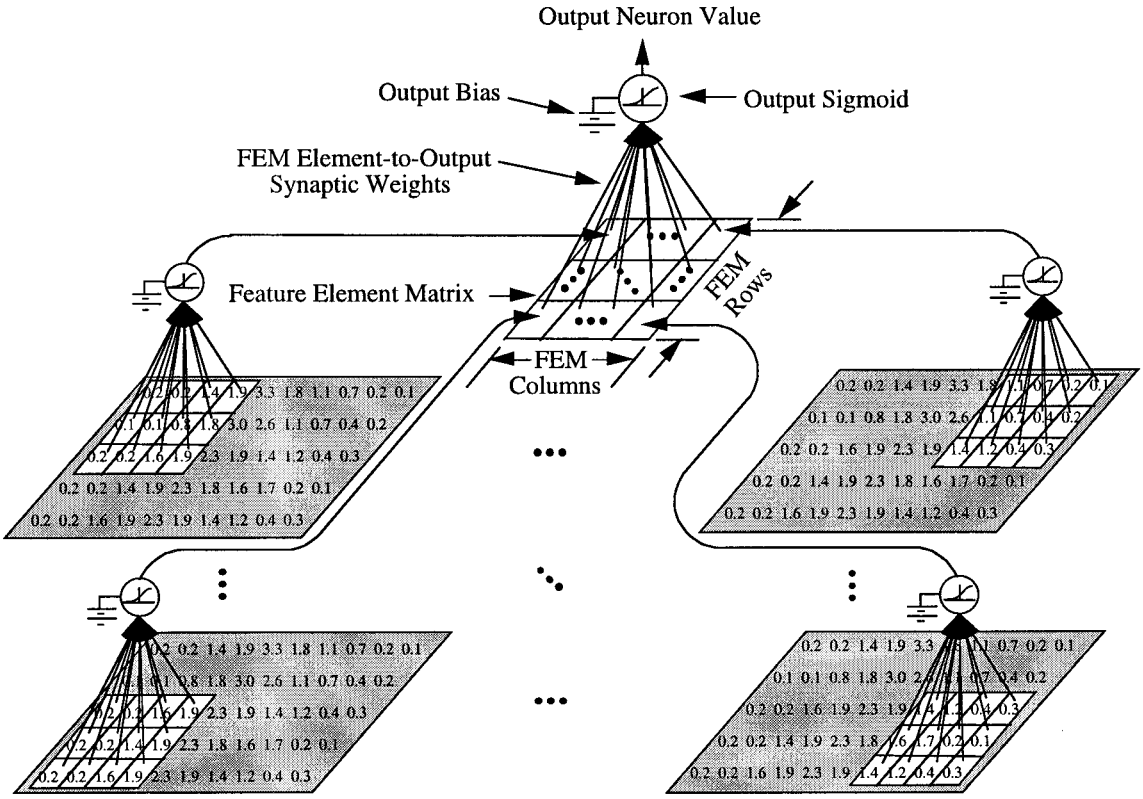

Figure 17. Connectivity of a FEM's elements to an output neuron.

$(row^r_{max}, col^r_{max})]_G$ with respect to the G reference frame. For each region r there are $f = 1, 2, \ldots, F_r$ features each of patch size $P^r_{size} = (N^{P^r}_{rows}, N^{P^r}_{cols})$ and overlap $O^r_{size} = (N^{O^r}_{rows}, N^{O^r}_{cols})$. Hence, for each feature f the FEM size is $\underline{FEM}^{r,f}_{size} = (N^{FEM^{r,f}}_{rows}, N^{FEM^{r,f}}_{cols})$ where,

$$N^{FEM^{r,f}}_{rows} = \left\lfloor \frac{row^r_{max} - row^r_{min} - N^{O^r}_{rows} + 1}{N^{P^r}_{rows} - N^{O^r}_{rows}} \right\rfloor \tag{7}$$

$$N^{FEM^{r,f}}_{cols} = \left\lfloor \frac{col^r_{max} - col^r_{min} - N^{O^r}_{cols} + 1}{N^{P^r}_{rows} - N^{O^r}_{rows}} \right\rfloor \tag{8}$$

and the FWM of input-to-hidden synaptic weights is

$$\underline{FWM}^{r,f} = \begin{vmatrix} \omega^{r,f}_{0,0} & \cdots & \omega^{r,f}_{0,N^{P^r}_{cols}} \\ \vdots & \ddots & \vdots \\ \omega^{r,f}_{N^{P^r}_{rows},0} & \cdots & \omega^{r,f}_{N^{P^r}_{rows},N^{P^r}_{cols}} \end{vmatrix} \tag{9}$$

where each synaptic weight, $\omega^{r,f}_{m,n}$ corresponds to the m^{th} row and n^{th} column of the FWM belonging to feature f in region r. Also, each feature has a bias weight associated with it, $\omega^{r,f}_b$.

Each element of $\underline{FEM}^{r,f}$ is filled by the standard sigmoidal measurement

$$f(net_{i,j}) = \frac{1}{(1 + \exp^{-net_{i,j}})} \tag{10}$$

where i and j are indices (which begin at (0, 0)) of $\underline{FEM}^{r,f}$ and

$$net_{i,j} = \sum_{m}^{N^{P^r}_{rows}} \sum_{n}^{N^{P^r}_{cols}} (\omega^{r,f}_{m,n} a_{x,y}) + \omega^{r,f}_b. \tag{11}$$

The term $a_{x,y}$ is the value of an input element from the data space inside region r. The indices x and y (which also begin at (0, 0)) represent the row and column, respectively, of the input element with respect to the data space reference frame G. Consequently,

$$x = [row^r_{min} + (i)(N^{P^r}_{rows} - N^{O^r}_{rows}) + m],$$

$$y = [col^r_{min} + (j)(N^{P^r}_{cols} - N^{O^r}_{cols}) + n] \tag{12}$$

If, as is the case with the RFNN, the neural net-

work is fully connected between the feature element matrices and output neurons, there exists a synaptic weight between every output neuron and every cell of every FEM. This characteristic gives the RFNN a tendency to be top-heavy. Since each output neuron considers the value associated with each placement of the feature patch, there can be significantly more FEM element-to-output neuron synapses than input-to-feature neuron synapses. So each output neuron $o = 1, 2, \ldots, O$ assumes a value according to the sigmoid

$$f(net_o) = \frac{1}{(1 + \exp^{-net_o})} \tag{13}$$

where,

$$net_o = \sum_{r}^{R} \sum_{f}^{F_r} \sum_{i}^{N_{rows}^{FEM^{r,f}}} \sum_{j}^{N_{cols}^{FEM^{r,f}}} [\omega_{r,f,i,j}^{o} FEM(i,j)] + \omega_b^o. \tag{14}$$

Although this notation seems more complicated than other fully connected feed-forward neural networks, it allows us to construct architectures with receptive fields and weight sharing.[14,15,17] If an architecture does not need receptive fields, the RFNN patch is the same size as the region. This makes the FEM matrices only one element in size. Hence, this notation lends greater flexibility to neural network architectures by giving them the option of using receptive fields.

The back-propagation algorithm for the RFNN is also just an extension of the standard back-prop algorithm with the exception that special learning provisions are made for the movable feature patch. Since feature patches are not necessarily fixed over the data space, each individual synapse can be affected by a multitude of input values per data example $q = 1, 2, \ldots, Q$. From ref. 5, the task of the network is to learn associations between a specified set of input-output pairs $\{(\underline{p}_1, \underline{q}_1), (\underline{p}_2, \underline{q}_2), \ldots, \{(\underline{p}_Q, \underline{q}_Q)\}$. The performance index for the network is

$$V = \frac{1}{2} \sum_{q}^{Q} (\underline{t}_q - \underline{o}_q)^T (\underline{t}_q - \underline{o}_q) = = \frac{1}{2} \sum_{q}^{Q} \underline{e}_q^T \underline{e}_q \tag{15}$$

where $\underline{o}_q$ is the vector of output neuron values resulting from the q^{th} input, $\underline{p}_q$, and $\underline{e}_q = \underline{t}_q - \underline{o}_q$ is the error for the q^{th} input. For the standard backpropagation algorithm we use an approximate steepest descent rule. For a single input / output pair the performance index can be approximated by $V_q = \frac{1}{2} \underline{e}_q^T \underline{e}_q$. For epoch $k = 1, 2, \ldots, K$ the approximate steepest (gradient) descent algorithm for the FEM element-to-output neuron synapse and output neuron bias weights require that

$$\omega_{r,f,i,j}^{o}(k+1) = \omega_{r,f,i,j}^{o}(k) - \alpha_{r,f,i,j}^{o}(k) \sum_{q}^{Q} \frac{\partial \hat{V}_q(k)}{\partial \omega_{r,f,i,j}^{o}(k)} \tag{16}$$

and

$$\omega_b^o(k+1) = \omega_b^o(k) - \alpha_b^o(k) \sum_{q}^{Q} \frac{\partial \hat{V}_q(k)}{\partial \omega_b^o(k)} \tag{17}$$

where $\alpha_{r,f,i,j}^{o}(k)$ and $\alpha_b^o(k)$ are the adaptive learning rates at epoch k for the individual FEM element-to-output neuron synapses and output bias synapses, respectively. Similarly, the steepest descent algorithm for feature biases require that

$$\omega_b^{r,f}(k+1) = \omega_b^{r,f}(k) - \alpha_b^{r,f}(k) \sum_{q}^{Q} \frac{\partial \hat{V}_q(k)}{\partial \omega_b^{r,f}(k)} \tag{18}$$

Unlike the fairly standard synaptic weight update rules of 16 to 18, the feature patch synaptic weights (FWM) must be updated such that they account for every placement in the region. Hence,

$$\omega_{m,n}^{r,f}(k+1) = \omega_{m,n}^{r,f}(k) - \alpha_{m,n}^{r,f}(k) \sum_{q}^{Q} \sum_{i}^{N_{rows}^{FEM^{r,f}}} \sum_{j}^{N_{cols}^{FEM^{r,f}}} \frac{\partial \hat{V}_q(k)}{\partial \omega_{m,n}^{r,f,i,j}(k)} \tag{19}$$

4.5.1. Training with Modified Adaptive Learning Rate Model

It has been shown that ''the method of adaptive learning rates[18] is much faster than steepest descent, generally reducing training time by an order of magnitude, and it is also very dependable. It is not prone to get into trouble and does not require special care . . . [It] is fast, dependable, and highly automatic . . .''[19] The adaptive learning rate modification proposed by Jacobs has become popular for two main reasons: minimal mathematical complexity and numerous reported successes at achieving faster convergences.

The adaptive learning rate model is based on four heuristics that suggest that each weight of a neural network should have its own learning rate, and that these rates be allowed to change over time. Qualitatively, the heuristics are: (1) every parameter of the performance measure should have its own individual learning rate; (2) every learning rate should be allowed to vary over time; (3) when the derivative of

a parameter possesses the same sign for consecutive time steps, the learning rate for that parameter should be increased; and (4) when the sign of the derivative of a parameter alternates for consecutive time steps, the learning rate for that parameter should be decreased. The math model for these heuristic rules are presented below. In general, each synaptic weight in a neural network architecture is allowed to have its own learning rate. We are concerned with the direction in which errors for a synaptic weight decrease over an exponential average f_ω,

$$f_\omega(k+1) = \theta \cdot f_\omega(k) + (1-\theta) d_\omega(k) \tag{20}$$

where $d_\omega(k) = \Sigma_q^Q \, \partial \hat{V}_q(k) / \partial \omega(k)$ and θ defines the weighting of consecutive error associations ($\theta \approx 0.1$). The signs of d_ω and f_ω give us a precise measure of the direction in which the error decreases both currently and recently. The equation for the adaptive learning rate of a synaptic weight is

$$\alpha_\omega(k+1) = \begin{cases} \alpha_\omega(k) \cdot \kappa & \text{if } d_\omega(k) f_\omega(k) > 0 \text{ and } \alpha_\omega(k) < \tau \\ \alpha_\omega(k) \cdot \phi & \text{if } d_\omega(k) f_\omega(k) \le 0 \end{cases} \tag{21}$$

where typically ($\kappa \approx 1.1$) and ($\phi \approx 0.5$).

Despite their success at reducing training times, adaptive learning rate models tend to create instabilities that can cause a neural network to saturate.[a,5] To envision this, we consider that the standard backprop model distributes the error associated with a synaptic weight to prescribe a weight change. Typically, the synaptic weight is updated by only a fraction (less than unity) of the prescribed weight change. While this suggests an inherent stability in the overall weight update method, it also causes training times to be slow. On the other hand, updating a synaptic weight by more than the prescribed weight change (learning rate greater than unity) can significantly reduce training times. However, this ''greedy'' approach increases the likelihood of instabilities.

4.5.2. Heuristic Conditions for Shocking Adaptive Learning Rates

Our research indicates that it *is* possible to let learning rates become greedy ($\tau > 1$) and still maintain relatively stable convergence. This is accomplished through an *ad hoc* approach called ''shocking.''[3,6,20] Simply stated, shocking a neural network reduces all synaptic learning rates to small values near their initial values. The conditions for shocking are quite simple, and yet they have a significant impact on the reduction of failure rates. The first heuristic condition stipulates that *if the training error at epoch k + 1 increases to a value larger than the error at epoch k, the neural net should be shocked* (see Fig. 18(a)). Reverting to small learning rates gives the neural net the opportunity to quickly (re)turn to its original destination or, due to the instability that triggered the shock, locate a better minima on the error surface.

The second heuristic condition for shocking requires that if the learning rates are large enough to significantly impact training, but the training error is decreasing at a very slow rate, the neural net should be shocked (see Fig. 18(b)). The reason for this is as follows: It is believed that when the learning rates are very large, it is possible for the synaptic weights to overshoot and vacillate over a minima while still very slowly converging and not causing the neural net to saturate.[21] As a part of this condition we specify that the learning rates should be large because they need time to grow enough to significantly impact the convergence. Hence, we can restate the second condition with, *if a number of epochs have elapsed* $\Delta k = k - k_{shock}$ *since the last time the net was shocked* $\Delta k \ge MRI$ (*minimum reset interval*) *and the training error is decreasing at a rate less than a specified minimum reset slope, MRS, the neural net should be shocked.* It has been observed that shocking under these conditions significantly reduces the chances of saturation and, hence, stabilizes greedy adaptive learning.[6]

4.6. Locking Matured Features to Expedite Training

The RFNN software allows for the preservation of matured synaptic weights. Loading previously learned features gives a new or modified neural net ''prior knowledge'' in the training process. In ref. 3 the training time to recognize eight different outdoor traffic signs was reduced 80% by preserving the features matured on three examples of a very small subset of classes. In ref. 4 training time for the GSL problem on actual data was reduced by 75%. Also in ref. 4 it was shown that two different mobile robots could share their matured features (sharing basic knowledge) to expedite the cross-platform training process. These concepts are significant to the GSL problem because the process of learning new rooms can be expedited when features learned from other rooms or other platforms are used.

[a]Saturation puts a neural network on a portion of the error surface that is difficult, if not impossible, to recover from.

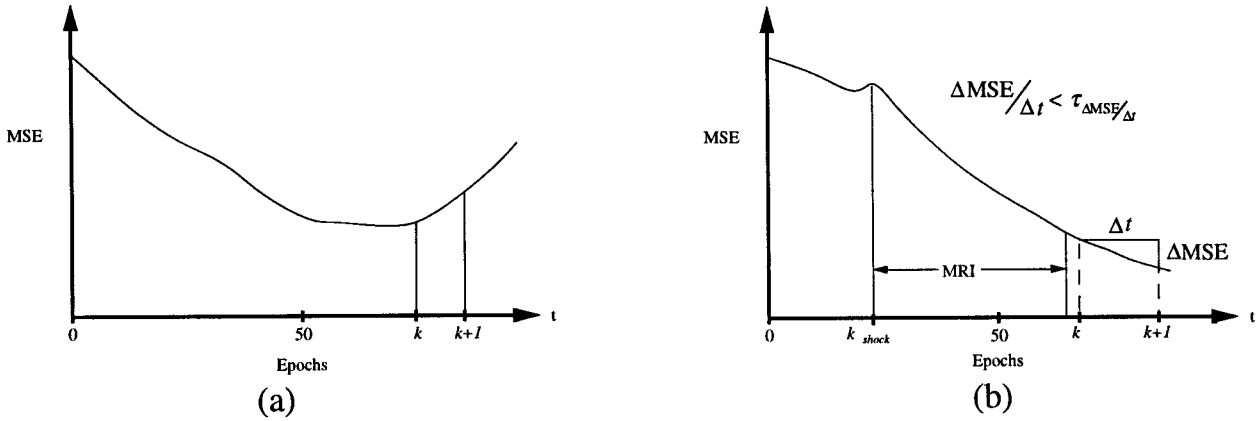

Figure 18. Shock conditions (a) training error increases from global minima; and (b) slow convergence.

4.7. Choosing an Architecture for the GSL/OCR Problem

The size of the architecture should stay as small as possible to minimize data size and training time. In the context of the RFNN software this requires that a neural network should have as few regions and features as possible. Single-region, multiple-feature, 10-output RFNNs are used in this article to learn the relevant features common and unique to each of the 10 rooms, introduced by the 29×29 binary pixel data space. Four different architectures are examined: the first has only one feature (sub-image); the second has two features; the third has three features; and the fourth has four features. Common to all architectures is the feature patch size (sub-image size of 5 rows by 5 columns), maximum overlap (4 rows and 4 columns), and that the entire data space is viewed through a single region.

4.8. GSL Classification with the RFNN

Shown in Figure 19, all four candidate architectures yielded classification rates better than 96.67%. Of the four, the single-region, three-feature, 10-output architecture yielded the highest classification rate, 98.89%. Of course, there is a tradeoff between data size and classification rates. After all, the discrepancy between the lowest and highest classification rates might actually be trivial when compared to training time or the amount of RAM needed for the larger architecture. What is truly interesting, however, is that only a single hidden layer neuron with a 5×5 patch was necessary to properly classify more than 96% of the 180 test sets from 10 different rooms. Even the best architecture seen in this research needed only three hidden layer neurons.

5. COMPARING KOHONEN- AND RFNN-BASED GSL

To get a 98.89% classification rate, the RFNN requires significantly less pre-processing time than the Kohonen. That is, the RFNN requires only two simple pre-processing steps (gross shifting and digitization) whereas the Kohonen neural network requires three (gross shifting, fine shifting, and dormant node pruning). Fine shifting, in particular, is extremely time consuming because, for each data set, the Kohonen is forced to do recall several times on sampled and incrementally shifted sonar data until a best fit between the Kohonen mesh and sonar data is found. Without fine shifting, the Kohonen neural network cannot compensate as well as receptive fields for small-scale translation and rotation discrepancies. Another important advantage to RFNN-based GSL

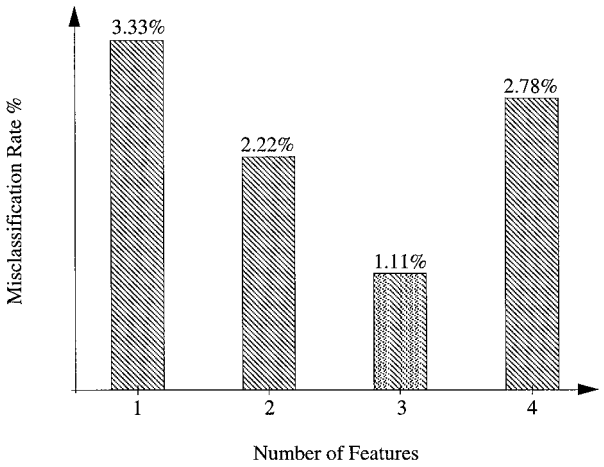

Figure 19. Misclassification rates of single region, variable-feature, ten-output architectures.

is that it can utilize previously learned features to learn any new rooms added. In fact, it was shown in refs. 3 and 4 that the training time is significantly reduced when previously learned features are utilized. Our Kohonen approach, on the other hand, requires us to create a new Kohonen each time a new room is added. It also requires that training begin without the advantage of previously learned features (i.e., from ''scratch'').

The Kohonen certainly has its merits, though. First, its cost function does not penalize for missing features. Second, the Kohonen does not require *a priori* knowledge of the data space size. The RFNN, on the other hand, not only requires that we assume the maximum dimensions of the data space, and an appropriate pixel size (resolution). Like most computer vision principles (threshold, kernel size, etc.), predicting a necessary field of view and / or resolution for a digitized pattern (like the room character) can be rather subjective. Third, algorithmically the Kohonen is much easier to train. However, it can be more time consuming than the RFNN-based approach because the input data for the RFNN is compressed with the digitization preprocessing step. Initialization is easier for the Kohonen because nodes are uniformly distributed over the data set size. Initialization for the RFNN is more subjective, however, because the synaptic weights must be initialized to values small enough to prevent saturation. Finally, the Kohonen seems more tolerant of a wider range of learning rates and neighborhood sizes. However, the RFNN seems rather sensitive to initial learning rates outside a small spectrum (between 0.002 and 0.004).

6. CONCLUDING REMARKS

Global self-localization is intended to enable a mobile robot to determine its position without knowing how or when it arrived there. The significance of GSL is that the discrete regions of space that are memorized by the robot can be an integral, high-level component of a topological network. This article proposes an OCR approach to solve the GSL problem. By mapping simulated sonar data to a two-dimensional plane we can generate a character-like pattern unique to a room explored by a robot. Recognizing the character suggests that the robot can recognize the room.

There are myriad variables in the overall GSL problem. Room configuration, for one, has a significant impact on the classification rate. For example, if certain doors are open, the mobile robot could end up collecting sonar data from more than a single region. Rearranged furniture and even other moving objects can also seriously alter the room character. The wander / explore routine, as well, can play a sometimes unpredictable role. That is, for every set of wander / explore rules, there is a room configuration that will hamper the robot's access to various features. Our best wander / explore routine was one where the robot would randomly choose a steering velocity and a travel distance.[4] Drive speed would depend on the proximity of detected obstacles. The robot would travel at these drive and random steer speeds for the random travel distance and then generate a new steer speed and travel distance.

While the classification rates of both models are promising, we acknowledge that what is documented in this article is based on simulated data. In ref. 4 we used two different mobile robot platforms (a Cybermotion K2A and a Nomad 200) to collect sonar data from actual dynamic environments to examine how well both robots utilize and share components of the RFNN. It is clear from ref. 4 that there are some other variables that should be accounted for. For example, the size of the robot has an effect on which portions of a region are accessible and, hence, explored. Placement of the sonars on the platform also determines which features can be detected. That is, on the Cybermotion platform there are two levels of sonars: some that can see above table tops and some that can see below table tops. The Nomad 200 platform, on the other hand, does not have the same vertical range. Thus, room characters mapped by the Cybermotion platform are sometimes thicker than those mapped by the Nomad 200.

REFERENCES

1. J. A. Janet, et al. ''Global self-localization for autonomous mobile robots using Kohonen neural networks,'' *Proc. IEEE/RSJ Int. Conf. Intell. Rob. Syst.*, Pittsburgh, 1995.
2. J. A. Janet, et al. ''Global self-localization for autonomous mobile robots using region- and feature-based neural networks,'' *Proc. IEEE/SICE/AEI Int. Conf. Ind. Electron.*, Nov. 1995, Orlando, FL.
3. J. A. Janet, et al. ''Pattern analysis for autonomous vehicles with the region- and feature-based neural network,'' *Proc. IEEE Int. Conf. Rob. Autom.*, Minneapolis, MN, 1996.
4. J. A. Janet, D. S. Schudel, M. W. White, and W. E. Snyder, ''Global self-localization for acutal mobile robots: Generating and sharing topographical knowledge using the region-feature neural network,'' *Proc. IEEE Int. Conf. Multisensor Fusion Integration Intell. Syst.*, Washington, DC, 1996.

5. S. Haykin, *Neural Networks: A Comprehensive Foundation,* Macmillan College Publishing, New York, 1994.
6. J. A. Janet, *The Region- and Feature-Based Neural Network Software: A User's Guide,* North Carolina State University, 1996.
7. J. A. Janet, R. Gutierrez, M. G. Kay, and R. C. Luo, ''Autonomous mobile robot self-referencing with sensor windows and neural networks,'' *Proc. IEEE/SICE/AEI Int. Conf. Ind. Electron.,* Orlando, FL, 1995.
8. J. A. Janet, R. C. Luo, and M. G. Kay, ''Autonomous mobile robot global motion planning and geometric beacon collection using traversability vectors,'' *IEEE Trans. Rob. Autom.,* **13**(1), 1997.
9. Polaroid, Ultrasonic Ranging System, Cambridge, MA, 1982.
10. Cybermotion, *User's Manual,* Roanoke, VA.
11. Nomadic Technologies, Inc., Mountain View, CA.
12. H. R. Everett, *Sensors for Mobile Robots: Theory and Application,* A. K. Peters Ltd., Wellesley, MA, 1995.
13. Precision Navigation, Inc., Mountain View, CA, 1995.
14. Y. LeCun, et al., ''Handwritten digit recognition with a back-propagation network,'' *Advances in Neural Information Processing Systems 2,* D. S. Touretsky, ed., Morgan Kaufmann, *San Mateo, CA,* 1990, pp. 396–404.
15. E. Sackinger, et al. ''Application of the ANNA neural network chip to high-speed character recognition,'' *IEEE Trans. Neural Networks,* 498–505, 1992.
16. R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision: Volume I,* Addison-Wesley, Reading, MA, 1992.
17. Y. LeCun, et al., ''Backpropagation applied to handwritten zip code recognition,'' *Neural Comput.,* **1,** 541–551, 1989.
18. R. A. Jacobs, ''Increased rates of convergence through learning rate adaptation,'' *Neural Networks,* **1**(4), 295–308, 1988.
19. M. Smith, *Neural Networks for Statistical Modeling,* Van Nostrand Reinhold, New York, 1993.
20. M. Azam, H. Potlapalli, J. A. Janet, and R. C. Luo, ''Outdoor landmark recognition using segmentation, fractal model and neural networks,'' *Proc. ARPA Image Understanding Workshop,* 1996.
21. T. P. Vogl, et al. ''Accelerating the convergence of the back-propagation method,'' *Bio. Cybern.,* 256–264, Sept. 1988.